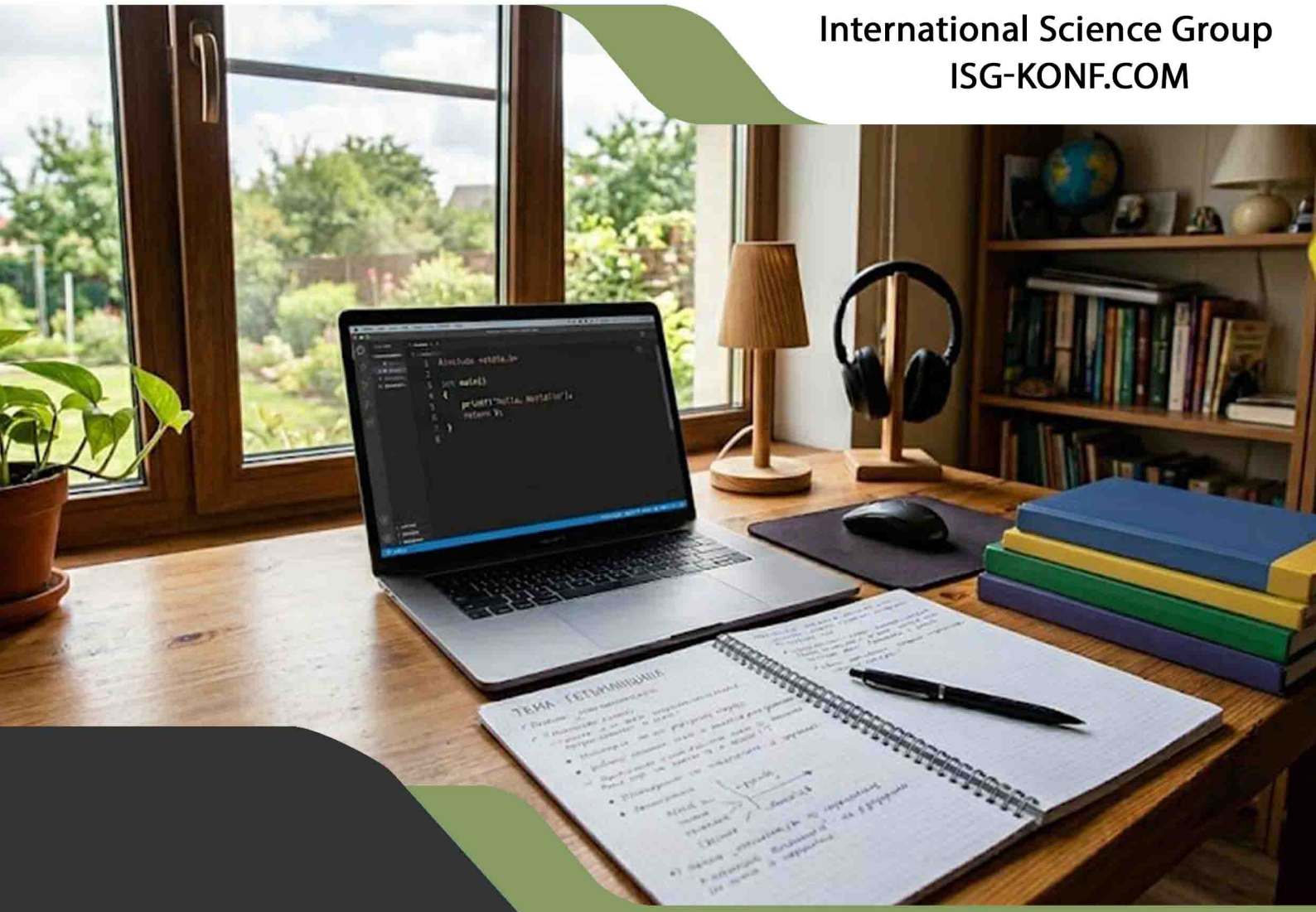
International Science Group
ISG-KONF.COM



SCIENTIFIC APPROACHES TO THE DEVELOPMENT OF IT TECHNOLOGIES

ISBN 979-8-90383-421-1
DOI 10.46299/979-8-90383-421-1

SCIENTIFIC APPROACHES TO THE DEVELOPMENT OF IT TECHNOLOGIES

Monograph

2026

UDC 004

Author:
Kolomiitsev O., Holubnychyi D., Tretiak V., Fedorchenko V., Voronin V., Bulba S., Dmitriiev O., Hulevych M., Katunin A., Nosko S., Pustovarov V., Rohulia O., Rudakov I.

Editor:
Edited by **Kolomiitsev O.V.**, Doctor of Technical Sciences, Professor, Honored Inventor of Ukraine, Professor in the Department of Computer Engineering and Programming, National Technical University is the «Kharkiv Polytechnic Institute», ORCID ID: 0000-0001-8228-8404

Reviewers:
Oleksandr Mozhaiev – Doctor of Technical Sciences, Professor, Department of Cyber Security and DATA Technologies Educational and Research Institute No. 5 Kharkiv National University of Internal Affairs

Heorhii Kuchuk – Doctor of Technical Sciences, Professor, Professor of Computer Engineering and Programming Department at the Educational and Scientific Institute of Computer Science and Information Technologies, National Technical University "Kharkiv Polytechnic Institute"

Nosko S., Kolomiitsev O., Bulba S., ets. Scientific approaches to the development of it technologies. Monograph. – International Science Group. Primedia eLaunch, Boston, USA, 2026. – 304 p.

Library of Congress Cataloging-in-Publication Data

ISBN – 979-8-90383-421-1
DOI – 10.46299/979-8-90383-421-1

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, distributed, or transmitted, in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher. The content and reliability of the articles are the responsibility of the authors. When using and borrowing materials reference to the publication is required.

UDC 004

INTRODUCTION

Information technologies in the 21st century have become a determining factor in the development of global society, transforming almost all spheres of human activity - from the economy, industry and education to public administration, defense and social communications. The current stage of development is characterized by the transition from traditional information systems to complex, intellectually oriented digital ecosystems that integrate computing, network and cognitive components.

The rapid growth of data volumes, the development of distributed computing, the emergence of new architectures and paradigms (cloud, fog, edge computing), as well as the active implementation of artificial intelligence technologies form new requirements for the design, operation and optimization of computer systems and networks. In these conditions, the issues of scalability, fault tolerance, energy efficiency, security and adaptability of information systems become particularly relevant.

At the same time, the development of information technologies is accompanied by the emergence of new challenges and risks. In particular, the number of cyber threats is increasing, information protection mechanisms are becoming more complicated, and problems arise in ensuring confidentiality and data integrity. Particular attention is drawn to the ethical and legal aspects of the use of artificial intelligence, in particular the issues of liability for automated decisions, transparency of algorithms and prevention of bias.

Modern research in the field of information technology is interdisciplinary in nature and covers a wide range of areas: from the theoretical foundations of computing and algorithms to applied problems in building intelligent systems, cyber-physical environments, the Internet of Things, high-performance computing platforms and distance learning systems. An important aspect is also the development of new technologies, such as quantum computing, neuromorphic systems, bioinspired algorithms and cognitive networks, which open up new horizons for solving complex scientific and technical problems.

The collective monograph, which is offered to the attention of readers, is dedicated to the study of modern directions of information technology development, the analysis of existing approaches and methods, as well as the development of new models and solutions in the field of computer systems and networks. The work considers both fundamental aspects and practical issues of implementing information technologies in various industries.

Particular attention is paid to the issues of optimization, intellectualization and automation of data processing processes, as well as the application of mathematical methods, in particular integer linear programming problems, to increase the efficiency of the functioning of complex information systems. The proposed approaches are aimed at solving current problems of resource management, routing, load distribution and security.

The monograph combines the results of research by the author team, reflecting the current state and prospects for the development of information technologies. It can be useful for scientific and pedagogical workers, postgraduate students and students of technical specialties, as well as for specialists involved in the development and practical application of issues of automation of data processing processes, mathematical methods, etc., in the field of information technologies.

Thus, the presented study is aimed at forming a holistic vision of modern information technologies as a complex, dynamic and integrated system of knowledge that determines the further development of digital civilization.

TABLE OF CONTENTS

1.	Nosko S.[1], Kolomiitsev O.[1], Bulba S.[1] MODELS AND METHOD OF ADAPTIVE REQUEST PROCESSING FOR A SIDECAR COMPONENT IN DISTRIBUTED MICROSERVICE SYSTEMS [1]National Technical University «Kharkiv Polytechnic Institute»	7
2.	Holubnychyi D.[1, 2, 3] INTELLIGENT OPTIMIZATION OF HIGH-CONCURRENCY DISTRIBUTED SYSTEMS [1] Department of information system, Kharkiv, Simon Kuznets Kharkiv National University of Economics [2] Department of Computer Science, Kharkiv, Kharkiv National University of Radio Electronics [3] Department of Information Technologies and Electrical Systems, Kharkiv, Ivan Kozhedub Kharkiv National Air Force University	56
3.	Kolomiitsev O.[1], Hulevych M.[1] A METHOD OF TEST CASE FORMATION FOR C++ LIBRARIES BASED ON A Q-LEARNING AGENT [1] National Technical University “Kharkiv Polytechnic Institute”	118
4.	Kolomiitsev O.[1], Rudakov I.[1], Katunin A.[2], Pustovarov V.[2], Dmitriiev O.[3] METHOD TO IMPROVE THE SAFETY OF UNMANNED AERIAL VEHICLE FLIGHTS IN URBAN AREAS BASED ON MACHINE LEARNING TECHNOLOGIES [1] National Technical University "Kharkiv Polytechnic Institute" [2] Ivan Kozhedub Kharkiv National Air Force University [3] Scientific Research Institute of Armament and Military Equipment Testing and Certification	163
5.	Tretiak V.[1], Voronin V.[1], Rohulia O.[1] RESULTS OF EXPERIMENTAL STUDY OF INTEGER LINEAR PROGRAMMING ALGORITHMS WITH BOOLEAN VARIABLES IN MILITARY APPLICATIONS PROBLEM SOLVING [1] Ivan Kozhedub Kharkiv National Air Force University	196

6.	Fedorchenko V.[1, 2] METHODS OF AUTOMATION OF SOFTWARE DEVELOPMENT PROCESSES FOR THE INTERNET OF THINGS BASED ON ARTIFICIAL INTELLIGENCE [1] Department of Electronic Computers, Kharkiv National University of Radio Electronics [2] Department of Information Systems, Simon Kuznets Kharkiv National University of Economics	221
	REFERENCES	292

DOI: 10.46299/979-8-90383-421-1.1

1. Models and method of adaptive request processing for a sidecar component in distributed microservice systems

Introduction. Modern distributed microservice systems (DMS) are actively being implemented in corporate environments. According to research by Gartner, 74% of organizations utilize microservice architecture [8]. Furthermore, the microservice technology market is demonstrating rapid growth, with a projected 12.7% increase in DMS adoption by 2033, according to studies by the IMARC Group [14]. Existing approaches to managing interaction between microservices via centralized components create significant limitations in request processing efficiency [17, 38].

DMS face complex challenges in managing inter-service interaction, including authorization, monitoring, load balancing, and error handling, which are typically addressed through centralized components with limited scalability. The Sidecar pattern serves as an architectural solution for decentralizing these auxiliary functions. However, existing sidecar components exhibit shortcomings in adaptability and limited application variability. Known sidecar solutions increase RAM consumption by 40-60%; in a DMS with 100 microservices, this results in an additional 20 GB to 50 GB of RAM usage. Such solutions lead to a 15-25% increase in request processing time for the sidecar component in DMS due to additional proxy layers [5, 31].

Unadapted sidecar components can consume between 500 MB and 1 GB of RAM, complicating the scalability and stable operation of the DMS architecture [31]. Moreover, existing sidecar components function primarily as static proxy solutions with limited extensibility, failing to meet the requirements of modern dynamic DMS which demand adaptive processing of diverse request types [5, 23, 25].

The relevance of developing adaptive mathematical models and a method with automated decision-making for the sidecar component is driven by the inability of existing static approaches to ensure effective operation in the dynamic conditions of modern DMS. Static caching algorithms do not account for the variable activity of tenants in multi-tenant environments, leading to reduced request processing efficiency. Fixed routing rules do not adapt to the current service load and tenant isolation

requirements, causing uneven request distribution. The classic backpressure mechanism is based on fixed thresholds, whereas modern DMS are characterized by variable loads ranging from 100 to 1000+ requests per second [37] and unstable network conditions [20, 39].

Ensuring high efficiency of the sidecar component with minimal CPU and RAM consumption under such conditions requires the development of adaptive decision-making mechanisms capable of dynamically reacting to changes in the DMS state. This justifies the necessity of developing both adaptive mathematical models and a method for adaptive request processing for the sidecar component.

Main Body. The Sidecar pattern is an effective solution for overcoming the primary challenges of microservice architecture. This architectural approach allows for moving general tasks outside the business logic of microservices, ensuring their centralized and unified management. Among the key tasks that sidecar components must solve, attention is focused on HTTP request processing, thanks to a multi-layered architecture that allows for the processing of headers, tokens, user access rights, and resource requests.

The basis of the sidecar pattern is the principle of separation of concerns, which allows for a clear division between business logic and infrastructural tasks. The sidecar component is deployed alongside the main microservice within a single deployment unit (e.g., in a Kubernetes pod), but performs exclusively auxiliary functions: network traffic proxying, authentication and authorization, logging, monitoring, service discovery, load balancing, and failure prevention.

A key advantage of the sidecar pattern is its transparency to the main microservice. The business code requires no modifications for integration with the sidecar component – all network traffic is automatically intercepted and processed by the auxiliary component. This allows for the implementation of complex infrastructural logic without impacting the microservice code and ensures backward compatibility with existing information systems (IS).

The sidecar component functions as an intelligent proxy, providing additional functionality at the network interaction level. It can perform automatic request retries

during temporary failures, implement exponential backoff to avoid microservice overload, perform request transformations, and more.

Effective design of sidecar components is based on adherence to fundamental principles that ensure their functionality and performance in complex distributed environments.

The principle of minimal impact implies that the sidecar component must add minimal latency to the overall IS efficiency. This approach requires optimization of CPU and RAM consumption, minimization of network delays, and ensuring high request processing throughput. The sidecar component must not become a bottleneck in the IS, even under peak loads.

The principle of operational transparency means that the sidecar component must function invisibly to end-users and client applications. All requests served by the sidecar component must preserve the semantics of the original APIs, must not alter request and response formats (except for explicit transformations), and must ensure consistent error handling.

The principle of configuration flexibility allows for adapting the behavior of the sidecar component to the specific requirements of different microservices without code changes. Behavior adaptation includes parameterization of timeout values, retry policies, monitoring configurations, and support for hot reloading to update parameters without restarting components.

The principle of failure isolation is based on the premise that problems within the sidecar component must not affect the functioning of the main microservice, and vice-versa. This is achieved through resource management during failures.

Analysis of selecting the optimal framework for sidecar component implementation

Based on the analysis of experimental studies presented in the literature, a comparison of frameworks was conducted using metrics such as startup time, RAM consumption, and garbage collection pauses. Testing was performed on standardized hardware under identical conditions.

The testing results indicate the superiority of the Quarkus framework across all

key metrics [7, 45] (Table 1).

Table 1

Comparison of Quarkus with other frameworks

Framework	Startup Time (ms)	Memory (Native MB)	GC Pauses (ms)	Reactivity Support	Cloud Environment Optimization	Community
Spring Boot	47	74	23-45	Partial	Average	Largest
Micronaut	28	42	8-12	High	High	Smaller than Spring
Quarkus	13	21	2-3	High	Maximum	Growing

Thanks to its effective support for microservice architecture, optimization for the Kubernetes platform, and reactive approach, the Quarkus framework ensures an optimal balance between request processing speed, scalability, and flexibility [18, 37].

Thus, the Quarkus framework is the optimal solution for implementing the sidecar component. It possesses the following advantages:

– It ensures high request processing speed while maintaining the flexibility of solutions based on the Java programming language for implementing complex logic;

– It allows, via native compilation with GraalVM, for RAM consumption of 20-30 MB and a startup time of 12-13 ms [35], which is critical for sidecar components that double the number of processes in the IS;

– It ensures, when using Vert.x, non-blocking I/O operations and high multi-threading, which is necessary for effective network traffic processing in sidecar components;

– It uses standard Java libraries, frameworks, and tools, lowering the barrier to entry for software development;

– It optimizes software deployment in the Kubernetes platform and containerized environments.

Scalability evaluation is performed using the formula [43]:

$$S(n) = T_1/T_n,$$

where T_1 is the execution time on a single core, and T_n is the execution time on n cores.

Scaling efficiency:

$$E(n) = S(n)/n.$$

For the Quarkus framework, $E(8) \approx 0.87$, according to [13], indicating linear scaling up to 8 CPU cores.

Analysis of deployment modes for the Quarkus framework for the sidecar component

Two main approaches can be used to deploy a sidecar component based on the Quarkus framework: Native Image (GraalVM Native Image) and the standard Java image (JVM).

The native image is compiled using the GraalVM Native Image platform [13], creating a standalone executable file without the need for a JVM during runtime. This approach provides the following advantages: IS startup time is reduced by 18% (from 4.4 s to 3.6 s), Docker image size is reduced by 67% (from 714 MB to 238 MB), and RAM consumption is reduced by 69%.

The scalability benefits of the native approach increase when deploying DMS in large clusters. Fig/ 1 presents RAM consumption for a typical cluster ranging from 10 to 100 pods. When deploying 100 instances of the sidecar component, total RAM consumption is 6.4 GB for the Java version versus 1.98 GB for the native version, ensuring a release of 4.42 GB of RAM.

Thus, the native mode can be selected as the base deployment platform for the sidecar component, providing the necessary foundation for the improvement and development of mathematical models and algorithms.

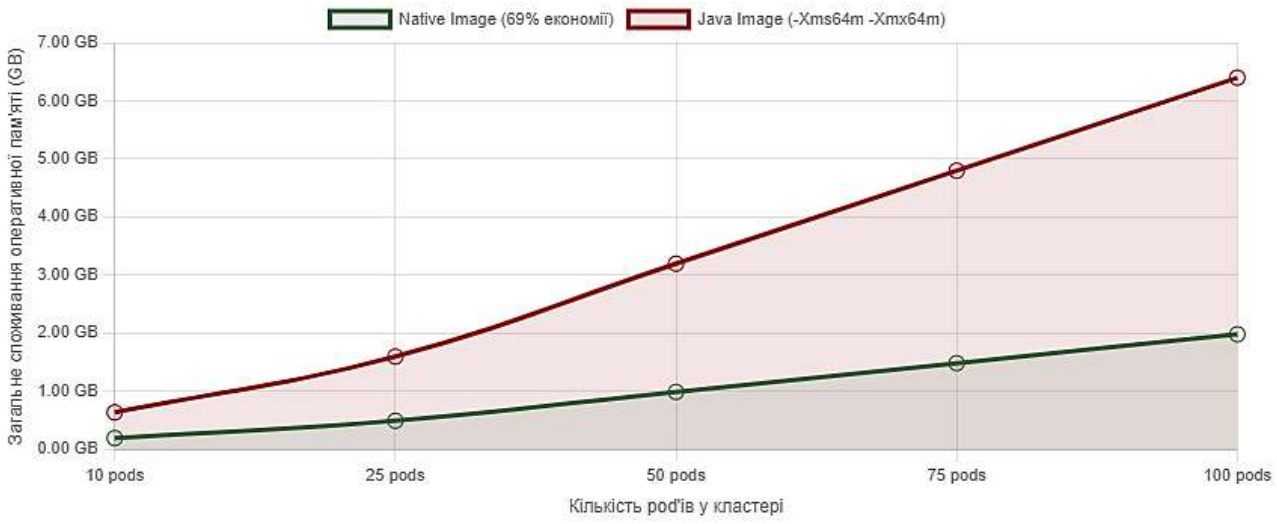


Figure 1. RAM consumption in the cluster

To optimize the operation of the sidecar component, a comparative analysis of various algorithms for garbage collection (GC) was conducted.

Based on the results of the experimental comparison of garbage collectors (Table 2), G1 GC was selected, ensuring pauses of <10 ms in 90% of cases with CPU consumption of 7-15%, which is optimal for low-latency sidecar components.

The choice of the Kubernetes platform for orchestrating DMS operation is based on its capability for automated deployment, scaling, and management of containerized applications, as well as support for the pod concept [32].

Figure 2 shows the proposed deployment scheme of the sidecar component together with the microservice, and Figure 3 shows an example of a real application of a container with the sidecar component in a single pod within the Kubernetes platform.

Table 2

Experimental comparison of garbage collector efficiency

Garbage Collector	Longest Pause	Majority of Pauses	Total Pauses (5-10 min)	Average CPU	Justification for Choice
Serial GC	59-89 ms	10-40 ms	300 ms – 3 s (per 5 min)	3-5%	Unacceptable pauses for high-speed sidecar components. Blocks all processing threads

Continuation of table 2

G1 GC	7.5-45 ms	4-10 ms	0.1 s – 4 s (per 10 min)	7-15%	Optimal choice: low pauses (< 10 ms for 90% of cases), effective operation with heap 64-160 MB
ZGC	78 µs – 0.5 ms	40-100 µs	2–100 ms	20–30%	Lowest pauses, but high CPU consumption may limit sidecar component throughput

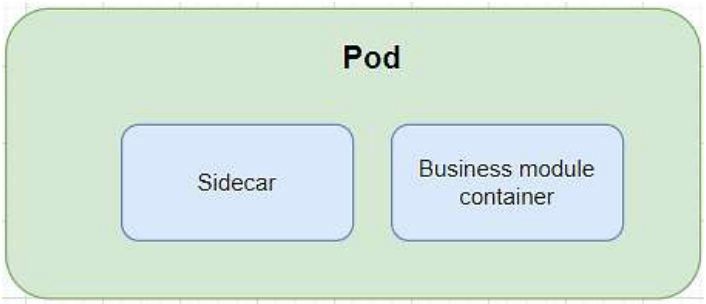


Figure 2. Deployment scheme of a pod with a sidecar component

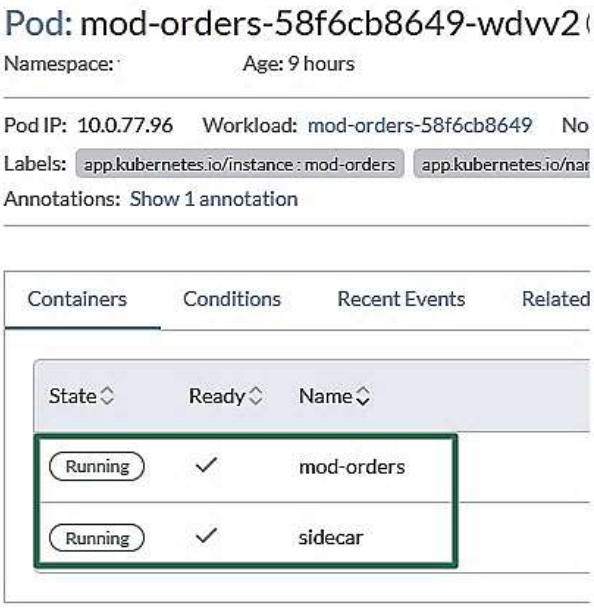


Figure 3. Microservice container with sidecar component in a single pod

Placing the sidecar component and the microservice in the same pod ensures shared management of the lifecycle and network space.

Figure 4 illustrates the interaction scheme of the Kong API Gateway with microservices utilizing sidecar components. Sidecar components are used for routing calls within the DMS without involving the Kong API Gateway, allowing for the

optimization of internal connections between microservices.

A feature of the proposed architecture is that the Kong API Gateway acts solely as an API gateway, while authorization functions are delegated to the sidecar components. This solution aligns with Domain-Driven Design (DDD) principles, where each component has a clearly defined area of responsibility [6].

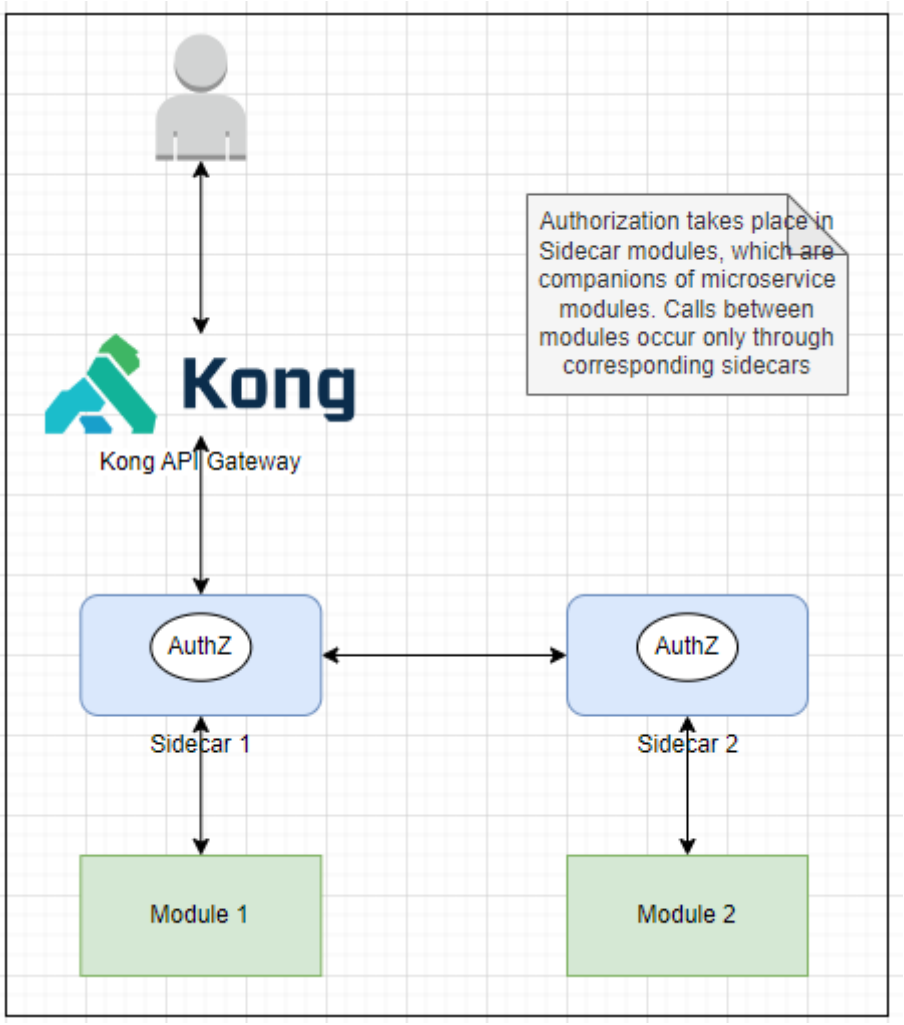


Figure 4. Interaction of microservices using sidecar components

Development of an adaptive filter structure for processing incoming requests to the sidecar component

The functionality of request authorization, validation, logging, and monitoring in each microservice individually leads to code duplication, complicates maintenance, and increases the probability of security errors in the DMS [14, 24, 42].

Based on the analysis of known architectural approaches to request processing,

it has been established that the Chain of Responsibility pattern is the optimal solution for use in the sidecar component. This pattern allows for centralized, efficient, and extensible processing of HTTP requests within the sidecar component without violating the Open/Closed Principle. It provides the ability to extend request processing functionality without modifying existing code. The Chain of Responsibility pattern belongs to the classic GoF patterns [12] and implements request processing where each handler can either process the request or pass it to the next handler in the chain.

An architecture for multi-level HTTP request processing in the sidecar component is proposed, based on the sequential application of developed filters, each responsible for a specific part of the processing: detection of cyclic calls to the sidecar component, JWT validation, protection of tenant access from others, RBAC (Role-Based Access Control) authorization, and DMS metric collection. This approach ensures a separation of responsibility between filters and allows them to be tested, optimized, and extended independently. Input request filters in the sidecar component process HTTP requests before transmitting them to the target microservice.

An incoming request is an HTTP request arriving at the sidecar component from an external client (a sidecar component of another microservice or a user). It is the primary request that must pass through all filters before being directed to the main microservice. The request body remains unchanged; only the headers are modified.

After processing the incoming request, the sidecar component forms an outgoing request and directs it to the target microservice using the forwardRequest() function.

Figure 5 shows the developed diagram of incoming and outgoing HTTP request processing by the sidecar component. Figure 6 shows the developed sequence diagram of all filters processed by the sidecar component for incoming HTTP requests.

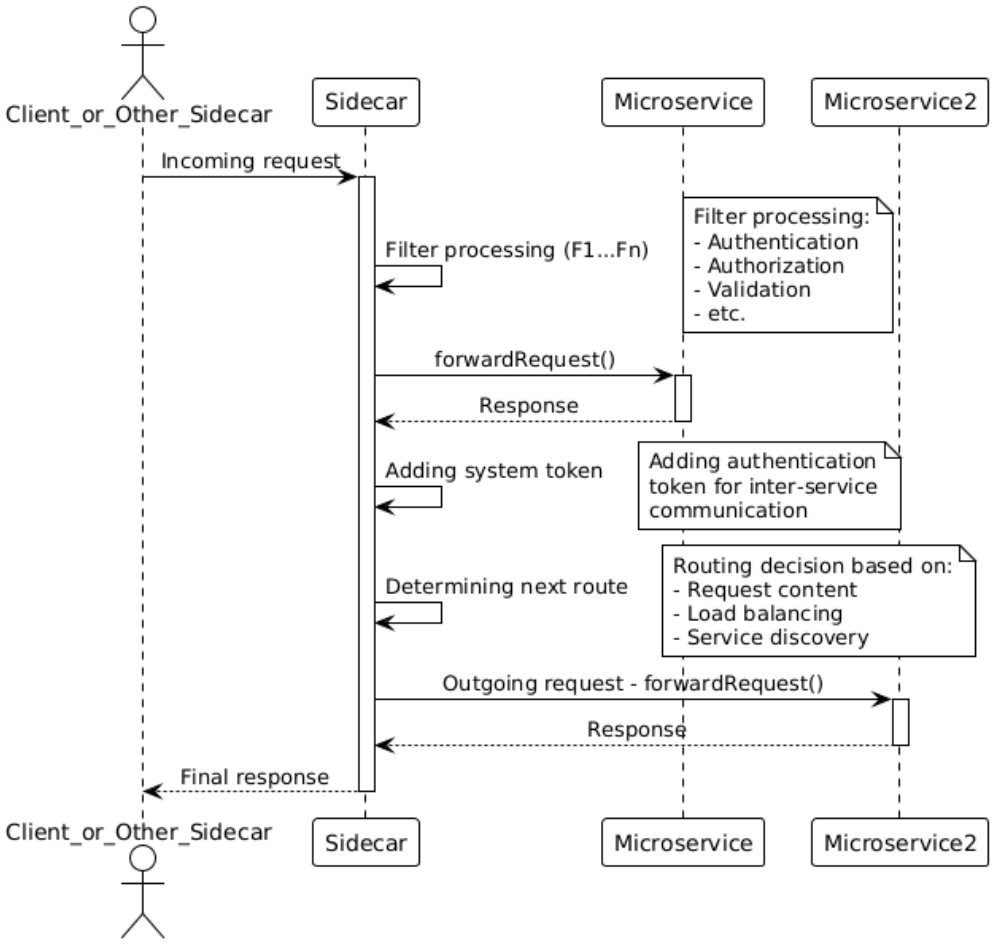


Figure 5. Processing of incoming and outgoing HTTP requests

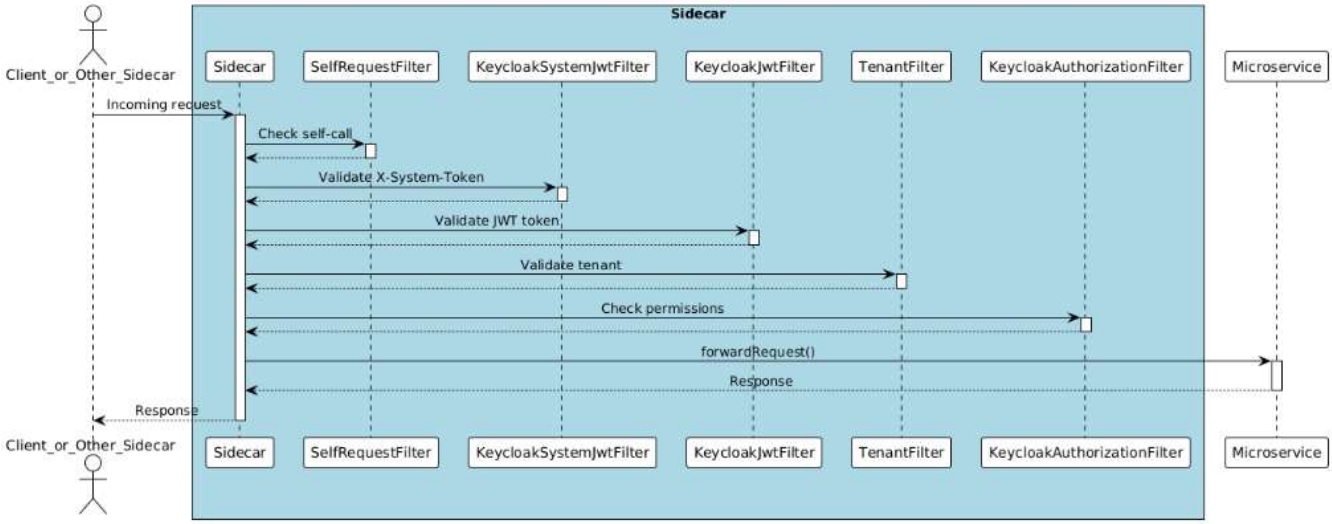


Figure 6. Description of filters in incoming HTTP request processing

Mathematically, this pattern can be represented as a set of functions in the category of HTTP transformations.

An incoming HTTP request is defined as a tuple:

$$R_{in} = (M, U, H_{in}, B_{in}), \quad (1)$$

where M is the HTTP method of the request (e.g., GET, POST); U is the request URI; H_{in} is the set of input headers; B_{in} is the request body.

An outgoing HTTP request can be defined as follows:

$$R_{out} = (M, U', H_{out}, B_{out}), \quad (2)$$

where U' is the transformed URI; H_{out} are the modified headers; $B_{out} = B_{in}$ – the request body remains unchanged (principle of request body idempotence).

The set of filters can be represented as a sequence of functions F_1, F_2, …, F_n, which process the request data R_{in} and modify headers or ensure verification.

The main properties of the filter set include:

- Associativity: $(F_3 \circ F_2) \circ F_1 = F_3 \circ (F_2 \circ F_1)$;
- Identity: $\exists I: F \circ I = I \circ F = F$;
- Monotonicity: if $F_1 \subseteq F_2$, then $F_1(R) \subseteq F_2(R)$;

To calculate request processing by filters, the following mathematical expression can be used:

$$R_{out} = F_n\left(F_{n-1}\left(F_{n-2}(\dots F_1(R_{in}) \dots)\right)\right), \quad (3)$$

where F_i is a function representing a specific filter operation (e.g., authorization, URL normalization, or header processing); R_{out} is the request after passing through all filters.

The total time for request processing by a filter is defined as the sum of execution times for all its operations:

$$T_F = \sum_{i=1}^{n} T(f_i), \quad (4)$$

where T_F is the total time for request processing by the filter; n is the number of operations in the filter; $T(f_i)$ is the time required to execute operation f_i.

Time complexity:

$$O(n * logk), \quad (5)$$

where n is the number of filters in the sidecar component, k is the size of the incoming request context.

The spatial complexity of request processing by a filter can be determined as follows:

$$O(|H| + |T|), \quad (6)$$

where $|H|$ is the number of headers in the request, $|T|$ is the number of tokens in the request.

The mathematical description for calculating request processing using a set of filters is an effective approach for adapting the sidecar component, as each filter has clearly defined actions that can be formalized via a mathematical function. This approach allows for:

– separating the processing flow into independent components to simplify maintenance and modification;

– determining request processing time and optimizing RAM and CPU consumption;

– formalizing the relationships between filters for scalable efficiency calculation.

Caching of authorization tokens in the KeycloakAuthorizationFilter

Experimental studies checking the number of requests from sidecar components to the Keycloak authorization server revealed an increased load on the DMS:

specifically, 132,239 requests in 17 minutes (Figure 7) make the Keycloak authorization server a limiting factor for increasing DMS throughput.

To increase DMS throughput, adaptive caching of authorization tokens with weighting coefficients [21] is applied in filter F_5. This allows for reducing the load on the Keycloak authorization server by 70% (from 132,239 to ~40,000 requests) and increasing the authorization token cache hit rate.

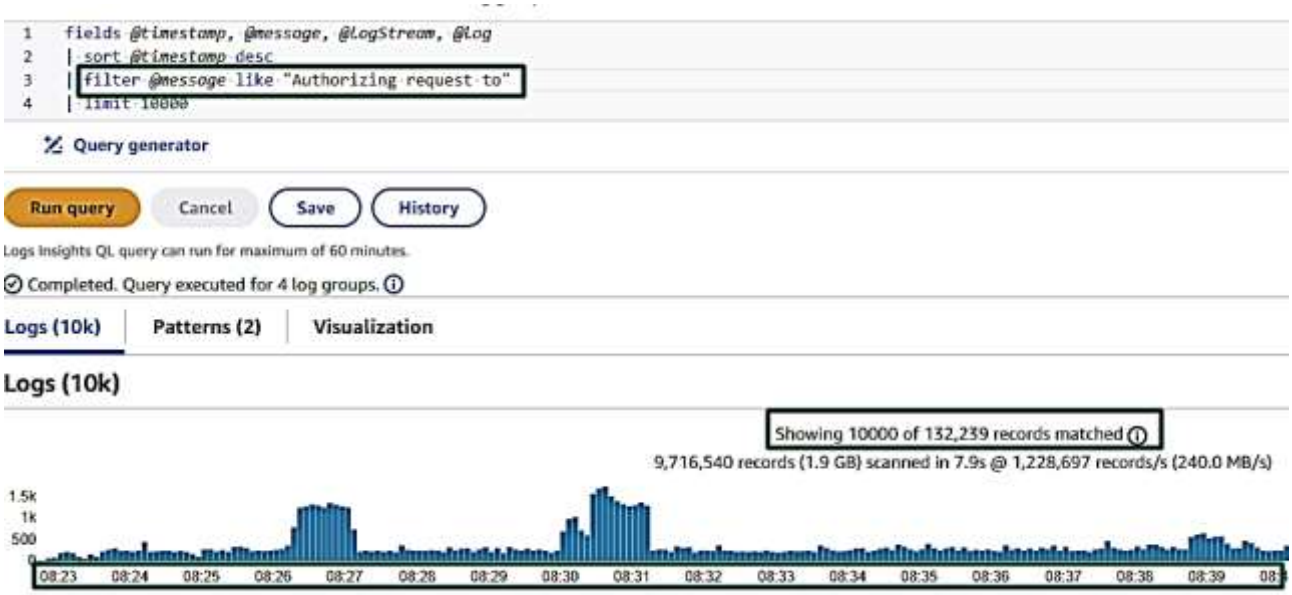


Figure 7. Load on the Keycloak authorization server over 17 minutes

Reducing the number of requests to the Keycloak authorization server leads to decreased traffic in the DMS network, improving its throughput.

Figure 8 shows the sequence diagram of filter F5 using the cache.

Caffeine [29] was selected as the caching library. It uses the Window TinyLFU algorithm, which combines frequency and recency analysis of cache access to achieve an optimal balance between hit rate and RAM consumption. The Quarkus framework used to implement the sidecar component provides built-in support for the Caffeine library. The Caffeine caching library supports RAM management mechanisms and an asynchronous API for cache operations [30].

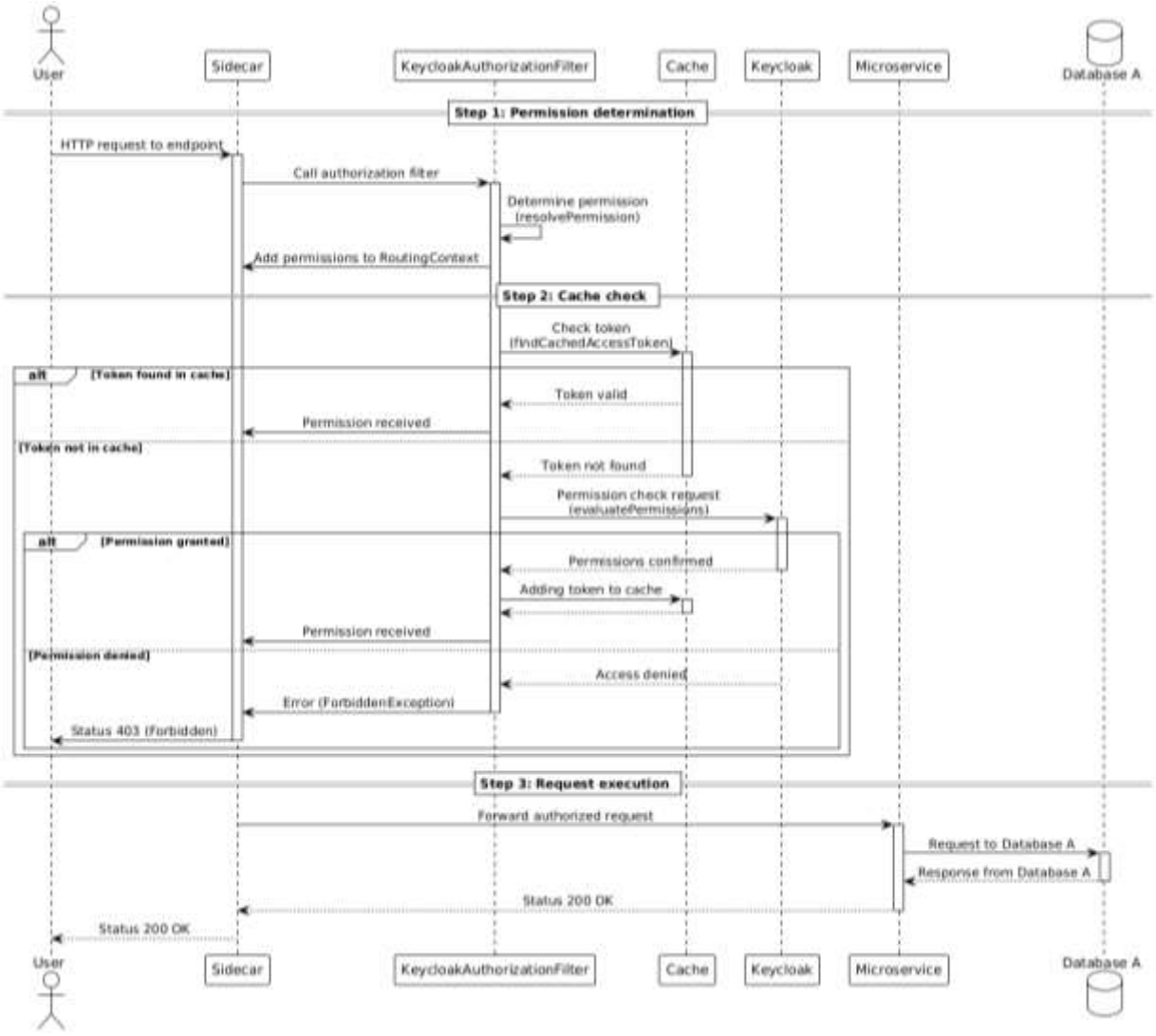


Figure 8. Sequence diagram of filter F_5 with cache

The initial code variant for creating the cache to store authorization tokens can be represented as follows:

```
@ApplicationScoped
public Cache<String, JsonWebToken>
kcAuthorizationCache(KeycloakProperties props, JsonWebTokenExpiry expiry)
{
  return Caffeine.newBuilder()
    .expireAfter(expiry)
    .initialCapacity(10)
    .maximumSize(props.getAuthorizationCacheMaxSize())
    .removalListener((k, jwt, cause) -> log.debug("Cached access token
removed: key={}, cause={}", k, cause))
    .build();
}
```

The main cache configuration parameters include:

– expireAfter – sets the time during which an object in the cache is considered valid;

– initialCapacity – sets the initial cache size (10 objects), reducing RAM allocation operations when adding the first data;

– maximumSize – limits the maximum number of objects that can be stored in the cache simultaneously;

– removalListener – adds an event handler for processing cache element removal events.

These parameters ensure basic caching functionality. Based on the initial implementation, an adaptive mathematical model of the cache with weighting coefficients is developed, which increases the cache hit rate under variable load conditions in multi-tenant DMS.

Thus, the adaptive structure of the developed HTTP request processing filters creates the basis for the improvement and development of mathematical models and a method for request processing.

Improvement of the mathematical model of adaptive data caching for the sidecar component

Known caching algorithms (LRU, LFU, ARC [2]) do not account for the dynamic operating conditions of the DMS in which the sidecar component functions, nor for the variability of user behavior patterns. To increase the cache hit rate in the sidecar component, adaptability is necessary – the ability of the algorithm to adjust to current loads, changing access scenarios, and DMS environment specifics.

For the sidecar component, it is necessary to apply construction principles that impose less load on the surrounding DMS infrastructure, allowing for adaptive caching without data stream blocking and manual RAM management. Creating such a caching component, designed to work with limited RAM volume, will allow for dynamically changing the cache size depending on the DMS load. The cache size changes depending on the load on the DMS, allowing for reduced RAM consumption during low system load and increased cache size as load increases.

To determine the dynamic cache size, the following formula can be used:

$$Cache_{maxSize}(t) = \min\left(BaseSize * \left(1 + \alpha * Load(t)\right), MaxLimit\right), \quad (7)$$

where *BaseSize* is the base cache size; α is the adaptation coefficient (established experimentally); *Load(t)* is the load level of the sidecar component at time t (e.g., request frequency); *MaxLimit* is the maximum permissible cache size.

Applying a dynamic approach to determining cache size will increase its operational efficiency and ensure the adaptability of RAM consumption by the sidecar component.

To control the weight of a cache element (for data eviction), a weighting mechanism can be used. Unlike known algorithms such as LRU or LFU, this is not limited to considering only one factor (time of last use or access frequency) but allows for considering multiple criteria based on dynamic metrics of the sidecar component's operation simultaneously. Such criteria ensure cache adaptability according to the current DMS state. The use of a weighting mechanism ensures flexibility and adaptability in data eviction. The formula for calculating cache element weight has been improved as follows:

$$w_i = \beta \cdot Frequency_i + \gamma \cdot \frac{1}{\mathrm{T}imeSinceLastAccess_i}, \quad (8)$$

where $Frequency_i$ is the number of requests to authorization token *i*; $TimeSinceLastAccess_i$ is the time elapsed since the last use of authorization token *i*; β, γ are weighting coefficients (determining the degree of influence of each factor).

Using (8) in mechanisms for determining cache element weight allows for retaining the most important authorization tokens in the cache – those that are primarily accessed or have been recently used. The mechanism for determining cache element weight is implemented using (8) in the weigher() function as follows:

```
.weigher((String key, JsonWebToken token) -> {
    int frequency = getUsageFrequency(token);
    long timeSinceLastAccess = getTimeSinceLastAccess(token);
    int frequency = getUsageFrequency(token);
    double beta = ...;  // значення β
    double gamma = ...; // значення γ
    double weight = beta * frequency + gamma / (timeSinceLastAccess + 1);
    return (int) weight;
});
```

To reduce RAM volume consumption, mechanisms for automatic and manual invalidation of authorization token cache data are proposed. Automatic cache invalidation is implemented by setting the token lifetime expireAfter(expiry). Additionally, a mechanism for reacting to special events, such as user logout, is proposed, which invalidates the authorization token cache. The source code for handling such events can be written as follows:

```
  .removalListener((k, jwt, cause) -> {
     if (cause == RemovalCause.EXPIRED || cause == RemovalCause.SIZE) {
       log.debug("Token {} removed due to {}", k, cause);
       handleTokenExpiry(k);
     }});
```

To evaluate the efficiency of the authorization token cache, it is proposed to collect several key metrics. Calculation of cache efficiency can be performed using the following formula:

$$Cache_{efficiency} = \frac{Hits}{Total_Requests}, \quad (9)$$

where *Hits* is the number of cache hits; *Total_Requests* is the total number of requests processed by the filter: *Hits* + *Misses*.

For example, if *Hits* = 70 and *Total_Requests* = 100, cache efficiency can be calculated as follows:

$$Cache_{efficiency} = \frac{70}{100} = 0{,}7.$$

Cache efficiency allows for evaluating how effectively the developed caching mechanism functions under real DMS operating conditions.

The source code for the implementation of the authorization token cache in the sidecar component consists of a combination of dynamic cache size management, a weighting mechanism for calculating cache element weight, and data invalidation mechanisms:

```
@ApplicationScoped
public Cache<String, JsonWebToken> kcAuthorizationCache(KeycloakProperties props,
JsonWebTokenExpiry expiry) {
    return Caffeine.newBuilder()
        .expireAfter(expiry)
        .weigher((String key, JsonWebToken token) -> {
            int frequency = getUsageFrequency(token);
            long timeSinceLastAccess = getTimeSinceLastAccess(token);
             double weight = beta * frequency + gamma /     (timeSinceLastAccess + 1);
             // Add 1 to avoid division by zero
             return (int) weight;
        }) // Weighting mechanism for prioritizing data in the cache
        .maximumWeight(10_000) // Maximum allowable weight value
        .removalListener((k, jwt, cause) -> {
             if (cause == RemovalCause.EXPIRED || cause ==    RemovalCause.SIZE) {
                log.debug("Token {} removed due to {}", k, cause);
                handleTokenExpiry(k);
             }
        }) // Tracking data removal via events
        .build();
}
```

The proposed dynamic cache size management, depending on the sidecar component load, allows for optimizing RAM volume consumption during low load and

ensuring high speed of the sidecar component during high DMS load.

In the caching mechanism under real DMS operating conditions, the Window TinyLFU algorithm (Caffeine library [40]) is used with (8), which determines the importance of authorization tokens based on access frequency $Frequency_i$ and time since last access $TimeSinceLastAccess_i$ using weighting coefficients β and γ.

In multi-tenant DMS, load asymmetry exists between tenants, creating additional challenges for known caching algorithms [35]. Known caching algorithms (LRU, LFU, CLOCK) [29, 31] do not account for the group activity of users of a single tenant – that is, the situation where users of a tenant send requests to it during specific time intervals. Therefore, it is necessary to optimize the distribution of cache RAM volume M among the set of tenants $T = \{t_1, t_2, ..., t_n\}$ with different access patterns, similar to approaches in [10, 29], which will allow for reducing the total number of cache misses in the sidecar component while maintaining fairness between tenants.

The basis of the improved mathematical model for calculating cache element weight is formula (8), which determines the priority of an element for retention in the cache. Different caching algorithms use different mathematical models for weight calculation. Known mathematical models for cache element weight calculation in caching algorithms most often used in modern DMS include:

–mathematical model of LRU:

$$w_{\text{i}} = \frac{1}{TimeSinceLastAccess_i},$$

where $TimeSinceLastAccess_i$ is the time elapsed since the last request access to cache element (token) i;

–mathematical model of LFU:

$$w_{\text{i}} = Frequency_i,$$

where $Frequency_i$ is the number of requests to cache element (token) i over a certain

period.

Both mathematical models have their advantages and disadvantages. The LRU mathematical model works effectively when data "freshness" is important but may ignore frequently used cache elements if they haven't been requested recently. The LFU algorithm's mathematical model, conversely, retains the most requested cache elements but may be inefficient when data access patterns change.

For multi-tenant DMS, known mathematical models for calculating cache element weight in LRU and LFU algorithms do not account for the specific activity of different tenants, which can lead to irrational distribution of RAM volume in the sidecar component between tenants.

Based on multi-tenant caching research [29], the necessity of considering tenant-specific metrics has been established. Facebook CacheLib [2] and Google's warehouse-scale [28] architectures demonstrate the effectiveness of adaptive approaches in industrial environments [27].

To improve the mathematical model for determining cache element weight in multi-tenant DMS, it is proposed to consider an additional metric – tenant activity. In this model, the frequency of requests to specific tokens is supplemented by the activity metric of the corresponding tenant, allowing for the creation of a more adaptive caching mechanism that prioritizes data from more active tenants, keeping them in the cache longer. This approach increases data access speed and optimizes RAM volume usage in multi-tenant DMS.

The structure of the sidecar component filters includes separate filters that ensure the collection and use of its operation metrics:

KeycloakTenantFilter F_4 – responsible for data segregation between tenants and determining the tenant to which the request is addressed. Logic for collecting tenant activity metrics is implemented in this filter;

KeycloakAuthorizationFilter F_5 – implements authorization of incoming user requests [19, 41] and uses the cache to store authorization tokens, as well as collecting metrics on sidecar component operation, particularly load (average number of requests per unit of time) using the monitoring system [44].

Cache adaptation depends on the logic of both filters, which ensure metric collection – specifically, the load metric for adaptive cache sizing and the tenant activity metric, which allows for distributing RAM volume usage between tenants in the DMS.

To account for tenant activity in weight calculations, the following improved formula for calculating cache element weight is proposed:

$$w_i = \beta \cdot Frequency_i + \gamma \cdot \frac{1}{\mathrm{T}imeSinceLastAccess_i} + \delta \cdot \mathrm{TenantActivity}_j, \quad (10)$$

where $Frequency_i$ is the number of requests to authorization token i (with exponential smoothing); $TimeSinceLastAccess_i$ is the time elapsed since the last access to token i (with temporal locality [2]); $TenantActivity_j$ is the number of requests to the corresponding tenant j (with exponential smoothing); β, γ, δ are weighting coefficients determining the influence of access frequency, recency of last access, and tenant activity, respectively.

Authorization token usage frequency $Frequency_i$ is calculated using exponential smoothing – a time series processing method that allows for "smoothing" (averaging) the metric value so that newer events have a greater influence on $Frequency_i$, and older ones have less.

For caching, this approach means that token usage frequency considers not only the sum of all accesses but also reduces the weight of old accesses over time.

The formula for calculating $Frequency_i$ can be written as follows:

$$Frequency_i(t) = \alpha \cdot x_i(t) + (1 - \alpha) \cdot Frequency_i(t - 1),$$

where $x_i(t)$ is the number of accesses to authorization token i at time t, α is the smoothing coefficient ($0 < \alpha < 1$).

If there was a request to token i at time t, then $x_i(t) = 1$, otherwise $x_i(t) = 0$. Thus, new accesses have a greater impact on frequency, and old ones are gradually

"forgotten".

Similarly, the tenant activity level $TenantActivity_j$ is calculated using exponential smoothing, allowing consideration not only of the total number of requests to the tenant but also ensuring that newer events have a greater influence on $TenantActivity_j$, and older ones have less.

The formula for calculating $TenantActivity_j$ can be written as follows:

$$TenantActivity_j(t) = \alpha \cdot y_j(t) + (1 - \alpha) \cdot TenantActivity_j(t - 1),$$

where $y_j(t)$ is the number of accesses to tenant j at time t, α is the smoothing coefficient ($0 < \alpha < 1$).

If there was a request to tenant j at time t, then $y_j(t) = 1$, otherwise $y_j(t) = 0$. Temporal locality for the $TimeSinceLastAccess_i$ metric is a data property where recently used elements are highly likely to be used again in the near future. For caching, this means it is advisable to keep recently accessed elements in the cache as they will likely be needed again.

The work conducted an experimental study of the coefficient values for parameters in (10) using the grid search method, obtaining the following data:
$\beta \in [0.4, 0.6]$, $\gamma \in [0.3, 0.5]$, $\delta \in [0.1, 0.3]$, where $\beta + \gamma + \delta = 1$.

The grid search method consists of systematically iterating through all possible parameter combinations within given limits to select optimal values based on the criterion of maximizing the cache hit rate. The conducted sensitivity analysis showed that a ±20% change in coefficients affects the hit rate by no more than ±3%, indicating the stability of the improved mathematical model.

The improved mathematical model for calculating authorization token weight, considering tenant activity, allows for retaining important authorization tokens of more active tenants in the cache. This approach reduces the risk of accidental deletion of required data and decreases the number of requests to the Keycloak authorization server.

The function for calculating element weight in the cache, considering the

improved mathematical model of adaptive caching, can be written as follows:

```
// Approach based on TinyLFU admission policy [10]
// with adaptation for multi-tenant DMS [29]
.weigher((String key, JsonWebToken token) -> {
    long lastAccessTime = getLastAccessTime(token);
    int frequency = getUsageFrequency(token);
    String tenantId = token.getClaim("tenantId");
    int tenantActivity = getTenantActivity(tenantId);
    // modified weight formula
    double beta = props.getBeta();
    double gamma = props.getGamma();
    double delta = props.getDelta();
    long timeSinceLastAccess = System.currentTimeMillis() -lastAccessTime;
    // Add 1 to avoid division by zero
    double weight = beta * frequency
        + gamma / (timeSinceLastAccess + 1)
        + delta * tenantActivity;
    return (int) weight;
});
```

The scheme of the weighted caching algorithm using the improved mathematical model is shown in Fig. 9.

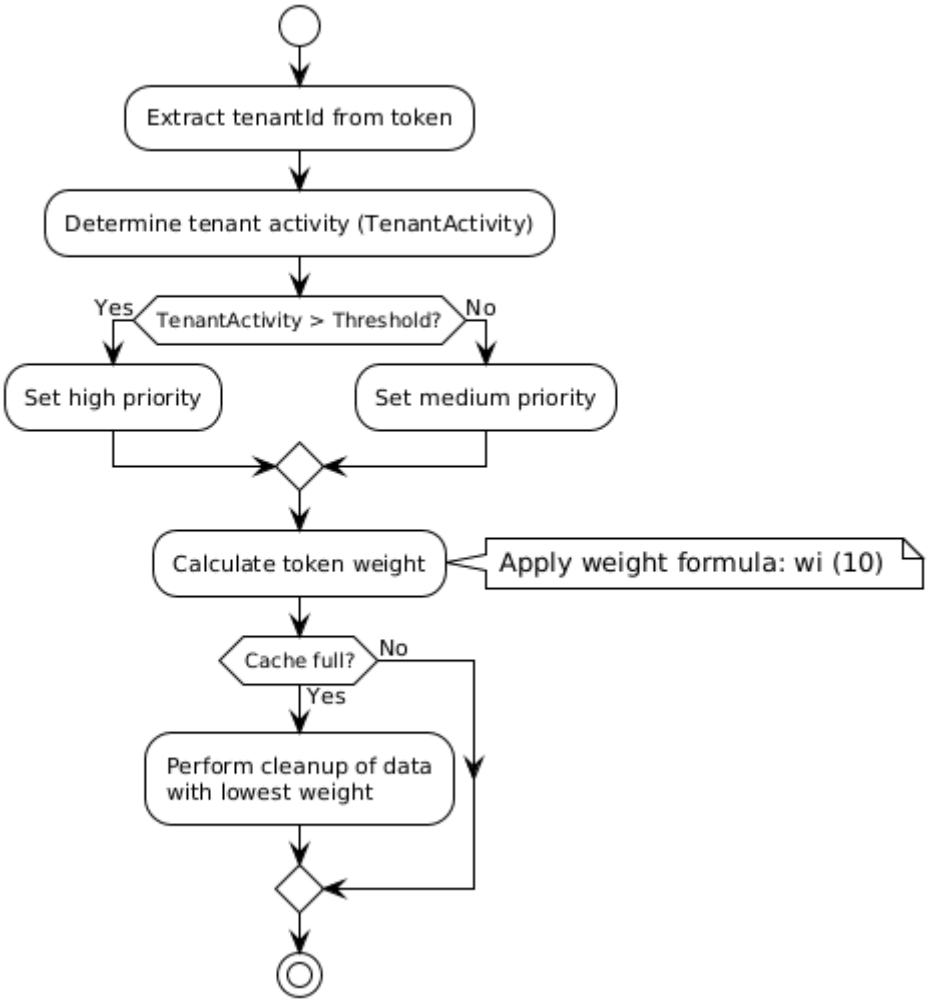


Figure 9. Scheme of the weighted caching algorithm

In the case of limited DMS deployment using only one tenant, the weighting coefficient δ is set to zero (δ=0). Under such a condition, the token weight formula in the cache (10) depends only on access frequency and time of last access, making the improved mathematical model universal for both standard and multi-tenant DMS, ensuring access balancing between tenants [29].

Thus, considering tenant activity ensures the improvement of the adaptive algorithm, which aligns with research results for multi-tenant caching [29] and practical experience in implementing industrial-scale systems [1, 15].

The improved mathematical model allows for considering the specific activity of each tenant, optimizing throughput and effective RAM volume management in multi-tenant DMS.

Using adaptive cache size based on sidecar component operation metrics makes

the caching algorithm flexible and capable of adjusting to load changes in real-time.

Based on the results of experimental testing using the CacheLib [2] and Kangaroo [3] methodology and comparative analysis with TinyLFU on a dataset of 50,000 unique authorization tokens (Zipf distribution for modeling real access patterns, α=1), with a cache size of 5,000 elements (10% of the dataset) and simulation over 1 hour for 5 tenants with varying activity, the algorithm using the improved mathematical model achieved a hit rate of 92% ± 2%, which is 7% higher than known models LRU (85% ± 2%), LFU (82% ± 3%), and ARC (88% ± 2%), Table 3.

Table 3

Comparison of the improved mathematical model with known models

Mathematical Model	Time Complexity	Space Complexity	Tenant-awareness	Hit Rate (theoret.)
LRU	O(1)	O(k)	No	~85%
LFU	O(log k)	O(k)	No	~82%
ARC	O(1)	O(2k)	No	~88%
Improved	O(1)	O(k + t)	Yes	~92%

Thus, a promising direction for development is the application of machine learning approaches [26] for automatic tuning of coefficients β, γ, δ based on sidecar component load metrics in real-time, which can further increase the efficiency of the improved mathematical model.

The evaluation of the efficiency of the improved mathematical model of adaptive data caching for the sidecar component was conducted under multi-tenant load conditions using formula (10). The coefficient $\delta \cdot TenantActivity_j$ accounts for tenant activity in multi-tenant cloud environments for the first time, ensuring prioritization of data from active tenants and significantly improving overall data caching efficiency (Fig. 10). The improved mathematical model of adaptive data caching has higher efficiency indicators across the entire load range from 50 req/sec to 1000 req/sec and has the least speed degradation (9.6%) compared to known mathematical model algorithms LRU (23.5%) and LFU (19.3%).

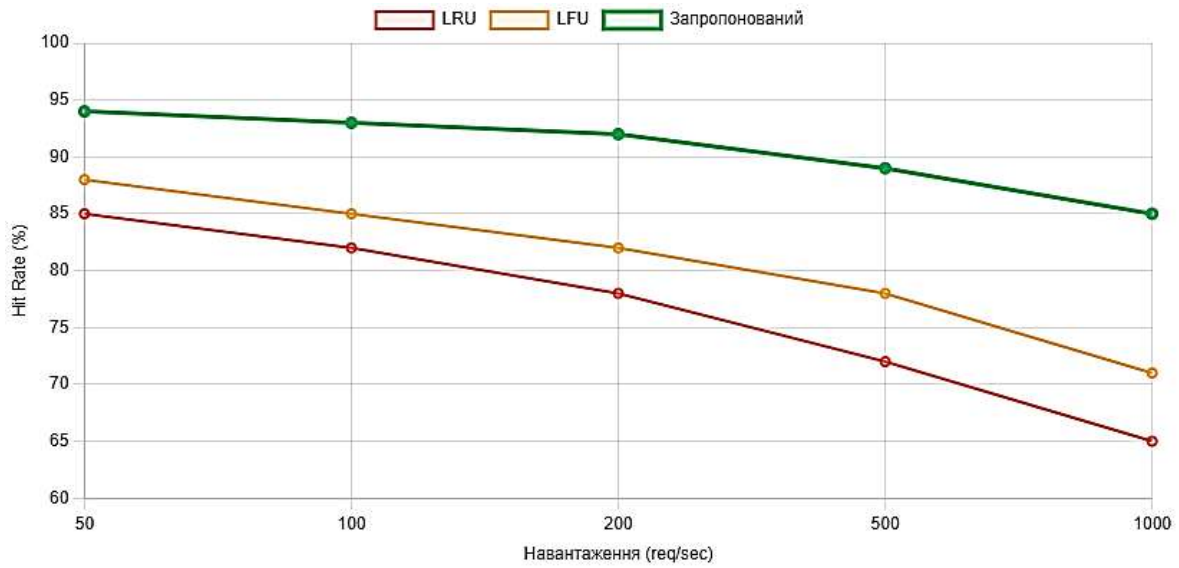


Figure 10. Comparison of cache hit rates of algorithms

The caching algorithm using the improved mathematical model ensures a hit rate of 92% and increases the speed of the authorization filter in multi-tenant DMS. It has been experimentally confirmed that considering tenant activity increases the hit rate by 8-12% compared to known algorithms of mathematical models LRU and LFU, which is important for multi-tenant cloud architectures (Fig. 11).

Thus, the improved mathematical model of adaptive data caching for the sidecar component, considering tenant activity, ensures both a reduction in load on the Keycloak authorization server and an increase in the cache hit rate to 92%, as well as an increase in DMS operational speed.

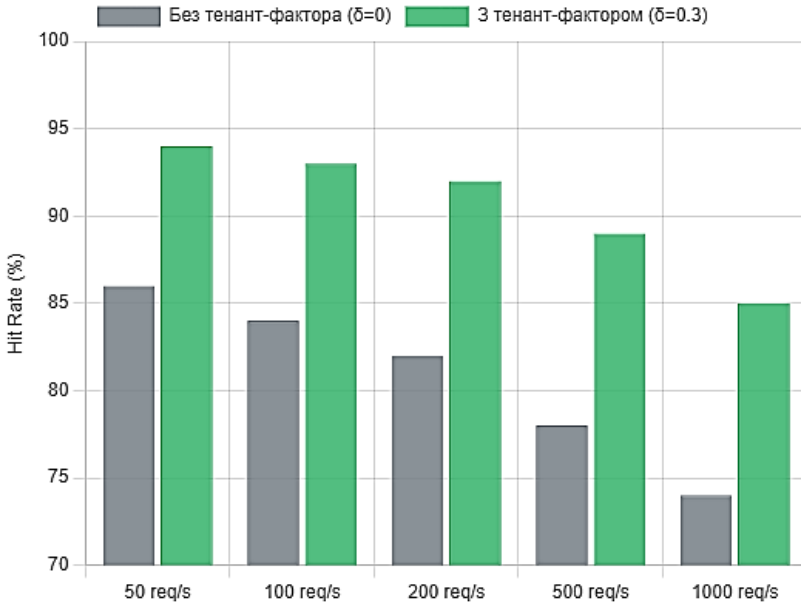


Fig. 11. Influence of the tenant factor

Development of a mathematical model of an adaptive routing graph for request processing by the sidecar component

Routing in the sidecar component is one of the main aspects of its functionality [11, 15, 32], ensuring efficient and managed interaction between microservices. Modern DMS are characterized by topology dynamism, where microservices can change the number of instances, move, or become temporarily unavailable, complicating the decision-making process regarding optimal request routing. Accurate and dynamic routing in sidecar components is of significant importance for the development of modern DMS, as it resolves the issue of determining the microservice to which a request is addressed for processing [31]. Both the speed and resilience of the DMS to loads depend on the correctness of routing. Therefore, developing an efficient mathematical model of routing is a relevant scientific task.

Fig. 12 shows the general scheme of incoming request processing in the sidecar component, which includes the stage of determining the outgoing request route.

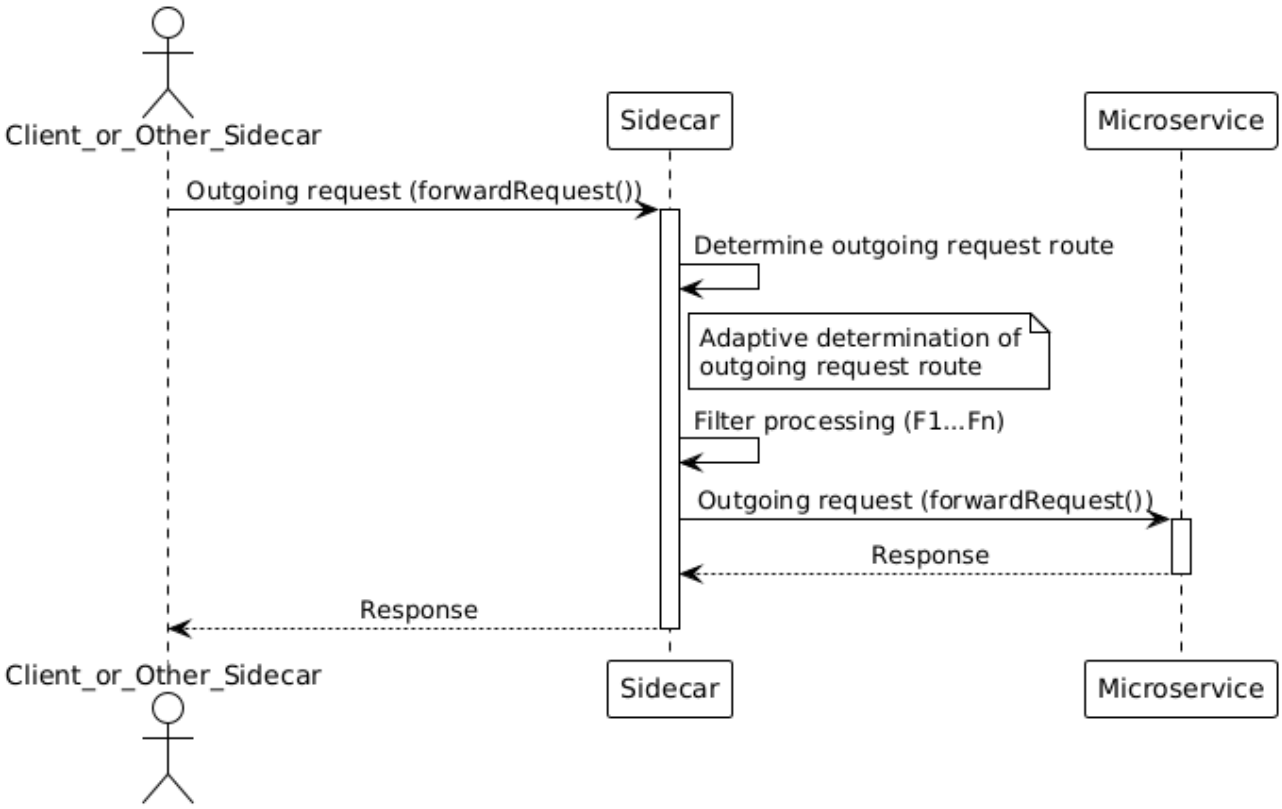


Figure 12. Scheme of request processing by the sidecar component with the stage of determining the outgoing request route

Modern solutions for managing microservice interaction, such as Istio and Envoy [15, 11], implement approaches to request processing in DMS sidecar architectures based on static rules, routing tables, and simple load balancing policies. However, their ability to adapt to dynamic changes in load and DMS topology is limited. These limitations are related to the lack of dynamic consideration of key sidecar component operation metrics, multi-tenancy, and automated decision-making regarding optimal route selection for various DMS scales. Therefore, a need arises to create an adaptive routing system with decision-making for sidecar components, ensuring efficient route selection under conditions of large-scale data traffic and variable DMS characteristics.

Application of the pattern-based incoming request routing approach

The pattern-based routing approach involves checking the correspondence of incoming requests against predefined route patterns. Optimization is achieved through the use of a pre-loaded pattern container, which accelerates the comparison process and ensures efficient processing of a large number of routes without significant overhead. This approach allows for supporting large DMS with hundreds (thousands) of routes, and also guarantees stable operation in complex multi-tenant DMS. The sequence diagram of the pattern-based routing approach is shown in Fig. 13.

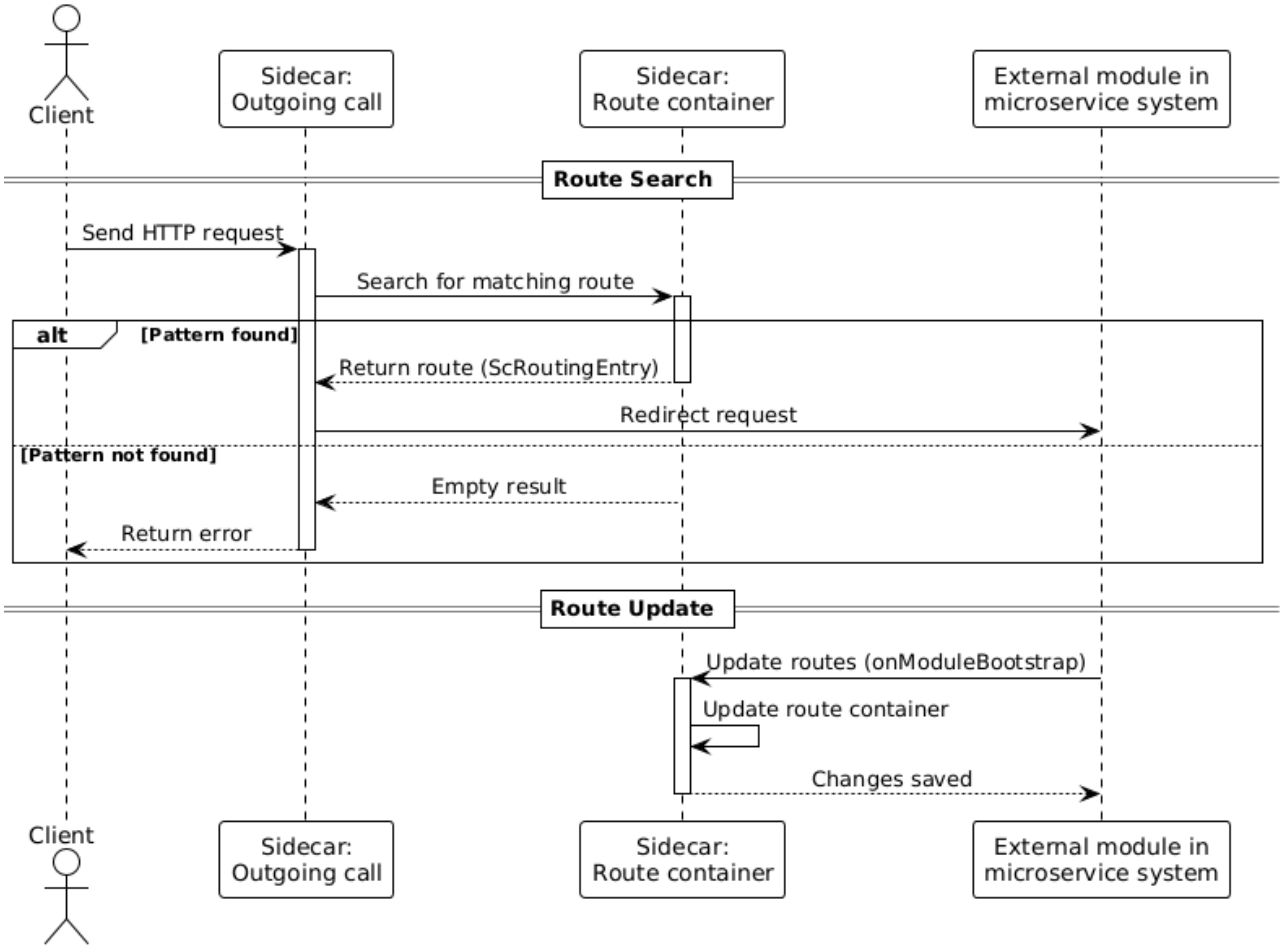


Figure 13. Sequence diagram of the pattern-based routing approach

One part of the diagram is the route update logic. If the DMS configuration changes, for example, when adding a new microservice, the sidecar component receives a corresponding event via a distributed messaging platform such as Kafka.

Upon receiving the event, the sidecar component updates the routing table, adding new endpoints implemented by the added microservice.

Thus, routing functions according to the latest version of microservice configurations and ensures access to all current DMS endpoints. Fig. 14 shows the sequence diagram of the sidecar component routing configuration update process.

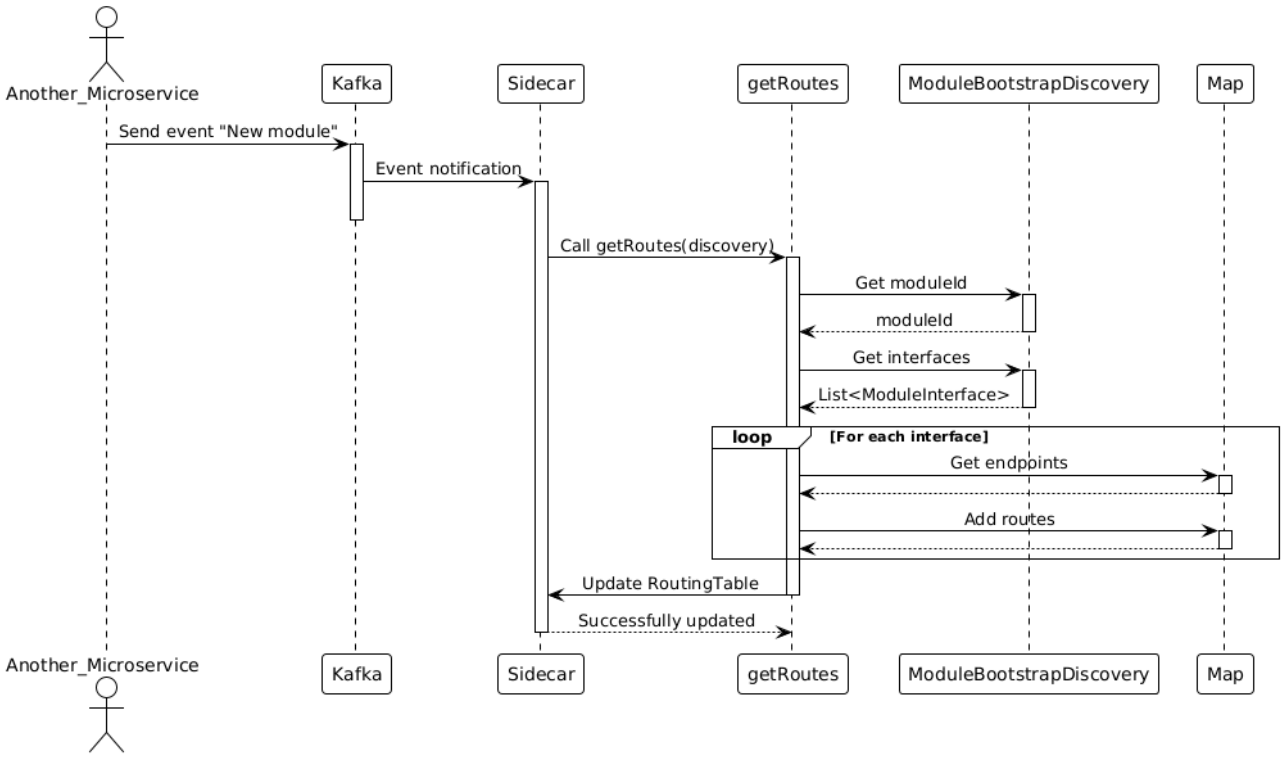


Figure 14. Sequence diagram of the sidecar component routing configuration update process

The logic of the getRoutes() function for populating the container of all routes provided by the microservice can be written as follows:

```
static Map<String, List<ScRoutingEntry>>
collectRouteContainer(ModuleBootstrapDiscovery moduleDiscovery) {
    var moduleName = moduleDiscovery.getModuleId();
    var routeContainer = new HashMap<String, List<ScRoutingEntry>>();
    log.debug("Collecting routes for module: {}", moduleName);
    for (var serviceInterface : moduleDiscovery.getInterfaces()) {
```

```
            var interfaceName = serviceInterface.getId();
            var interfaceType = serviceInterface.getInterfaceType();
            for (var endpoint : serviceInterface.getEndpoints()) {
                var patternPrefix = getPatternPrefix(endpoint);
                var routingList = routeContainer.computeIfAbsent(patternPrefix,
                k -> new ArrayList<>());
                var routingEntry = ScRoutingEntry.of(
                    moduleName, moduleDiscovery.getLocation(),
                    interfaceName, interfaceType, endpoint);
                routingList.add(routingEntry);
                log.debug("Routing entry added: prefix [{}], entry [{}]",
patternPrefix, routingEntry);
            }
        }
        return routeContainer;
}
```

Using the distributed messaging platform Kafka [1] for event-driven route updates ensures high availability and eventual consistency [4] in the DMS. If a different data structure, specifically a graph, is used, it must be updated, including adding new nodes (endpoints) received via messages transmitted through the Kafka messaging platform.

In the sidecar component, route search is performed by checking the routing table with given patterns [8]. To check the correspondence of the request URI to the pattern, it was developed the fastMatch algorithm, which allows for processing route patterns with both fixed and dynamic segments, ensuring a balance between flexibility and route search speed.

The main stages of the algorithm include:

– iterative comparison of pattern characters and request URI;

– processing of dynamic pattern segments (e.g., Tenant ID);

– flexible skipping of part of the request path if necessary.

Thanks to the developed fastMatch algorithm, the sidecar component processes complex incoming request URI patterns with high speed, ensuring efficient match

search, even in large DMS.

The route container can be represented as a set R:

$$R = \{r_1, r_2, \ldots r_n\}. \quad (11)$$

Each route r_i is described as a tuple:

$$r_i = (Pi_{,}, M_i, S_i), \quad (12)$$

where P_i is the route pattern, which may include dynamic segments (e.g., /api/{tenantId}/users); M_i is the set of allowed HTTP methods for r_i; S_i is the microservice to which the request will be directed.

The route container is the following mapping:

$$C: P \to 2^R, \quad (13)$$

where P is the set of all patterns; R_p is the subset of routes corresponding to each pattern.

To evaluate the efficiency of the developed algorithm, the parameter T_{lookup} – route search time – is used. Routes are organized in a container where for each incoming request URI, matching with the route pattern is performed. Then, the route search time can be estimated as follows:

$$T_{lookup} = O(n * m), \quad (14)$$

where n is the number of routes in the container; m is the average length of the route pattern.

This complexity is due to the necessity of checking the correspondence of the request URI with each route pattern, which also includes dynamic segments (e.g., tenantId).

To estimate the time for loading all routes into the route container, the parameter T_{init} is used, which depends on the number of microservices in the DMS $|M|$ and the number of routes in the module $|R_k|$:

$$T_{init} = O(|M| \times max(|R_k|)). \quad (15)$$

The total number of routes in the DMS can be estimated as follows:

$$|R| = \sum_{k=1}^{|M|} |R_k|, \quad (16)$$

where $/M/$ is the number of microservices in the DMS; $/R_k/$ is the number of routes in the k-th microservice.

The pattern-based approach ensures simplicity of implementation and sufficient efficiency for DMS with a small number of routes, and is flexible enough to support DMS with a variable number of microservice instances within a defined time interval. However, with an increase in the number of routes, limitations begin to appear:

– linear growth of route search time with an increase in the number of microservices in the DMS – as the number of routes increases, the number of correspondence checks grows proportionally to $O(n \times m)$, which negatively affects the sidecar component response time;

– lack of adaptability to the current DMS state – routes are loaded during sidecar component startup with fixed priorities, and correspondence search is based exclusively on URIs and HTTP request methods without considering dynamic parameters of corresponding microservices. This means there is no automatic consideration of changes in response time, load, or microservice availability, limiting real-time routing optimization possibilities.

Development of a mathematical model of an adaptive routing graph for request processing by the sidecar component

To increase routing adaptability to the current DMS state, a mathematical model of an adaptive routing graph [17, 34] has been developed, replacing the pattern

container with a route graph where nodes represent patterns and services, and edges have dynamic weights considering the current DMS state. The mathematical model serves as the basis for using known graph route search algorithms, ensuring high search speed as well as adaptability to changes in load, response time, and microservice multi-tenancy requirements.

Graph G consists of: a set of vertices V, edges E, and a dynamic weight function W:

$$G = (V, E, W), \tag{17}$$

where W is the dynamic weight function, which maps each edge to a positive number calculated based on current microservice operation metrics collected by the sidecar component (response time, load, and tenant segregation).

A feature of the mathematical model is the dynamism of function W: graph edge weights are updated every 30 seconds or event-driven upon DMS state change, allowing the graph to automatically adapt to current DMS operating conditions.

The graph representation is based on graph theory [9] with adaptation for dynamic microservice topologies. Graph vertices can be represented as follows:

$$V = \{P_1, P_2, \dots P_n,\ S_1, S_2, \dots S_m\}, \tag{18}$$

where P_n are route patterns (e.g., URI /api/service1 or /api/service2); S_m are microservices (e.g., Service1 or Service2).

Each vertex represents an entry point (route) or exit point (specific microservice) in the DMS. The set of all graph edges can be written as follows:

$$E = \{e(u, v)\} \mid u, v \in V. \tag{19}$$

A specific graph edge $e(u, v) \in E$ describes the possibility of transition from node u to node v. For example, the edge /api/service1 → Service1 indicates a link

from route /api/service1 to the specific microservice Service1.

The dynamic weight function for calculating the weight of each edge $e(u,v) \in E$ of the graph is defined as:

$$w(u,v) = \alpha * f_{latency} + \beta * g_{load} + \gamma * h_{tenant_isolation}, \quad (20)$$

where:

$$f_{latency} = \frac{latency(u,v)}{max_{latency}}$$

– is the normalized interaction time between nodes u and v, $f_{latency} \in [0,1]$;

$$g_{load} = \frac{load(v)}{max_{load}}$$

– is the normalized current load on node v, $g_{load} \in [0,1]$;

$$h_{tenant_isolation} = 1 - \frac{requests_from_current_tenant(v)}{total_requests(v)},$$

where *requests_from_current_tenant(v)* is the number of active requests to microservice v from the tenant making the current request; *total_requests(v)* is the total number of active requests to microservice v.

If the microservice is already serving a significant number of requests from the current tenant, the $h_{tenant_isolation}$ value approaches 0, corresponding to a low edge weight and, accordingly, an increased priority for selecting this microservice for the current tenant. Conversely, if the microservice serves an insignificant number of requests from the current tenant (and many from others instead), the $h_{tenant_isolation}$ value approaches 1, corresponding to a high edge weight and contributing to avoiding the selection of this microservice to preserve tenant segregation.

Based on the results of experimental studies, the coefficient values in (20) for calculating edge weight were determined:

$$\alpha = 0,5, \quad \beta = 0,3, \quad \gamma = 0,2.$$

The choice of coefficients was made using the grid search method, which involves iterating through all possible parameter combinations in a given range to find the optimal balance between microservice response time, total load on it, and load distribution from individual tenants. Weight normalization $(\alpha + \beta + \gamma = 1)$ ensures proportional consideration of all factors when calculating edge weight.

The application of formula (20) is considered for a specific DMS operating state at time *t*. Metrics of response time and load for each microservice are collected by the sidecar component, which is deployed together with the corresponding microservice within a single pod.

Table 4 presents the metric values of six microservices (Service A-F) and the calculated graph edge weights based on the current DMS state at time t.

Edge weights are calculated with coefficients $\alpha=0,5, \beta=0,3, \gamma=0,2$, reflecting the priority of response time over load and tenant segregation in the weight function.

Normalization of response time metrics between route and microservice, as well as current microservice load, is performed relative to maximum values:

$$max_latency = 40 \text{ ms}, \quad max_load = 90\%.$$

Table 4.

Microservice metrics and calculated routing graph edge weights for a request from Tenant A at time *t*

Service	Response Time (ms)	Load (%)	Requests from Tenant A (% of total)	w(u,v)
Service A	10	30	80	0,24
Service B	25	60	50	0,57
Service C	15	45	90	0,3

Continuation of table 4

Service D	35	80	30	0,79
Service E	8	20	70	0,226
Service F	40	90	20	0,9

As shown in Table 4, for requests from Tenant A, the optimal choice is Service E, which has the lowest weight ($w = 0.226$).

The lowest weight for Service E is due to the minimum microservice response time of 8 ms ($f_{latency}$=0.2), low total load of 20% ($g_{load} = 0.22$), and a high share of requests from Tenant A to this microservice – 70% ($h_{tenant_isolation} = 0.3$).

The combination of these parameters using formula (20) with coefficients: $\alpha = 0.5$, $\beta = 0.3$, $\gamma = 0.2$ gives the following result:

$$w = 0,5\times0,2 + 0,3\times0,22 + 0,2\times0,3 \approx 0,226,$$

making Service E the priority choice for routing requests from Tenant A.

Service F, with weight w=0,9, is characterized by a critically high response time of 40 ms, maximum load of 90%, and a low share of requests from Tenant A (only 20%, i.e., htenant_isolation = 0,8). Accordingly, Dijkstra's path search algorithm automatically avoids this route to Service F for Tenant A.

The mechanism of the influence of the htenant_isolation parameter on the routing graph edge weight is demonstrated in the example shown in Fig. 15.

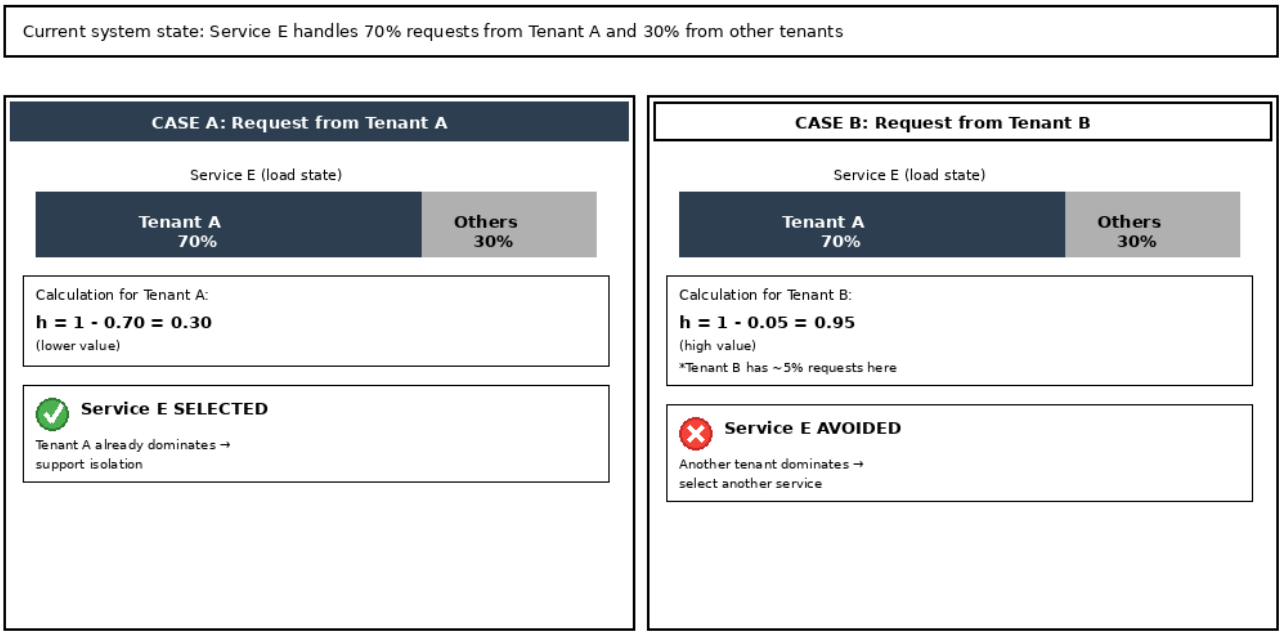


Fig. 15 Influence of the $h_{tenant_isolation}$ parameter for tenant segregation on routing edge weight

As shown in Figure 15, the main feature of the $h_{tenant_isolation}$ parameter is ensuring natural tenant segregation between microservices. If Service E is already processing 70% of requests from Tenant A, then for a new request from Tenant A, the value htenant_isolation = 1 – 0.7 = 0,3 (low) decreases the edge weight and makes the route prioritized, supporting tenant isolation. Conversely, for a request from Tenant B to the same Service E, the $h_{tenant_isolation}$ value would be high (if Tenant B has only 5% of requests there, then $h_{tenant_isolation}$ = 0.95), which increases the routing edge weight and requires the algorithm to choose another microservice. This approach minimizes the mutual influence of tenants on the distribution of computing power in the DMS, as they do not compete for the use of the same microservices.

The developed mathematical model of the adaptive routing graph $G=(V,E,W)$ accounts for all described parameters (microservice response time, load on it, and tenant segregation).

Fig. 16 presents the developed routing graph for six microservices (Service A-F) at a specific time t, where edge weights are calculated using formula (20) with coefficients α=0,5, β=0,3, γ=0,2 based on current DMS operation metrics from Table 4.

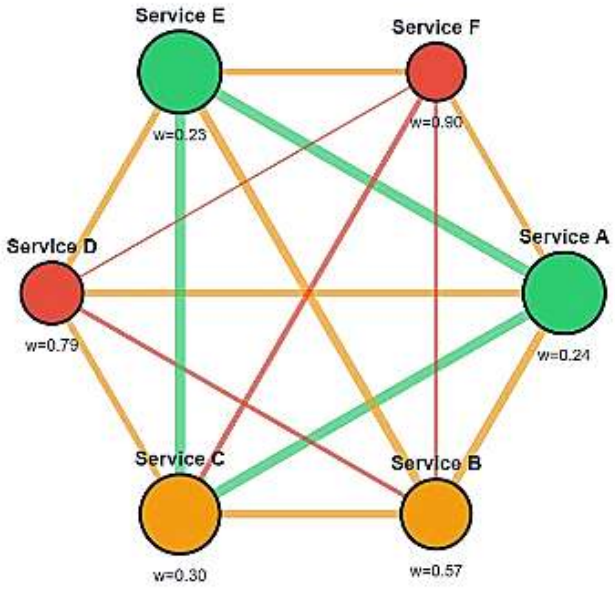


Figure 16. Routing graph $G=(V,E,W)$ at time t

The color coding of graph edges reflects route quality for Tenant A: green (w<0,3) – optimal route, yellow (0,3≤w<0,6) – average quality, red (w≥0,6) – avoid due to high response time or load. Lower edge weight corresponds to a better route.

An incoming request for routing can be defined as follows:

$$Req = (U, M), \tag{21}$$

where U is the request URI; M is the HTTP method of the request (e.g., GET).

To find the route node P_i corresponding to U, the function Match(U, P_i) returns True if the URI matches the pattern P_i.

For example, a request Req = (/api/users, GET) arrives. The request URI "/api/users" matches the pattern P_1 = /api/users via the Match(U, P_1) function. To execute the request, it is necessary to determine the optimal microservice among those available.

If pattern P_1 = /api/users can be processed by two microservices – Service E (w=0,23) or Service F (w=0,9) – Dijkstra's algorithm will choose the path via Service E due to the significantly lower edge weight. Thus, the request will be directed to Service E as the optimal route in the current DMS state.

Graph construction is performed based on the static structure of routes. All nodes P_i and services S_i are added during the sidecar component startup. Graph edges, defining connections between nodes, are formed based on routes registered in the DMS.

The search for the optimal route in the graph is implemented using the well-known Dijkstra algorithm [9, 32], which ensures the selection of the route with the minimum total edge weight.

The buildGraph() function code can be written as follows:

```
public static Graph buildGraph(List<ModuleBootstrapDiscovery> modules) {
   Graph graph = new Graph();
   for (var module : modules) {
     var routes = getRoutes(module);
     for (var route : routes.entrySet()) {
        var path = route.getKey();
        var entries = route.getValue();
        for (var entry : entries) {
          // Add graph edge with a default weight 1.0
          graph.addEdge(path, entry.getRoutingEntry().getPathPattern(), 1.0);
          // Weight will be calculated dynamically later
        }
      }
   }
   return graph;
 }
```

The scheme of the route search algorithm is shown in Fig. 17.

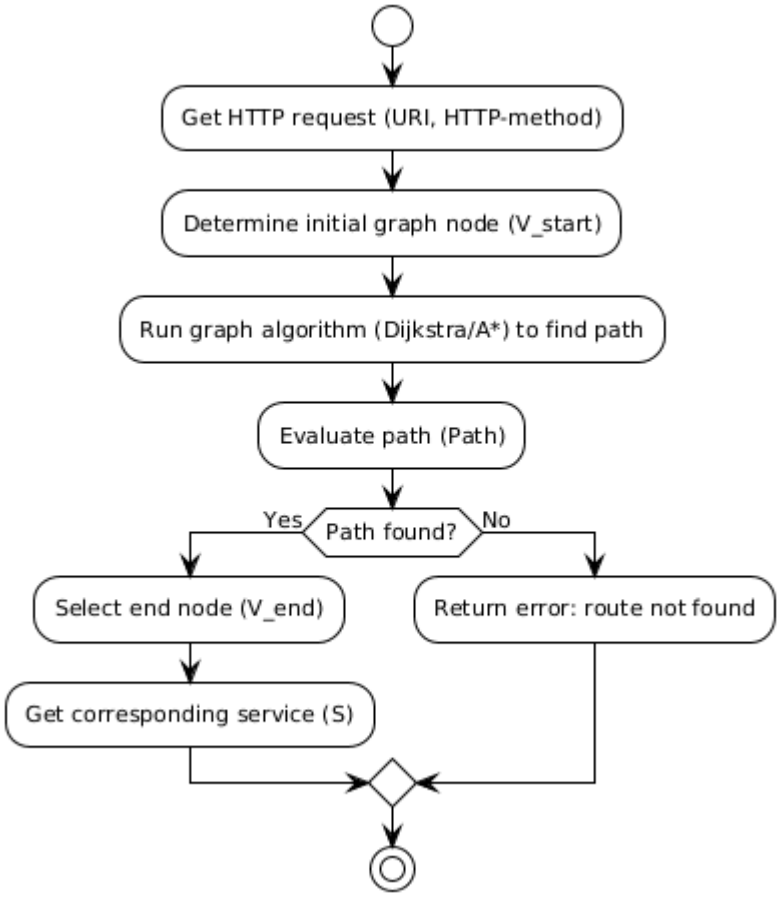


Figure 17. Scheme of the route search algorithm in the graph

Dijkstra's algorithm is used to find the shortest path in graphs with non-negative edge weights, analyzing all possible paths and gradually narrowing the search space, which is effective for service networks. For each node, the path cost to it from the start node is determined, calculated using the function:

$$cost(n) = g_n, \tag{22}$$

where *cost(n)* is the total cost of the path to node *n*, *g(n)* is the accumulated sum of edge weights on the path from the start node to *n*.

The algorithm executes in real-time, analyzing the current connection scheme between routes and microservices considering dynamic edge weights calculated using formula (20).

Thus, an adaptive mathematical model of the routing graph has been developed,

which accounts for the current DMS state and allows for increasing DMS throughput via dynamic request distribution among less loaded microservices, and also ensures the processing of requests from a single tenant on the same microservices to support data affinity. As a result, all DMS components are used maximally effectively, ensuring DMS operational stability against increasing request volumes. The known graph search algorithm (Dijkstra) is applied to the developed adaptive mathematical model, allowing for the consideration of dynamic DMS operation characteristics when determining the optimal request processing route.

Figure 18 presents the routing algorithm constructed based on the developed adaptive mathematical model of the routing graph.

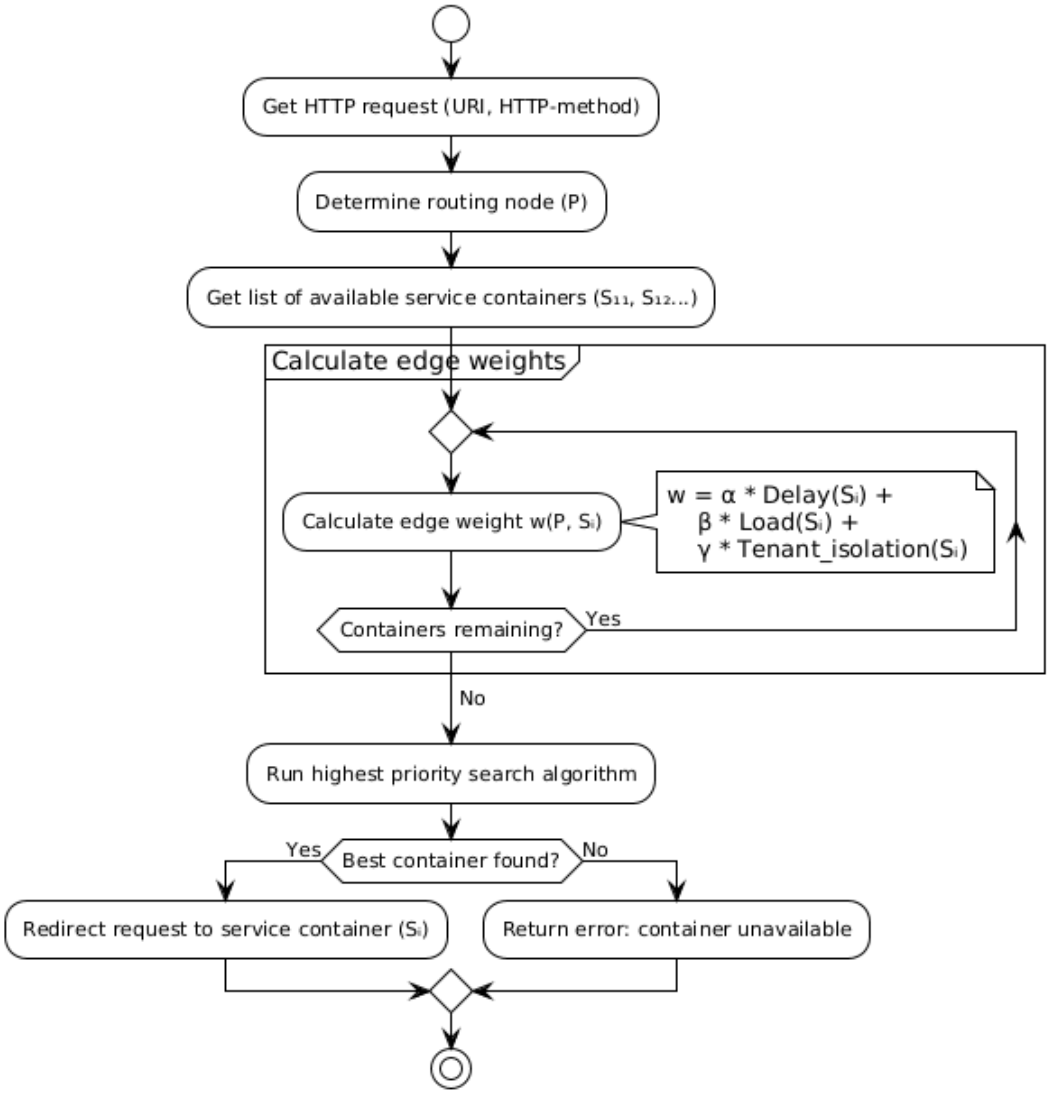


Figure 18. Routing algorithm based on the adaptive mathematical model of the routing graph

Hybrid approach to routing in the sidecar component

Theoretical analysis of the graph-based routing algorithm shows that it has logarithmic complexity $O((|V|+|E|)log|V|)$, ensuring higher speed with increasing route numbers compared to the pattern-based routing algorithm, which has a complexity of $O(n*m)$.

The pattern-based routing approach remains efficient for DMS operation with a small number of routes, due to implementation simplicity and the absence of overhead for graph construction.

Thus, a hybrid routing mode is proposed, which switches between pattern and graph approaches depending on the number of routes in the DMS. The operating mode and switching threshold are configured in the sidecar component configuration file:

```
routing:
   mode: hybrid  # hybrid | pattern-only | graph-only
   pattern-to-graph-threshold: 500
   metrics-update-interval:    30s
```

The main advantage of the adaptive routing graph is its adaptability thanks to dynamic edge weights calculated using formula (20), allowing the sidecar component to automatically respond to changes in response time, load, and microservice multi-tenancy requirements. Such flexibility allows for adapting DMS operation both for small architectures where the pattern approach is sufficient, and for high-load cloud platforms with thousands of routes, where the mathematical model of the routing graph ensures high speed and adaptability.

Development of a method for adaptive request processing for the sidecar component in distributed microservice systems

The method for adaptive request processing for the sidecar component in distributed microservice systems is defined as a tuple:

$$Method = ⟨Filtering, Caching, Routing, FlowControl⟩,$$

where each component is a separate mathematical model.

The developed mathematical model for data flow management, FlowControl [22], is the basis of the adaptive request processing method and is presented in detail in [16]. It includes the main formulas for calculating dynamic request processing queue sizes, request processing recovery delays – both adaptive and exponential – and conditions for making decisions on switching between three data flow management modes [33, 40].

Incoming request processing is performed according to the expression:

$$R_{out} = FlowControl(Routing(Caching(Filtering(R_{in}))))$$

using all developed mathematical models.

Figure 19 shows a diagram presenting the processing sequence of such an HTTP request. The request passes through four stages: multi-level filtering composed of five sequential filters F_1-F_5, adaptive authorization token caching within filter F_5, outgoing request graph routing with dynamic adaptive edge weight calculation, and data flow management with three request processing queue management strategies.

The basis of the method is the mathematical model for data flow management [16], which coordinates the operation of all previous mathematical models (Fig. 20). The essence of the adaptive request processing method's operation is presented in the form of schema shown in Fig. 21.

The developed method of adaptive request processing for the sidecar component in DMS is adaptive to expanding the sidecar component's capabilities by adding new filters for request processing, as well as using the developed metric collection mechanism for application in new mathematical models of request processing. The method is capable of adapting to specific business requirements and applying complex adaptive request processing.

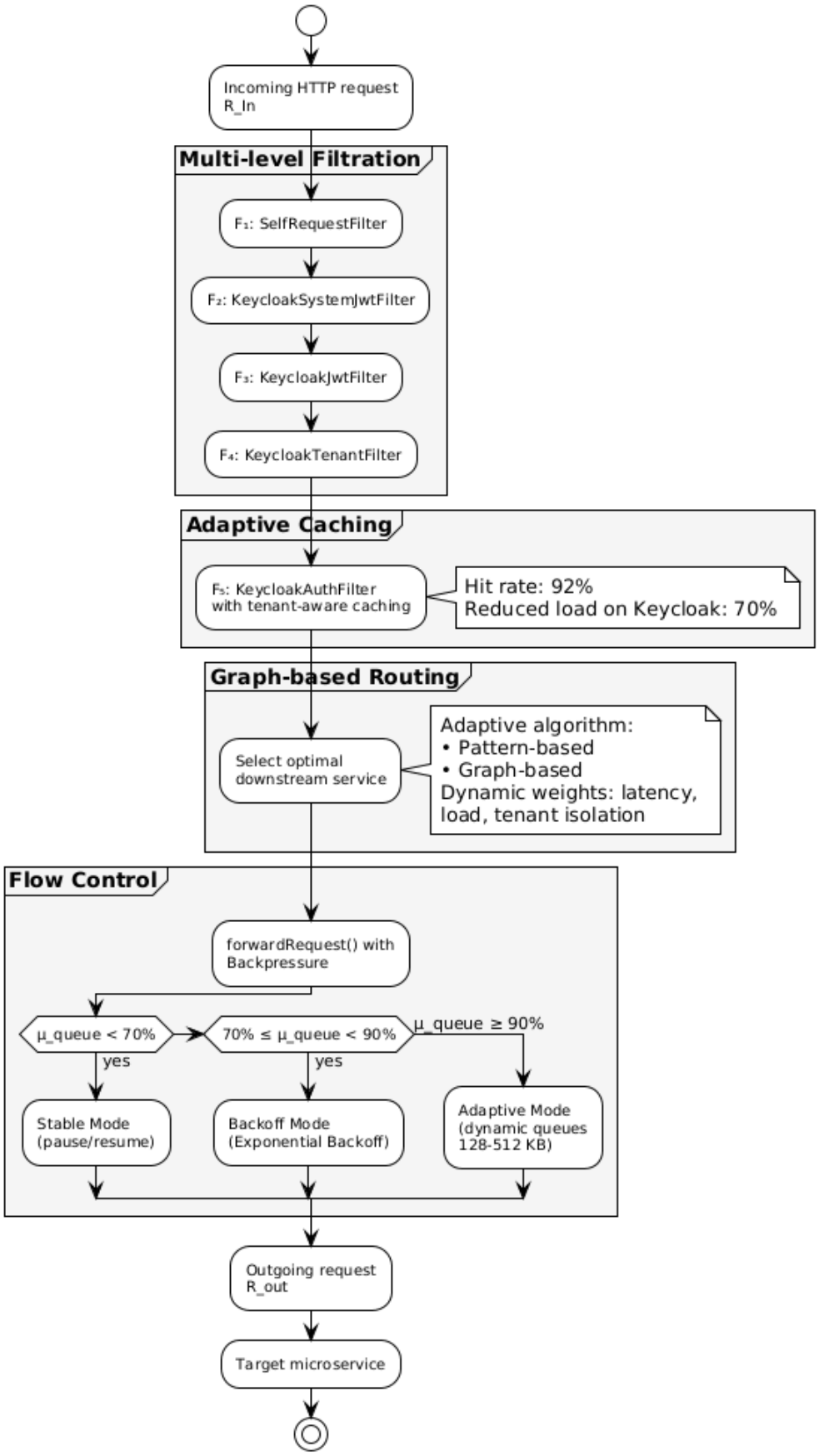


Figure 20. Architecture of the adaptive request processing method

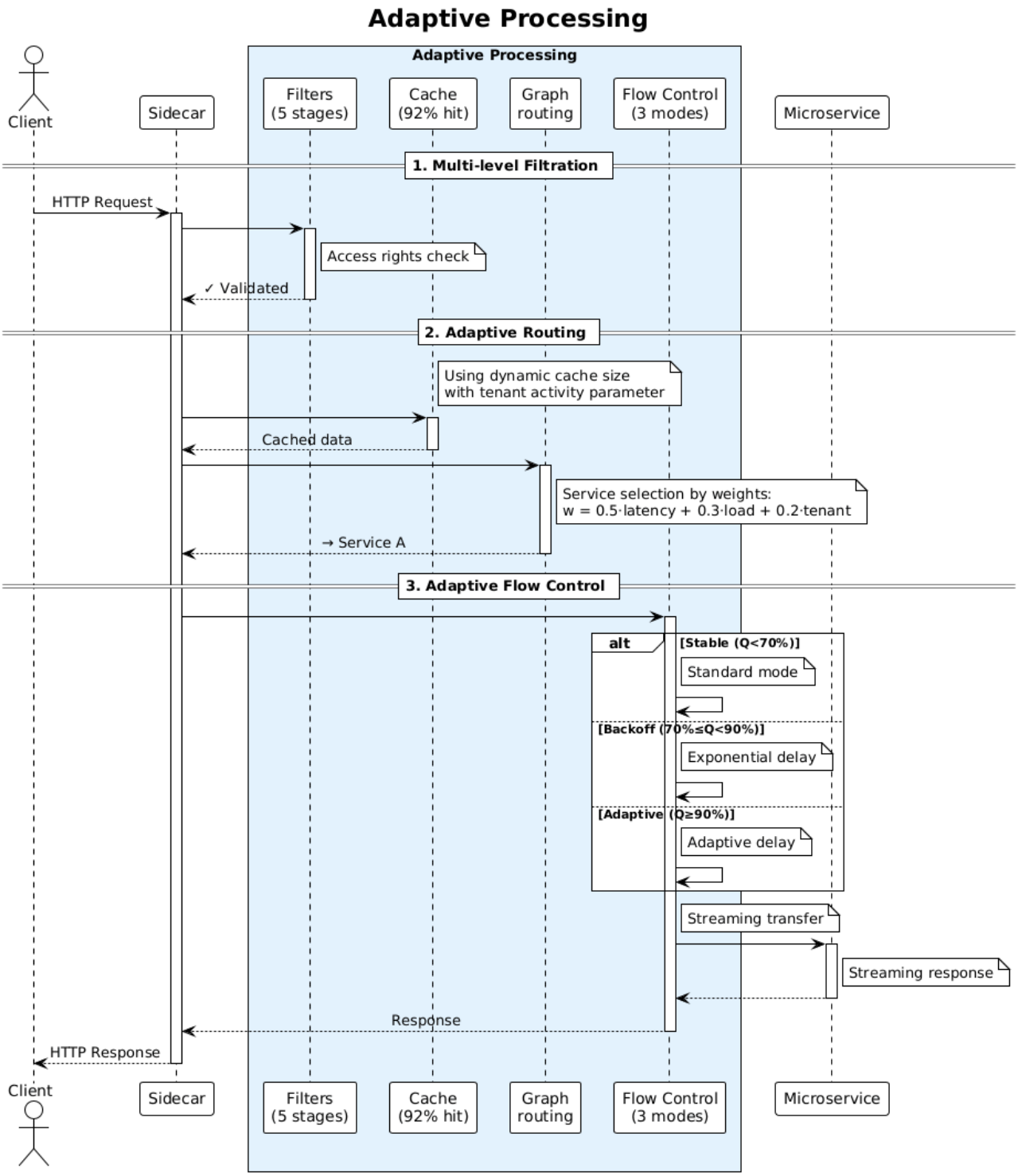


Figure 21. Sequence diagram of the adaptive request processing method

The architectural flexibility of the developed sidecar component is based on a modular approach that extends sidecar component functionality without changing its structure. This approach is achieved through the use of the "Strategy" architectural

pattern for routing algorithms, the "Chain of Responsibility" pattern for filter processing, and the adaptive mathematical model for data flow management, which can dynamically adjust to various load scenarios (Table 5). Thanks to the developed method, a sidecar component has been developed that possesses a high level of flexibility and request processing adaptability in high-load DMS (Fig. 21, 22).

Comparative analysis of the developed sidecar component with existing ones shows that the developed sidecar component has higher indicators for the considered characteristics [7, 31].

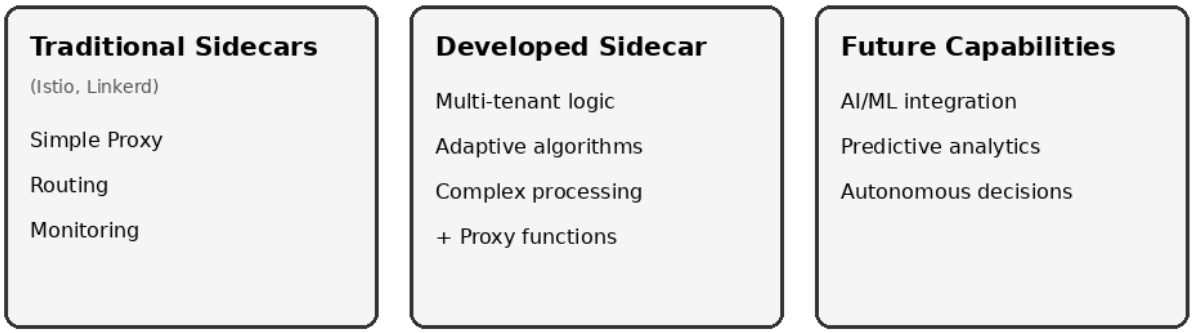


Figure 21. Comparison of known sidecar components and the developed one

Table 5.

Advantages and variability of the developed sidecar component

Characteristic	Istio Envoy	Linkerd2-proxy	Developed Sidecar Component
Main Function	Network Proxy	Network Proxy	Proxy + Adaptive Processing
Customizability	Configuration via YAML	Limited Configuration	Programmatic Extensibility
System Rule Integration	Absent	Absent	Native Support
Adaptability	Static Rules	Static Rules	Dynamic Adaptation
AI/ML Readiness	Requires External Services	Requires External Services	Architecturally Ready

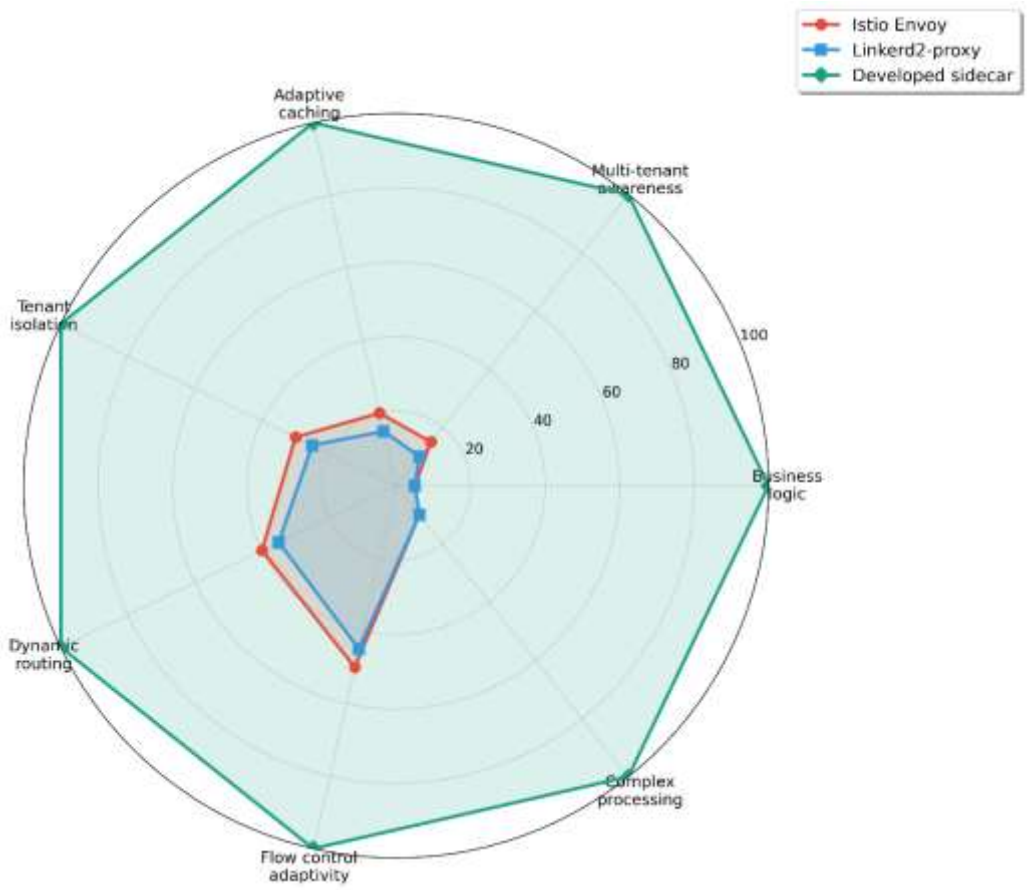


Figure 22. Comparison of functionality extension capabilities of the developed sidecar component with existing ones

Conclusions. Thus, a method for adaptive request processing for the sidecar component of DMS has been developed, which applies improved and developed mathematical models of data caching and request routing, combined into a request processing filter structure.

The application of the method is based on the proposed architectural solution, which involves deploying the sidecar component directly in a pod with the main microservice using a shared network namespace. Based on the conducted comparative analysis, the feasibility of using the Quarkus framework for building the sidecar component is substantiated. It is the combination of Quarkus capabilities with a reactive programming approach and native Kubernetes support that allows for achieving minimal startup time and resource consumption indicators in a dynamic environment.

The mathematical model of adaptive data caching for the sidecar component has been improved, distinguishing it from existing ones by considering the tenant activity parameter. The mathematical model ensures dynamic cache size management depending on the sidecar component load, allowing for optimized RAM volume consumption during low DMS load and increased request processing throughput during high DMS load. Application of the improved mathematical model allows for increasing the hit rate to 92% compared to 78% in LRU and 82% in LFU algorithms in multi-tenant DMS environments.

A mathematical model of an adaptive routing graph for the sidecar component with dynamic edge weights has been developed; these weights are automatically updated based on microservice operation metrics: latency time and load, for even load distribution in the DMS. The mathematical model also includes a parameter ensuring localization of request processing for a single tenant on the same microservice, contributing to data processing segregation between tenants in a multi-tenant DMS environment.

The adaptability of the developed method allows the sidecar component to respond to changes in DMS load and microservice state, ensuring stable and efficient request processing under unstable or peak operating conditions. Thanks to the ability to adjust to current conditions, the sidecar component maintains high speed and reliability of request processing.

As future research, it is of interest to develop mechanisms for autonomous tuning of sidecar component mathematical model parameters based on machine learning [26, 36]. Since the sidecar component processes all DMS requests, this will allow for using accumulated data to predict optimal parameter values in real-time.

DOI: 10.46299/979-8-90383-421-1.2

2. Intelligent optimization of high-concurrency distributed systems

Introduction

As live-stream e-commerce and social group buying emerge as new consumption models, e-commerce promotion platforms are experiencing a dual surge in traffic volume and business complexity. According to data from iResearch, during major promotions, the average daily PV (page views) of leading e-commerce platforms exceeds 3 billion, with peak TPS (transaction processing rate) reaching up to 120,000 transactions per second, placing stringent demands on system capacity. Traditional monolithic architectures, under the impact of traffic surges, have exposed multi-dimensional performance bottlenecks: when traffic spikes, service circuit breakers frequently trigger, causing core business response delays exceeding 500ms; at the database level, due to row lock contention, the order creation interface throughput drops by 40%; cache penetration issues lead to a sharp increase in database load, triggering a snowball effect. Actual test data shows that during a platform's 618 event, user access timeouts rose from a normal 1.2% to 18.7% due to architectural flaws, directly resulting in a GMV loss of over 230 million yuan.

Under the constraints of CAP theorem in distributed systems, how to balance system availability and data consistency, achieve elastic scalability of architecture, and dynamic optimization of performance has become a core issue in e-commerce technology. By building microservices architecture, designing multi-level caching strategies, and applying traffic peaking techniques, not only can the average user response time be reduced to within 200ms, but also system resource utilization can increase by over 35%. Such technological innovations can significantly enhance the user shopping experience and boost page conversion rates by 28%, directly translating into higher traffic monetization efficiency and commercial value for enterprises, which is of great practical significance for promoting high-quality development of the digital economy.

The main goal is to develop and optimize the architecture of a high-load e-

commerce promotion platform by implementing a microservices model, elastic scaling, and intelligent resource planning to ensure high performance, scalability, and stability under peak loads.

Analysis of the current state of the subject area

During a flash sale for a certain brand of mobile phone, due to the exhaustion of the database connection pool, 100,000 user requests were blocked, causing order processing delays exceeding 15 minutes.

In this context, high-concurrency e-commerce promotion platforms have emerged [46]. These platforms support e-commerce companies in conducting promotional activities (such as time-limited discounts, flash sales, live streaming, etc.), and can handle massive user visits, transaction requests, and data interactions through distributed technology systems [47]. Their core features include:

1. Traffic Spikes. During promotional events, traffic grows exponentially, with peak volumes reaching 50 to 100 times the normal level (for example, a platform's "Double 11" request volume at midnight can exceed 8 million requests per second). This pulsatile traffic pattern stems from intensified competition in the e-commerce industry, where merchants attract users through large-scale promotions, causing a sharp increase in user visits over a short period. During the "618" period in 2023, a live-streaming e-commerce platform experienced a server overload rate of 40% within the first hour due to inaccurate traffic forecasting, forcing an emergency expansion of 300 servers [48].

2. Business Real-time Performance. User operations (such as product inquiries, inventory deductions, and order submissions) must be responded to within 200 ms; otherwise, the conversion rate drops by over 30%. Neuroscience research indicates that human tolerance for system response times decreases with increasing digitalization. For every additional 100ms of response delay on mobile shopping apps [49], user bounce rates increase by 18%. A fresh food e-commerce platform once experienced a 30% failure rate for flash sale orders due to delayed inventory verification, leading to a 500% week-over-week increase in user complaints.

3. Data Consistency Requirements [50]. Core transaction operations such as

inventory deductions and amount calculations must meet strong consistency to avoid business risks like over-selling and financial errors. In 2015, a maternal and infant e-commerce company experienced a distributed transaction anomaly that led to the over-selling of 2,000 cases of formula milk. The company ultimately compensated users at three times the original price, resulting in direct economic losses of 800,000 yuan and a 25% decline in brand trust. This incident became a critical event that spurred the industry to prioritize data consistency technology research and development.

The core business scenarios of the platform cover three key levels: traffic entry, business logic and data storage [51]. Each level faces significant technical challenges:

1. Traffic entry layer. As the first interface for users to reach the platform [52], it carries high-traffic scenarios such as homepage recommendation and special pages for limited-time flash sales activities. There are two technical bottlenecks:

– static resource transmission efficiency is low. The loading time of high-definition product images (averaging 2MB per image) and promotional videos (averaging 50MB per video) accounts for over 60% of the total page rendering time [53]. During a major promotion in 2022, a cosmetics e-commerce platform failed to enable the WebP image format, causing the homepage loading time on mobile devices to reach 8 seconds, resulting in a 40% decrease in user conversion rate compared to expectations;

– inadequate burst traffic management capabilities. Traditional load balancing [54] algorithms (such as round-robin and random) can easily lead to node overload when traffic is skewed (for example, if a popular product account for 70% of visits). During the 2023 "Double 11" event, a platform failed to implement a load balancing strategy guided by traffic prediction [55], resulting in a homepage response delay exceeding 3 seconds, causing a 15% user churn.

2. Business logic [56] layer: responsible for processing complex promotion rules and inventory management and other highly complex business processes:

– promotion rules are computationally intensive. During the "Double 11" mega sale, real-time calculations for cross-store discounts [57] (such as a 50% off on purchases over 300), member discounts (95% off), category coupons (a 30% off on

purchases over 200), and other 12 types of discount combinations are required. The logic for calculating a single order involves over 200 lines of code. Before optimization, a certain platform maintained a CPU utilization rate above 95%, leading to a 10% error rate in order calculations;

– inventory Lock Competition [58]. In distributed transaction processing using pessimistic locking mechanisms (such as SELECT FOR UPDATE), the lock wait requests for inventory deduction operations during peak periods account for up to 30%. During a digital product pre-sale event in 2024, lock competition led to an order creation success rate drop to 85%, with over 2,000 users experiencing the abnormal situation of "successful order placement but insufficient inventory."

3. Data storage layer: Facing the double test of high concurrent [59] read and write:

– read operation cache bypass [60]. When querying popular items, if the cache is not hit and no null value protection is applied, it will directly bypass to the database. During the pre-sale period of a certain hot-selling phone in 2023, due to the Redis cache failing and the Bloom filter not being enabled, the database's QPS surged from 5000 to 30000, causing the primary database to overload and crash for 10 minutes;

– write operation transaction inconsistency. Under the order sharding architecture, distributed transactions rely on the two-phase commit (2PC) [61] protocol. Network latency leads to a transaction timeout rate of 5%. In 2022, a fresh e-commerce company experienced an issue where inventory deductions and order creation transactions were inconsistent, resulting in over 500 orders with deducted inventory but no generated orders, leading to collective user complaints.

With the deep penetration of artificial intelligence technologies, e-commerce platforms have entered the optimization stage, which is based on algorithmic and intelligent monitoring of the entire communication channel. Starting from 2020, through the integration of big data analysis, machine learning and distributed system optimization technology, the entire process of intelligent modernization has been realized - from traffic management to transaction guarantee [66].

Traffic Forecasting Technology. In the context of major e-commerce

promotions, the sudden and unpredictable nature of traffic poses significant challenges to system stability. The innovative traffic forecasting model based on LSTM neural networks (He et al., 2020) integrates over ten feature dimensions, including historical traffic data, promotional event calendars, regional weather information, and holiday patterns, to build a dynamic prediction system. After applying this model to a leading live-streaming e-commerce platform, the accuracy of peak traffic predictions was successfully increased to 92%. By triggering elastic scaling mechanisms two hours in advance, the server resource allocation became dynamically managed, improving resource scheduling efficiency by 35%. Additionally, precise resource allocation reduced server costs by 25%, effectively balancing business growth with operational expenses.

Distributed Transaction Optimization [67]. In high-concurrency transaction scenarios (such as flash sales and live-streaming sales), traditional transaction processing models often face performance bottlenecks and data consistency issues. Seata (2019), an open-source distributed transaction solution, offers various transaction modes including AT (Automatic Compensation), TCC (Try-Confirm-Cancel), and SAGA. The flexible transaction scheme designed for flash sale scenarios combines asynchronous message queues with compensation mechanisms, increasing the order creation transaction success rate from 85% to 99.5%. Compared to the traditional TCC mode, which requires a large amount of hand-written code for transaction compensation, the Seata framework reduces development costs by 40% through automated mechanisms, significantly improving development efficiency and system stability.

Full-Chain Performance Monitoring: The Prometheus + Grafana combination (Vikunja, 2021) establishes a comprehensive monitoring system covering the client side, server side, and database. The client side collects metrics such as browser loading speed and resource request latency. The server side monitors JVM memory usage and thread pool status. The database side tracks slow query logs, enabling real-time data collection and visual analysis across the entire chain. After implementing this solution on a cross-border e-commerce platform, the time required to identify performance

bottlenecks was significantly reduced from 4 hours to 30 minutes. Combined with automated alerts and self-healing mechanisms, the efficiency of fault recovery increased by 80%, effectively ensuring the continuity of the user shopping experience [68].

There are three unresolved problems in the existing research:

1. Business-Lack of Architecture Mapping Mechanism. In the e-commerce industry, traffic from promotional activities exhibits significant pulse-like characteristics. Taking a typical flash sale as an example, according to industry data, the peak traffic in the last 30 seconds before the event usually accounts for over 50% of the total daily traffic. However, both academia and industry have yet to establish dynamic matching models for such traffic patterns and system resource scheduling strategies. Current elastic scaling strategies mostly rely on static metrics, such as CPU utilization thresholds that trigger scaling mechanisms, which inherently have a lag. This lag results in insufficient resource allocation during peak traffic periods, leading to service degradation; while during off-peak periods, resources are wasted, significantly increasing operational costs. According to internal statistics from a leading e-commerce platform, service degradation caused by untimely resource scheduling results in direct economic losses exceeding tens of millions of yuan annually.

2. Inadequate fault tolerance in extreme scenarios. Current mainstream fault recovery mechanisms, such as circuit breaking and rate limiting, have shown significant shortcomings when dealing with large-scale node failures and other extreme situations. When more than 30% of the system's nodes fail simultaneously, existing automated recovery mechanisms will become ineffective, necessitating manual intervention for restoration operations. For example, during a major "Double 11" promotion on an e-commerce platform in 2023, a sudden power outage in a regional data center led to 40% of server nodes being unavailable. During this period, system recovery took as long as 12 minutes, far exceeding the user-tolerable threshold of 5 seconds. This incident directly resulted in over 100,000 lost orders and a 300% surge in user complaints, severely damaging the platform's brand reputation and user trust.

This demonstrates that current fault tolerance mechanisms cannot meet the stringent requirements for system reliability in high-concurrency scenarios.

3. Lack of Cross-Layer Optimization Collaboration. In terms of technical optimization, the optimization methods at the network layer, application layer, and hardware layer have long been fragmented, lacking systematic integration. For example, commonly used TCP BBR congestion control algorithms at the network layer, cache preheating strategies at the application layer, and GPU acceleration technology at the hardware layer are often implemented independently. A certain e-commerce platform once optimized its database connection pool, improving transaction processing capability (TPS) by 30% through technical improvements. However, due to the lack of synchronized optimization of network bandwidth configuration, the overall throughput only increased by 15%, significantly reducing the effectiveness of the technical optimization. This absence of cross-layer collaboration results in the failure to fully leverage the benefits of each layer's technical optimizations, limiting the potential for overall system performance improvement.

Based on the above analysis of the characteristics of traffic fluctuations, system architecture bottlenecks and performance optimization pain points of e-commerce platforms, combined with the development trend of the industry and the limitations of existing technologies, the following research hypotheses with progressive relationship are put forward.

Assumption 1. To address the issue of insufficient resource allocation and traffic feature matching in e-commerce systems, a dynamic mapping model of "traffic features-architecture parameters" is constructed. This model includes a module for predicting peak traffic times (based on historical data and time series prediction models considering external variables such as holidays and promotional activities), a user geographic distribution heat map analysis (in conjunction with CDN node scheduling strategies), and an assessment of business module resource consumption (based on microservice call chain tracing and resource monitoring data). The model integrates deep reinforcement learning algorithms (such as Deep Q-Networks DQNs) to build an intelligent decision engine. This model can dynamically adjust server resource

allocation strategies (including container scaling, CPU core binding, and memory pre-allocation) based on real-time traffic changes, achieving precise system resource scheduling. It is expected that during typical peak scenarios of major promotions, CPU utilization will increase from the current 70% to over 85%, and memory utilization will rise to over 90%, while simultaneously reducing resource wastage rates and cost expenditures.

Assumption 2. To address the issue of excessive system response latency during sudden traffic surges, an end-to-end optimization solution that integrates machine learning with traditional performance tuning methods is proposed. By constructing a hot item prediction model using the XGBoost algorithm, potential best-sellers can be identified in advance based on user browsing history, collection and purchase behavior, and product popularity index. This approach combines the Nginx dynamic weight load balancing strategy (which dynamically adjusts request distribution weights according to real-time server load and network latency) with the Redis cache preloading mechanism (preloading hot data into the cache layer before major promotions), forming an "prediction-diversion-caching" optimization chain. It is expected that under sudden traffic scenarios, the system's 99-line response latency (P99 Latency) can be reduced from 500ms to below 300 ms, a reduction of over 40%, significantly enhancing user interaction experience.

Assumption 3. To address the ceiling issue in optimizing a single technical layer, design a cross-layer collaborative optimization mechanism. At the network layer, replace traditional TCP with QUIC protocol, using multiplexing, 0-RTT connection establishment, and forward error correction mechanisms to reduce network transmission latency and packet loss rate; at the application layer, transform microservices into stateless services to eliminate session stickiness issues and enhance horizontal scalability; at the data layer, implement a cold-hot data separation strategy (store frequently accessed hot data on SSD or in-memory databases, and archive low-frequency cold data to distributed file systems). Through collaborative optimization across these three layers, break through the bottlenecks of traditional optimization methods, and expect to increase system throughput from 50,000 QPS to over 150,000

QPS, achieving a performance boost of more than 200%, supporting stable operation under high-concurrency business scenarios.

As competition in the e-commerce industry intensifies, major promotional events like "Singles' Day" and "618" occur frequently, making the performance bottlenecks of traditional e-commerce platforms under high-concurrency scenarios increasingly evident. This study aims to break through these bottlenecks by building a technical system with capabilities of "dynamic perception-intelligent decision-making-rapid response," achieving a leap in the performance of e-commerce platforms. The specific objectives are as follows.

Innovative Architecture. In response to the characteristics of sudden traffic surges, short peak durations, and complex request types during promotional activities, a layered decoupling architecture has been designed. At the access layer, invalid requests are filtered through traffic cleaning strategies to reduce backend pressure; at the business layer, microservices orchestration is adopted to achieve flexible deployment and dynamic adjustment of business modules; at the data layer, elastic scaling technology is utilized to automatically increase or decrease resources based on real-time traffic changes. The design goal of this architecture is to support a tenfold surge in traffic and ensure high availability of 99.99% through mechanisms such as multi-data center disaster recovery and fault isolation, guaranteeing stable operation even under extreme traffic impacts.

Algorithm Breakthrough. Conduct in-depth analysis of performance impact factors on e-commerce platforms under high-concurrency scenarios, and develop data-driven performance optimization algorithms. The algorithm will integrate historical transaction data and real-time monitoring data to build a dynamic performance prediction model, enabling early prediction of traffic trends and pre-allocation of resources. By employing intelligent load balancing, cache optimization, and asynchronous processing technologies, the system throughput is increased by over 200%, with critical interface P99 response times controlled within 200ms. Additionally, server resource utilization is improved by more than 40%, breaking through the limitations of traditional single-technology optimizations and achieving

systematic performance enhancement.

System Construction. Establish a full-chain performance evaluation system from three dimensions: definition of performance metrics, design of evaluation methods, and development of optimization tools. In terms of performance metrics, refine core indicators such as throughput, response time, and resource utilization, and establish tiered standards for different business scenarios; regarding evaluation methods, use techniques like stress testing and chaos engineering to simulate real business scenarios, ensuring the reliability of evaluation results; develop performance monitoring and analysis tools to achieve real-time data collection, intelligent analysis, and automatic generation of optimization recommendations. Ultimately, form a replicable and scalable technical solution to promote the transition of the e-commerce industry from an experience-dependent optimization model to a data-driven, scientific, and intelligent optimization model (table 1).

Table 1

Core elements of architecture design

Architecture Layer	Key Components	Design Objectives	Technical Challenges
1	2	3	4
Access Layer	Load Balancing, API Gateway, CDN	Achieve traffic cleaning and request routing, reduce client-side latency	Efficiency of load balancing algorithms under sudden traffic surges (e.g., round-robin strategy causing node overload during traffic skew)
Business Layer	Microservices, Message Queue, Cache	Decouple business logic and improve processing parallelism	Distributed transaction consistency (e.g., distributed lock contention in inventory deduction during flash sales)
Data Layer	Distributed Database, Search Engine	Ensure data storage and query performance, support elastic scaling	Cross-database transaction processing after database sharding (e.g., network latency issues with the two-phase commit protocol)

From the perspective of user experience and system efficiency, three core optimization dimensions are defined:

– response performance. Including request response time (RT, Response Time),

99-line delay (P99 Latency), the target P99 delay under peak traffic is less than 200ms;

– processing performance. With throughput (TPS/QPS) and concurrent user number (Concurrency) as indicators, the system throughput is required to increase by more than 5 times in the promotion scenario than in normal times;

– resource efficiency. Measured by containerization rate, CPU/memory utilization, storage cost, etc., the goal is to increase server resource utilization from 30-40% in traditional architecture to 60-70% in cloud native architecture.

Core indicators of the business layer:

– promotion rule calculation delay. The calculation time of complex discount combination (such as the superposition of 3 types of coupons) should be less than or equal to 50ms (before optimization, it was 200ms on a certain platform, resulting in 10% order calculation error);

– distributed lock competition rate. The proportion of lock waiting requests in inventory deduction operations, which was reduced from 30% to 5% after Redisson optimization;

– message queue throughput. Kafka single Topic processing capacity, which should support 100,000 + messages per second (to respond to asynchronous inventory change notifications in the case of kill);

– service dependency success rate. The proportion of fault-free nodes in the microservice call chain, which was 70% after introducing Hystrix circuit breaker, increased to 95%.

Core indicators of the data layer:

– cache hit rate. Redis the cache hit rate of hot data in the cluster should be greater than or equal to 95%. A case of database QPS surge caused by cache penetration in a mobile phone pre-sale activity proves that every 1% decrease in hit rate increases the database load by 5%;

– search response time of the search engine. The delay of the product search interface P99 is less than 100ms (based on Elasticsearch shard optimization);

– success rate of distributed transactions. The success rate of AT mode under

Seata framework is more than 99.9%, which is 4.9 percentage points higher than that of traditional 2PC protocol;

– database sharding throughput. After the database is divided into tables, the QPS of a single database is greater than 20,000. Read-write separation and horizontal expansion are realized through Sharding-JDBC.

User experience indicators:

– front-end rendering time. The first screen loading time is less than 3s (mobile terminal)/2s (PC terminal), otherwise the user bounce rate increases by 25% (Google Analytics statistics);

– order creation success rate. The success rate of the core transaction link is more than 99%, but before the optimization of a certain platform, the success rate is only 92% due to the delay of inventory verification, resulting in the loss of millions of GMV during the promotion period;

– system availability. SLA requires up to 99.99%, that is, annual downtime of less than 52 minutes, through multi-active architecture and automatic failover.

Architecture design of high-concurrency e-commerce platform

High availability is a core element to ensure business continuity and user experience for e-commerce platforms. In the modern e-commerce environment, any system downtime or delay may directly lead to transaction failures, user loss, and even damage to the brand image. Therefore, high availability is not only a technical requirement but also an important guarantee for business success. E-commerce platforms need to support large-scale concurrent access and deal with uncontrollable factors such as hardware failures and network fluctuations, which requires the system design to have strong fault tolerance and quick recovery mechanisms. For example, during peak traffic periods, the system should be able to dynamically expand resources to meet demand; in case of sudden failures, it needs to quickly switch to backup nodes to ensure that services are not interrupted.

Load balancing and failover are key means to achieve high availability. Load balancing avoids single-point overload by evenly distributing requests to multiple servers, improving overall performance and stability. Common algorithms such as

round-robin, weighted round-robin, and least connection can be selected according to specific scenarios. The introduction of a failover mechanism further enhances system reliability. When the primary server is abnormal, the system can automatically detect and switch to the backup server, which depends on heartbeat monitoring and health check functions. Combining load balancing and failover can effectively reduce the risk of single-point failure and improve resource utilization and response speed.

Data backup and recovery, as well as monitoring and alarm systems, are equally important. As the core asset of e-commerce platforms, data security is of vital importance. Regular off-site backups can prevent data loss and provide a basis for disaster recovery. Real-time monitoring of the entire system operation status is indispensable. With the help of monitoring tools such as Zabbix or Prometheus, 运维 personnel can timely discover potential problems and trigger processing procedures through preset alarm rules, shorten the fault repair time, and reduce the impact on business.

The application of microservice architecture provides deeper support for high availability. By decomposing complex systems into multiple independently deployed service modules, each module can flexibly adjust resource configuration according to needs, improving overall stability and elasticity. The microservice architecture also facilitates the implementation of continuous integration and continuous delivery (CI/CD), making new function launches more efficient and secure. Building a highly available e-commerce platform requires the comprehensive use of various technologies and methods.

In the design of high-concurrency e-commerce platforms, the scalability principle is crucial, which ensures the long-term stable operation of the system and its adaptability to future demand changes. Modular design, as the core embodiment of scalability, greatly improves system maintainability and reduces the complexity caused by function expansion or modification by decomposing the system into functional and independent modules, such as user management, order processing, and payment settlement. This design reduces the development cycle and potential risks and improves system flexibility, making it possible to develop or optimize related modules when

adding promotional activities or adjusting payment interfaces.

The microservice architecture further enhances the scalability of the system. By splitting the system into multiple autonomous service units, each service focuses on specific business logic and is independently deployed, achieving higher decoupling. For example, the order service can be expanded independently of the product service to cope with the surge in orders. The microservice architecture also supports on-demand resource expansion, such as increasing order service instances through elastic scaling technology to meet instantaneous high-concurrency requirements. This flexibility enables the system to quickly respond to business changes and facilitates the use of different technology stacks to improve development efficiency and technological diversity.

The layered architecture and the open-closed principle provide a solid foundation for the long-term development of the system. The layered architecture clearly divides the functional responsibilities of the presentation layer, business logic layer, and data access layer, allowing developers to focus on the development and optimization of specific layers. The open-closed principle encourages the implementation of new functions through extension rather than modification of existing code, maintaining system stability. For example, when introducing a new payment channel, it can be implemented by adding a payment service subclass without changing the original payment logic. The dependency inversion principle enhances the system's scalability by defining the interaction mode between services through abstract interfaces, enabling specific implementations to be flexibly replaced without changing the high-level logic.

Scalable design is of great significance to the long-term development of the system. It reduces maintenance costs and improves system flexibility and market competitiveness. In the e-commerce field, enterprises need to respond to market demands in a timely manner, launch new functions or optimize the user experience. A good scalable design enables these goals to be achieved efficiently, such as quickly deploying promotional activities or supporting new payment methods to attract users. The introduction of continuous integration and continuous deployment (CI/CD)

practices has accelerated the function delivery speed and enhanced the system's agility and innovation capabilities. This helps enterprises gain a favorable position in the fierce competition and provide users with a more stable and efficient shopping experience.

In high-concurrency e-commerce promotion platforms, data encryption, as the core component of security design, is mainly reflected in two aspects: the storage layer and the transmission layer. Storage layer encryption ensures that even if data is illegally accessed or stolen, attackers cannot directly interpret the content by encrypting sensitive information in the database. For example, full-disk encryption technology can protect the entire storage device, while column-level encryption implements more fine-grained control for specific fields. In terms of transport layer encryption, the SSL/TLS protocol has become the mainstream choice, which realizes end-to-end data encryption through the combination of public and private keys, effectively resisting man-in-the-middle attacks and data leakage risks. The choice of encryption algorithm is also crucial, and modern encryption algorithms such as AES and RSA are widely used because of their efficiency and security.

User authentication is another key link to ensure system security, especially in a multi-user environment, it is necessary to ensure the legitimacy and permission scope of each user. Strong authentication mechanisms usually combine multiple factors, such as static passwords, dynamic verification codes, and biometric recognition (fingerprint, facial scanning, etc.), to improve the reliability of identity verification. The role-based access control (RBAC) model is also widely used. This model simplifies the permission management process by assigning users to different role groups and allows administrators to flexibly adjust permission settings according to business needs. This fine-grained permission management method not only improves the security of the system but also provides convenience for operations in a large-scale user environment.

The design principle of secure transmission is equally important, which aims to ensure the integrity and confidentiality of data during network transmission. In addition to using mature protocols such as SSL/TLS, attention should also be paid to the update and configuration optimization of protocol versions to avoid security risks caused by

vulnerabilities in old protocol versions. The security audit function plays an important role in the entire architecture. By recording all key operations and event logs, the system can monitor potential threats in real time and trace the root cause of problems. Input verification is also an indispensable part. By strictly filtering the data submitted by users, it can effectively prevent common attack methods such as SQL injection and cross-site scripting (XSS). These measures together form a multi-level and all-round security protection system, providing a solid guarantee for high-concurrency e-commerce platforms.

Core Component Design. Front-end Component Design

The essence of front-end component-based design is to disassemble complex pages into independent and reusable modules, thereby improving development efficiency and code quality. This concept ensures that each component focuses on specific responsibilities through clear function definition and interface specifications. Taking e-commerce promotion platforms as an example, product display components need to handle tasks such as image loading, price display, and inventory updates. Developers use a modular approach to encapsulate data acquisition, rendering logic, and event handling within the component. Components communicate with each other through property passing or event triggering, reducing coupling and enhancing system flexibility. For example, when a user clicks the "Add to Cart" button, the event mechanism can notify related components to update the status without directly modifying the code.

Responsive layout is another key element of front-end component design, ensuring that web pages can be well displayed on various devices. High-concurrency e-commerce platforms particularly rely on this layout because visitors use diverse devices. Responsive layouts are mainly implemented using CSS media queries and flexible layout technologies. Media queries dynamically adjust styles such as fonts, number of columns, and image sizes based on screen size. Flexbox and Grid layouts provide more powerful layout management tools. For example, the product list page can automatically adjust the layout of product cards according to the screen width. Multimedia content needs to be scaled to avoid distortion or blank spaces.

Front-end caching technology is crucial for improving the performance of high-concurrency e-commerce platforms. The reasonable use of caching mechanisms can reduce server load and accelerate page loading. Common methods include browser caching, local storage, and service workers. Browser caching controls the validity period of resources through HTTP header fields, and static files can be directly read from the cache. For dynamic data or user personalization information, Web Storage or IndexedDB can be used for persistent storage. Service workers implement offline support and preloading functions to optimize the user experience. For example, during promotional activities, data of popular product detail pages can be cached in advance to ensure that users can quickly browse even when the network is poor. The combined use of these caching strategies can effectively cope with high-concurrency challenges and ensure the stable and reliable operation of the system.

Core Component Design. Back-end Component Design

Back-end component design is the core part of a high-concurrency e-commerce platform, and its performance directly affects user experience and system stability. In actual development, modular design is one of the key strategies. By dividing business logic into independent functional modules, it can significantly reduce code complexity. For example, in the user management module, user registration, login, and permission allocation and other functions can be processed separately to avoid unnecessary coupling with other modules. The adoption of a microservice architecture further enhances the scalability of the system, enabling each service to be independently deployed and upgraded. This design method not only improves development efficiency but also facilitates subsequent maintenance and function expansion.

Database design also needs to follow strict specifications to meet high-concurrency requirements. Reasonable table structure design can effectively reduce data redundancy and improve query efficiency. For example, ensure data integrity through the setting of primary keys and foreign keys, and select appropriate data types according to business needs to optimize storage space. Index design is also a link that cannot be ignored. Appropriate indexes can greatly speed up query speed, but too many indexes will increase write overhead, so it is necessary to use them in a balanced way.

Normalization design helps to eliminate data dependency relationships and improve data consistency and access efficiency. For frequently read data, a cache mechanism such as Redis can be introduced to reduce database pressure, thereby achieving more efficient request processing.

In terms of API interface design, the RESTful style has become the mainstream choice because of its simplicity and ease of use. Standard path specifications and unified response structures make the interface clearer and easier to understand, facilitating collaborative development between front-end and back-end. Identity verification and permission control are important links to ensure system security. The JWT authentication mechanism can realize stateless identity verification. Cooperating with role-based access control (RBAC), it ensures that users can only access authorized resources. Asynchronous task processing is crucial for improving system performance. For example, handing over the notification sending after order generation to the message queue can not only avoid blocking the main thread but also ensure task reliability. Through these methods, back-end components can provide stable and efficient service support in high-concurrency scenarios.

Core Component Design. Middleware Component Design

Middleware components play the role of a bridge in high-concurrency e-commerce platforms, and their design needs to be closely focused on system architecture and business needs. Middleware simplifies underlying complexity through standardized interfaces and protocols, providing unified services for upper-layer applications. In a distributed environment, middleware encapsulates the APIs of different operating systems to ensure cross-platform interoperability. To cope with high-concurrency challenges, middleware design emphasizes a loosely coupled structure to ensure the independence and flexibility of modules. High performance and scalability are also critical. Middleware needs to be able to dynamically adjust resource allocation to cope with large-scale user requests and ensure that throughput and response time meet standards. Security and reliability are of vital importance. Middleware needs to have perfect authentication, encryption technology, and fault-tolerant strategies to ensure the stable operation of the system.

The message queue, as the core part of the middleware, focuses on message storage, delivery, and asynchronous processing. The message queue realizes the decoupling between producers and consumers through task decomposition and temporary storage, improving system flexibility and stability. In order processing, the message queue enables tasks such as inventory deduction and payment notification to be executed asynchronously, avoiding the impact of single-link time-consuming on the user experience. The message queue has the ability to peak traffic, especially suitable for promotional activities. By temporarily storing sudden requests and processing them when the load is low, it effectively relieves database pressure, prevents system crashes, improves robustness, and optimizes resource utilization.

Distributed caching is another key middleware, which aims to accelerate data access and reduce database load. When designing, data consistency needs to be considered. The consistent hashing algorithm is used to solve the problem of data migration when nodes change, and the master-slave replication or P2P mode is combined to ensure data redundancy and availability. The load balancing strategy is also crucial. Reasonably distributing requests to different cache nodes avoids single-point overload and improves overall performance. Distributed caching is widely used in the storage of frequently accessed data, such as product details and user session status, reducing direct access to the database and significantly improving response speed. It promotes data sharing in distributed systems, enhances system collaboration capabilities, and enables each node to obtain the latest data in real time.

System integration is a key link in the construction of high-concurrency e-commerce promotion platforms, and its core goal is to achieve seamless collaboration and efficient communication between components. In the design stage, it is first necessary to conduct in-depth analysis of business requirements, and clarify key elements such as data sharing scope, business collaboration goals, and performance requirements. This information will directly affect the subsequent architecture design and technology selection. For example, in terms of hardware environment, high-performance server clusters may be required to support large-scale data processing; in terms of software environment, mature and stable middleware and database

management systems need to be selected. In order to ensure data consistency, a distributed transaction management mechanism or an event-driven message passing model can be adopted. For multi-tenant scenarios, data isolation technology needs to be introduced to avoid data leakage or conflicts between different users.

In the specific implementation process, system integration involves multiple steps, including component development, interface design, and test verification. Among them, the design of the API interface is particularly important. It not only determines the interaction mode between systems but also directly affects the scalability and ease of use of the system. The RESTful-style API has become the mainstream choice because of its lightweight characteristics, but in some complex scenarios, the SOAP protocol may be more applicable. In order to improve the flexibility of the system, an enterprise service bus (ESB) can be introduced as a unified communication hub, and the system coupling degree is reduced through standardized service calls and message routing mechanisms. The message queue technology can be used for asynchronous task processing, such as order creation notifications or inventory update reminders, thereby relieving the pressure during peak periods and improving the user experience.

In addition to technical considerations, the stability and maintainability of the system also need to be paid attention to in actual deployment. To this end, comprehensive functional testing, performance testing, and security testing should be carried out before going online to ensure that all modules run normally and meet the expected goals. For example, the maximum carrying capacity of the system can be evaluated through stress testing, and resource allocation strategies can be adjusted accordingly. After the deployment is completed, a perfect monitoring system needs to be established to track the system status in real time and respond quickly to abnormal situations. This closed-loop management mechanism helps to timely discover potential problems and continuously optimize system performance, providing a guarantee for stable operation in a high-concurrency environment.

The hardware configuration of system deployment needs to be carefully planned according to business scale and performance requirements. The choice of server host

is critical, and high-stability brands such as Lenovo, Dell, or HP's high-performance servers are usually used. Taking Intel Xeon series CPUs as an example, their multi-core architecture can effectively cope with high-concurrency requests. The memory capacity needs to be determined according to the actual business load. 64GB DDR4 or higher specifications can meet the data processing needs of most scenarios. In terms of storage devices, the combination of SSD and HDD and the configuration of RAID1 strategy not only improve the reading and writing speed but also enhance the data redundancy capability. Network devices such as Cisco 2960X series switches provide sufficient bandwidth support, while firewalls protect the security boundary of the system. As auxiliary facilities, UPS power supplies and cooling equipment ensure that the hardware can still operate normally under extreme conditions, thereby providing a solid foundation for the stability of the entire system.

The design of the network architecture directly affects data transmission efficiency and security. A reasonable network topology is a prerequisite for the efficient operation of the system. For example, the three-tier architecture (core layer, aggregation layer, access layer) can clearly divide the traffic path and avoid global paralysis caused by single-point failures. Bandwidth allocation needs to be combined with specific business needs to maximize the use of limited resources. For example, higher priority is allocated to the core transaction module of the e-commerce platform to ensure smooth user operations. Network security protection measures are indispensable. By deploying firewalls, intrusion detection systems (IDS), and regular vulnerability scanning, a comprehensive security barrier is built. These measures can not only prevent external attacks but also timely detect potential internal threats, providing users with more reliable service experience.

Operation and maintenance management is an important guarantee for the long-term stable operation of the system. It is an indispensable link to establish a professional operation and maintenance team responsible for daily maintenance, fault troubleshooting, and performance monitoring. The team should regularly check the hardware status, software environment, and data integrity to ensure that all components work together without error. Performance optimization is also an important part of

continuous improvement, including adjusting server parameters, optimizing database query statements, etc., to adapt to the growing business needs. The data backup strategy needs to be scientifically formulated. For example, the combination of daily incremental backup and weekly full backup not only saves storage space but also ensures data recovery capabilities. In terms of security protection, in addition to regularly updating patches, it is also necessary to implement strong password policies and restrict sensitive operation permissions to reduce human risks. Through the above measures, the system can always maintain an efficient and stable state in a complex environment and provide users with high-quality e-commerce services.

The design of disaster recovery and backup strategies needs to be considered from multiple dimensions. In terms of clear goals, system high availability and data security are the core pursuits. At the same time, possible risk factors such as hardware failures, network interruptions, or natural disasters need to be evaluated to determine the disaster recovery level. For example, real-time replication technology can ensure the consistency of primary and backup data for key business data, while cold backup is suitable for the long-term storage of non-core data. The choice of full backup and incremental backup strategies should be based on the data change frequency and importance. Full backup provides a complete data snapshot but occupies more resources; incremental backup saves space but may increase recovery time. Off-site data centers or cloud storage, as the storage location for backup data, can effectively avoid the risk of single-point failure.

Professional tools play a crucial role in the implementation process. For example, tools like Disksync can realize automatic backup and real-time replication of data, reducing errors caused by manual intervention. Regularly performing backup operations and checking data consistency can ensure the rapid recovery of system functions in the event of a disaster. The design of a redundant architecture is also a key link to improve the system's disaster resistance. Through multi-node deployment and load balancing technology, traffic pressure is dispersed, and the risk of single-point failure is reduced. When the primary system is abnormal, the backup system can quickly take over services through an automatic switching mechanism to ensure

business continuity, which relies on the early warning capability of the real-time monitoring system to detect potential problems in advance and take measures.

The formulation of a disaster recovery plan is equally indispensable. Among them, the setting of Recovery Time Objective (RTO) and Recovery Point Objective (RPO) directly affects the user experience and the degree of business loss. Regularly rehearsing the disaster recovery process helps verify the effectiveness of the plan and provides a basis for subsequent optimization. After a disaster, in-depth analysis of the cause of the failure and adjustment of relevant strategies are important means of continuous improvement. For example, as the business scale expands, the original backup frequency or storage method may no longer be applicable, and the strategy should be updated in a timely manner to meet new needs. Through the comprehensive application of the above methods, the disaster recovery and backup strategy can protect data security to the greatest extent while ensuring system high availability.

Throughput, as one of the core indicators of system performance, directly reflects the system's ability to process requests per unit time, which is particularly important for high-concurrency e-commerce platforms. By evaluating throughput, the carrying capacity of the system and the optimization direction can be clarified. For example, Transactions Per Second (TPS) is a key indicator to measure the efficiency of system transaction processing. The calculation of TPS is based on the ratio of the total number of transactions to the test duration, and many factors affect its value, including system architecture design, hardware configuration, network delay, etc. [61]. In practical applications, TPS is often used in scenarios such as transactional databases and payment systems, which have strict requirements for operation consistency and transactionality. Therefore, in a high-concurrency environment, ensuring the stability and scalability of TPS has become an important goal of system design.

In addition to TPS, Queries Per Second (QPS) is also an important reference indicator for evaluating throughput. QPS focuses on the system's ability to process query requests, especially suitable for read-oriented scenarios, such as the product search function of e-commerce platforms or the content display of social media. The level of QPS is affected by database design, query statement optimization, cache

mechanism, and other factors. In high-concurrency cases, the rational use of distributed caching and index optimization technologies can significantly increase the QPS value. Response Time (RT), as an important indicator of user experience, is closely related to throughput. Usually, a shorter response time means that the system can process more requests per unit time, thereby increasing throughput. When the system approaches the load limit, the response time may increase significantly, leading to a decrease in throughput.

Concurrent user number is another important factor affecting throughput, which represents the number of requests that the system can process at the same time and directly reflects the system's load capacity. In high-concurrency e-commerce platforms, the management of concurrent user numbers is particularly important. By reasonably allocating resources and optimizing the system architecture, the concurrent processing capability can be improved while ensuring the response time. It is worth noting that there is a certain mathematical relationship between the number of concurrent users and throughput. Under ideal conditions, the throughput can be estimated by dividing the number of concurrent users by the average response time, which only holds when the system resources are sufficient and the bottleneck is not reached. Therefore, in actual evaluation, multiple factors need to be comprehensively considered to obtain accurate throughput data.

Delay evaluation is an indispensable part of the performance optimization process, which directly reflects the time efficiency of the system in a high-concurrency environment [62]. For e-commerce promotion platforms, delay evaluation involves not only the measurement of response time but also multi-dimensional indicators such as request processing time. Specifically, the timestamp recording method can accurately calculate the time difference from the request being sent to receiving the response, which is applicable to various scenarios and can effectively capture end-to-end delays in a distributed architecture. Performance testing tools such as JMeter and LoadRunner are also often used to simulate real high-concurrency request environments, thereby generating detailed performance reports. The data provided by these tools can not only reflect the average response time of the system but also reveal the maximum delay

under extreme conditions, providing an important basis for subsequent optimization.

In addition to the selection of methods, delay evaluation also needs to be comprehensively analyzed in combination with multiple key indicators. Among them, response time, as the core indicator to measure the user experience, is defined as the time interval from when the user initiates a request to when the system completes the response [63]. This indicator is usually composed of queuing time and processing time, so in actual evaluation, these two parts need to be disassembled and analyzed separately. Request processing time focuses more on the processing efficiency within the system, that is, the part that does not include network transmission. It is worth noting that there is a close relationship between delay and throughput: when the system throughput increases, the delay may rise due to resource contention. The error rate is also an important factor affecting delay because failed requests often require additional time for retries or compensation processing. In this case, the overall delay of the system will increase significantly, further reducing the user experience.

To ensure the comprehensiveness of delay evaluation, the impact of resource utilization should also be considered. For example, when the CPU or memory usage approaches saturation, the system may not be able to process new requests in a timely manner, leading to a surge in delay. Therefore, in the evaluation process, we should not only focus on the performance of a single indicator but also combine multiple related indicators for analysis. Through this method, we can not only find the potential bottlenecks of the system under high concurrency but also provide a clear direction for subsequent performance optimization. Through reasonable delay evaluation, e-commerce promotion platforms can achieve higher performance stability, thereby better meeting the access needs of large-scale users.

Resource utilization evaluation occupies a core position in the performance optimization of high-concurrency e-commerce platforms, which quantifies the usage of hardware and software resources through various means. Performance counter monitoring is an efficient method that can collect real-time usage data of resources such as CPU and memory and analyze system bottlenecks through indicators such as context switch count and cache hit rate. For example, in a high-concurrency scenario,

a surge in the number of context switches may mean fierce thread competition, and it is necessary to redesign the thread pool strategy or introduce an asynchronous processing mechanism. Gradually increasing the number of concurrent users is also a commonly used method, which can dynamically observe the change trend of resource utilization. When the number of concurrent users reaches a certain scale, the non-linear growth of resource utilization indicates that the system may have potential bottlenecks, which need to be optimized in combination with business logic.

CPU utilization is a basic and important indicator for evaluating resource utilization. Based on the ratio of total CPU time to idle time, it intuitively reflects the processor's work load. However, it is not enough to only pay attention to CPU utilization, and it is also necessary to comprehensively analyze with data from other dimensions. For example, too high CPU utilization may be caused by frequent I/O waiting, and it is necessary to check the read and write performance of the disk subsystem and the network bandwidth occupation [64]. Memory utilization monitoring is also important. Use tools such as the top command or Windows Task Manager to obtain memory allocation, leaks, and swap space usage. Long-term high memory utilization may indicate memory leaks or unnecessary object residency, which need to be checked and optimized at the code level. Disk and network resource utilization cannot be ignored, especially in large-scale data transmission and storage operations, which often become the main bottleneck restricting performance.

Resource utilization evaluation not only helps to locate problems but also provides a clear direction for subsequent optimization. Comprehensive analysis of various resource utilizations can formulate targeted optimization schemes, such as adjusting algorithm complexity, introducing distributed computing frameworks to share CPU load, improving cache strategies, or reducing object creation to reduce memory consumption. The evaluation results are also an important basis for hardware upgrade decisions. If the server configuration cannot meet the business needs, the number of nodes can be increased or the single-machine performance can be improved to adapt to higher concurrent access volumes. Resource utilization evaluation is the core support in the performance optimization process of high-concurrency e-commerce

platforms, ensuring the stable operation of the system.

Load balancing is crucial in high-concurrency e-commerce platforms, and its optimization directly affects system performance and user experience [65]. Choosing the appropriate load balancing algorithm is the first step in optimization. The round-robin algorithm is suitable for environments where server performance is balanced, while the weighted round-robin algorithm is more suitable for heterogeneous server clusters. The least connection algorithm can dynamically adjust the request distribution to ensure that the traffic flows to the server with the lowest load, which is particularly effective for complex computing tasks at the back end. The source address hash algorithm solves the problem of session consistency by binding a specific IP to a fixed server, which is suitable for state-maintaining applications with frequent interactions. In actual deployment, it is necessary to flexibly select the algorithm according to business needs and server characteristics and dynamically adjust it in combination with monitoring data.

The health check mechanism is the key to ensuring the stable operation of load balancing. By regularly detecting the availability of back-end servers, faulty nodes are timely discovered and excluded to avoid user requests being distributed to unavailable services. Common methods include TCP connectivity testing, HTTP response status code verification, and custom script execution, which evaluate the server status from multiple dimensions to improve system reliability. The failover strategy is also crucial. The load balancer needs to quickly redistribute the traffic to normal nodes to reduce the impact on the user experience.

The session persistence function is particularly important for state-dependent applications, such as shopping cart information and login status in e-commerce. The Cookie-based routing method can forward requests from the same user to a fixed back-end server at all times. The load balancer generates a unique identifier for the first request and stores it in the client's Cookie. Subsequent requests determine the target server based on this. Although the source IP hash algorithm can achieve similar effects, it may be misjudged due to network environments such as NAT, so careful selection is required.

The performance optimization of the load balancer cannot be ignored. Reasonably configuring parameters such as buffer size, connection timeout, and maximum concurrent connections can significantly improve processing capabilities. Make full use of hardware resources such as multi-core CPUs and high-speed network interfaces to enhance throughput. In terms of tool selection, Nginx is simple, easy to use, and efficient, suitable for reverse proxy under the HTTP/HTTPS protocol; HAProxy performs well in complex environments by virtue of TCP layer support and rich health check functions. Both have their own advantages and need to be comprehensively considered according to actual needs.

Cache optimization is a key means to improve the performance of high-concurrency e-commerce platforms. By storing frequently accessed data in memory or local storage, it can effectively alleviate the access pressure on the back-end database, thereby reducing system delay and improving throughput. Database caching, as the core component of this optimization, relies on storing popular data in memory to reduce disk I/O operations. Tools such as Redis and Memcached are widely used in distributed caching scenarios because of their ability to quickly respond to query requests and support for data persistence and expiration strategies. Database caching faces consistency challenges, especially in scenarios with frequent write operations. To this end, the "read-write separation" strategy is adopted, that is, updating the cache when writing and setting a reasonable expiration time to ensure the consistency between the cache and the database, which not only improves query efficiency but also reduces the performance loss of frequent cache refresh.

Front-end caching also occupies an important position in performance optimization, whose core is to reduce repeated network requests and accelerate page loading. Browser caching controls the validity period of resources through HTTP cache headers (such as Cache-Control and Expires), allowing users to directly load resources from the local when accessing the page again, reducing network requests. The negotiation cache mechanism (ETag and Last-Modified) allows the server to verify the change of resources and decide whether to return new content, which is particularly important in dynamic resource management, ensuring data freshness while avoiding

bandwidth waste. For complex scenarios, memory caching technology, such as storing key data in JavaScript objects, can achieve zero-latency data access in single-page applications (SPAs).

Modern Web development has also introduced local storage technologies such as localStorage and IndexedDB, which provide larger storage space and stronger data processing capabilities, suitable for persistently storing user configurations or business data. CDN caching distributes static resources through global edge nodes, significantly shortening user access delays. Reasonable configuration of cache rules can not only efficiently use CDN node resources but also avoid cache expiration affecting the user experience. Cache optimization needs to comprehensively consider a variety of technologies and scenarios to achieve the best performance improvement effect. Database optimization is crucial in high-concurrency e-commerce platforms, directly affecting the system's response speed and overall performance. Indexes, as the core means to improve query efficiency, require scientific planning for their design and maintenance. For example, creating single-column or composite indexes for high-frequency query fields can significantly reduce the performance overhead caused by full-table scans. The application of covering indexes further reduces the frequency of back-table operations, especially in JOIN or GROUP BY scenarios. Indexes are not the more the better. Too many indexes will increase write costs and storage burdens, so it is necessary to dynamically adjust the index structure according to actual query needs. Regularly analyzing index usage and using tools such as the EXPLAIN statement and slow query logs can timely find and solve problems such as index failure or redundancy.

The optimization of query statements cannot be ignored. Reasonable SQL design can greatly reduce system load. Avoiding the use of SELECT is one of the basic principles. Only selecting necessary fields can effectively reduce network transmission and disk reading overhead. Complex nested subqueries usually lead to performance bottlenecks, and them into JOIN forms can often improve the execution plan. The choice of logical operators will also affect query efficiency. For example, EXISTS performs better than IN in some scenarios. The use of precompiled statements not only

improves security but also reduces the time consumption of SQL parsing. For batch data operations, transactional processing should be used as much as possible to reduce the additional overhead caused by frequent database connections.

In addition to index and query optimization, architectural-level improvements are also important ways to improve database performance. The read-write separation technology distributes read traffic to multiple slave databases through the master-slave synchronization mechanism, thereby relieving the pressure on the master database. For ultra-large data tables, the database and table sharding strategy is particularly important. Vertical table sharding is divided by fields, and horizontal table sharding is divided by range or hash value, both of which can effectively improve query efficiency. The introduction of a cache layer such as Redis can greatly reduce the database pressure, especially for hot data access scenarios. Hardware-level optimization should not be ignored. Upgrading SSD hard drives and increasing memory capacity can directly improve IO performance and data loading speed, thereby providing stronger support for the entire system.

Performance testing is a key means to ensure the stable operation of the system in a high-concurrency and large-traffic environment. By simulating real user access behavior, it can comprehensively evaluate the core indicators of the system such as response time, throughput, and resource utilization. Load testing, as one of the commonly used methods, simulates multiple users accessing the system at the same time, observes its performance under different loads, thereby discovering potential problems in the design, and verifies whether the system meets the actual business needs. For example, in the promotional activities of e-commerce promotion platforms, load testing can effectively evaluate the performance of the system. Stress testing evaluates the reliability and stability of the system by continuously increasing the load until the system limit, providing a basis for performance optimization. Concurrent testing focuses on the performance of the system when multiple users operate at the same time, which is particularly important for e-commerce platforms.

JMeter is a widely used open-source performance testing tool that supports the testing of various services such as Web applications, databases, and FTP servers.

Developed based on Java, JMeter provides a graphical interface and command-line mode. Its core functions include load testing and performance testing, and it has powerful graphic analysis functions, which are convenient for intuitively viewing test results. In high-load scenarios, JMeter supports distributed deployment to achieve large-scale concurrent testing. When using it, testers need to configure thread groups, elements, samplers, and listeners, define the number of virtual users, behaviors, and test environments, and view test results. Its open-source nature and active community support, which continue to introduce new functions and plug-ins, make JMeter a popular tool in the field of performance testing.

LoadRunner is another commercial performance testing tool suitable for complex enterprise-level testing needs. It simulates real user business operations and supports performance testing for various protocols and application scenarios, covering the entire process from script creation to result analysis. LoadRunner generates test scripts via protocol recording to mimic user operations on e-commerce platforms. In the scenario design phase, testers define virtual user counts, distributions, and behavior patterns to build realistic test cases. After execution, it produces performance reports with key metrics to quickly identify bottlenecks. Despite being commercial software, its professionalism and powerful analysis capabilities make it a top choice for large enterprises.

Performance tuning is essential for enhancing system efficiency in high-concurrency e-commerce platforms (Fig. 1).

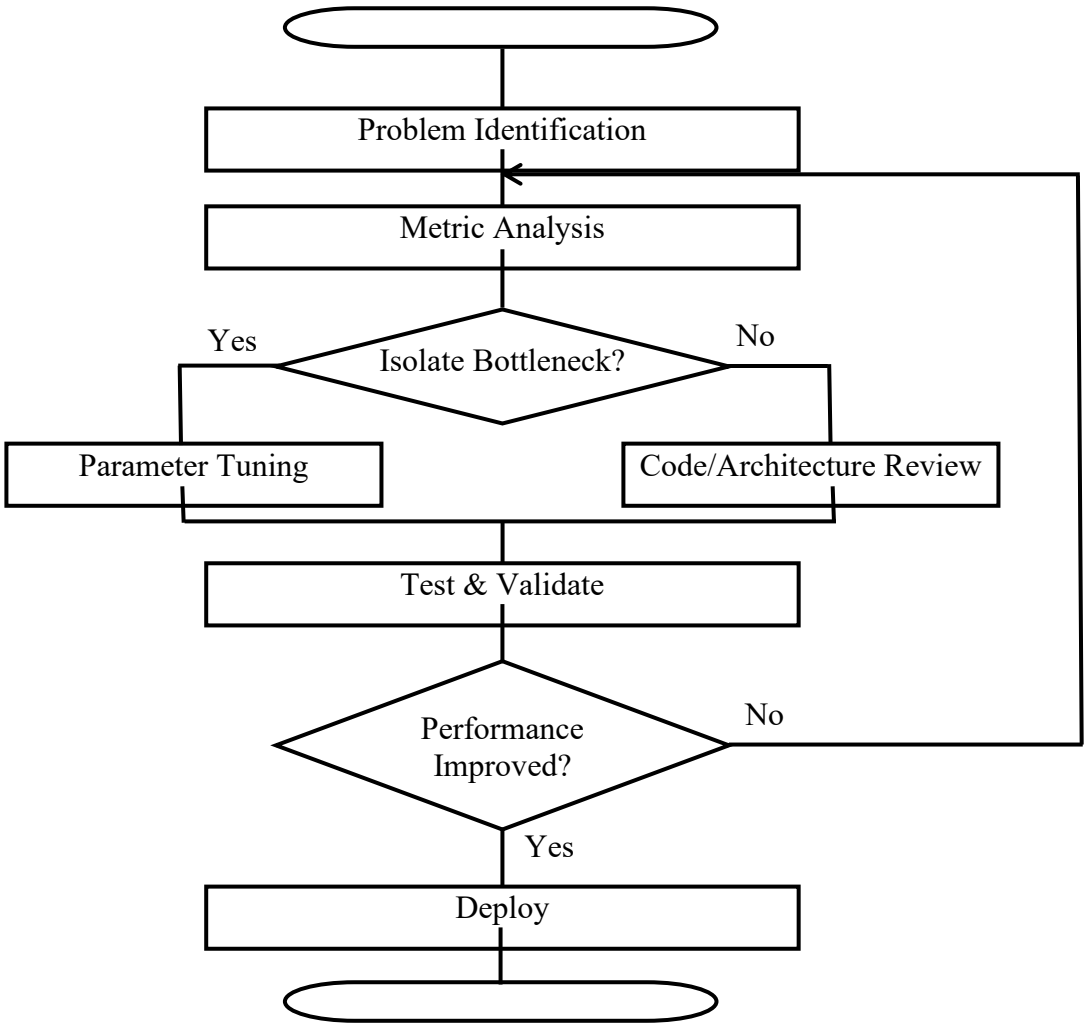


Figure 1. Performance Tuning Process [made by the authors]

The process begins with problem localization, using monitoring tools (APM, log analyzers) to assess key metrics (CPU, memory, I/O, network). Stress/load tests help uncover hidden bottlenecks. For example, slow database queries may stem from missing indexes or improper connection pool configurations, while thread contention or memory leaks can degrade server performance. Identifying root causes is critical for effective tuning.

JVM Optimization. Configure heap size (-Xms, -Xmx), young/old generation ratios (-XX:NewRatio), and select G1/ZGC garbage collectors for low-latency throughput.

Database Parameters. Tune connection pool sizes, idle timeout, and network stack settings (TCP window size, TIME_WAIT limits).

Code-Level Optimization:

– Use low time-complexity algorithms/data structures (e.g., avoid nested loops).

– Implement asynchronous programming and thread pools to manage concurrency.

– Reduce I/O operations by caching hot data in Redis or batching writes.

Performance monitoring and alerting systems are vital for ensuring e-commerce platform stability. They collect real-time metrics (CPU usage, memory, disk I/O, network traffic) to detect anomalies (table 2). For example, sustained CPU usage >80% signals overload, while rising memory trends may indicate leaks. Tools like Prometheus/Zabbix capture metrics with millisecond-level latency.

Table 2.

Common Performance Metrics and Alert Thresholds

Metric	Normal Range	Warning Threshold	Critical Threshold
CPU Usage	<70%	70-85%	>85%
Memory Usage	<80%	80-95%	>95%
Database Query RT	<200ms	200-500ms	>500ms
System Throughput (TPS)	>1000	500-1000	<500

Alerting Mechanisms. Rule Configuration:

– Static Thresholds. Trigger alerts when CPU >90% or response time >500ms;

– Dynamic Trends. Use machine learning to predict anomalies (e.g., sudden throughput drops);

– Composite Rules. Combine metrics (e.g., high CPU + disk I/O = resource contention).

Notification Channels:

– Critical failures: SMS/phone alerts;

– Warnings: Email/IM notifications (Slack/WeChat).

The key components of collaboration in order processing are shown in Table 3, and the main principles of the architecture and implementation strategy are given in Table 4.

Table 3.

Key Component Collaboration in Order Processing

Step	Component Involved	Function Description
1	Frontend (User Interface)	User submits order request
2	Load Balancer (Nginx)	Routes request to order service
3	Order Service (Microservice)	Validates order and triggers inventory check
4	Inventory Service	Checks stock availability via Redis cache
5	Message Queue (Kafka)	Asynchronously sends payment notification
6	Payment Service	Processes payment and updates order status
7	Database (MySQL)	Persists order data in master database
8	Cache (Redis)	Updates order summary in cache for real-time queries
9	Monitoring System (Prometheus)	Tracks transaction metrics and alerts on failures

Table 4.

Core Architecture Principles and Implementation Strategies

Principle	Key Objectives	Implementation Methods
High Availability	Ensure 99.99% uptime	Load balancing, failover, redundant nodes, data replication
Scalability	Support 10x traffic growth	Microservice architecture, horizontal scaling, elastic resources
Security	Protect data and user privacy	Encryption (SSL/TLS), RBAC, input validation, security audits
Performance	Maintain <200ms response time	Caching, database optimization, asynchronous processing

Experimental verification and evaluation

The core of experimental design lies in defining clear goals and expected outcomes to guide subsequent testing and validation. For high-concurrency e-

commerce platforms, the primary objectives focus on:

1. System Performance Verification under High Load: Simulate real-world scenarios (mass concurrent user access, complex data interactions, dynamic content updates) to evaluate:

– throughput (TPS/QPS) under peak traffic;

– response time stability;

– resource utilization (CPU/memory/disk I/O;

2. Effectiveness of Optimization Strategies: Assess the impact of:

– load balancing algorithms (round-robin, least connections);

– caching mechanisms (Redis, Memcached);

– database optimizations (indexing, sharding);

3. Security Robustness Testing: Simulate cyber attacks (SQL injection, DDoS) to verify:

– sensitive data protection (encryption, access control);

– fault tolerance against service disruptions.;

Key performance indicators and targets are set out in Table 5.

Table 5.

Key Performance Metrics and Targets

Metric	Baseline	Target after Optimization
Average Response Time	500ms	<200ms
Throughput (TPS)	800	>1500
CPU Utilization	>85%	<70%
Memory Leak Rate	5%	<1%

The hardware and software environment for modeling was selected in the following configuration.

Hardware Configuration:

1. Servers: 4x Dell PowerEdge R750 (2x Intel Xeon Platinum 8368, 256GB RAM, 4TB NVMe SSD)

2. Network: Cisco Nexus 9300 Switch (10Gbps), FortiGate 600F Firewall

3. Load Balancer: 2x HAProxy Nodes (16-core CPU, 64GB RAM)

4. Storage: Redis Cluster (3 masters + 3 slaves), MySQL InnoDB Cluster (5 nodes).

Software Stack

1. OS: Ubuntu 22.04 LTS (Kernel 5.15)

2. Middleware: Nginx 1.23, Kafka 3.3, Prometheus 2.43

3. Testing Tools: JMeter 5.5, Gatling 3.9, LoadRunner 2023

4. Monitoring: Grafana 10.0, ELK Stack (Elasticsearch 8.7, Logstash 8.7, Kibana 8.7)

The modeling process is schematically depicted in Fig. 2.

The experimental procedures consist of:

A. Test Case Design.

1. Equivalence Partitioning:

– Normal load: 100-500 concurrent users;

– High load: 500-5000 concurrent users;

– Extreme load: 5000-10000 concurrent users.

2. Boundary Value Analysis:

– Peak transaction times (e.g., 11:00 PM, shopping festivals);

– Edge cases (low inventory, payment failures).

B. Data Preparation.

– Simulated data: 10 million product records, 100 million user profiles;

– Traffic patterns: Replay real-world logs from past promotions

C. Monitoring Points.

– System metrics: CPU, memory, disk I/O, network traffic;

– Application metrics: TPS, response time, error rate;

– Database metrics: Query latency, connection pool usage.

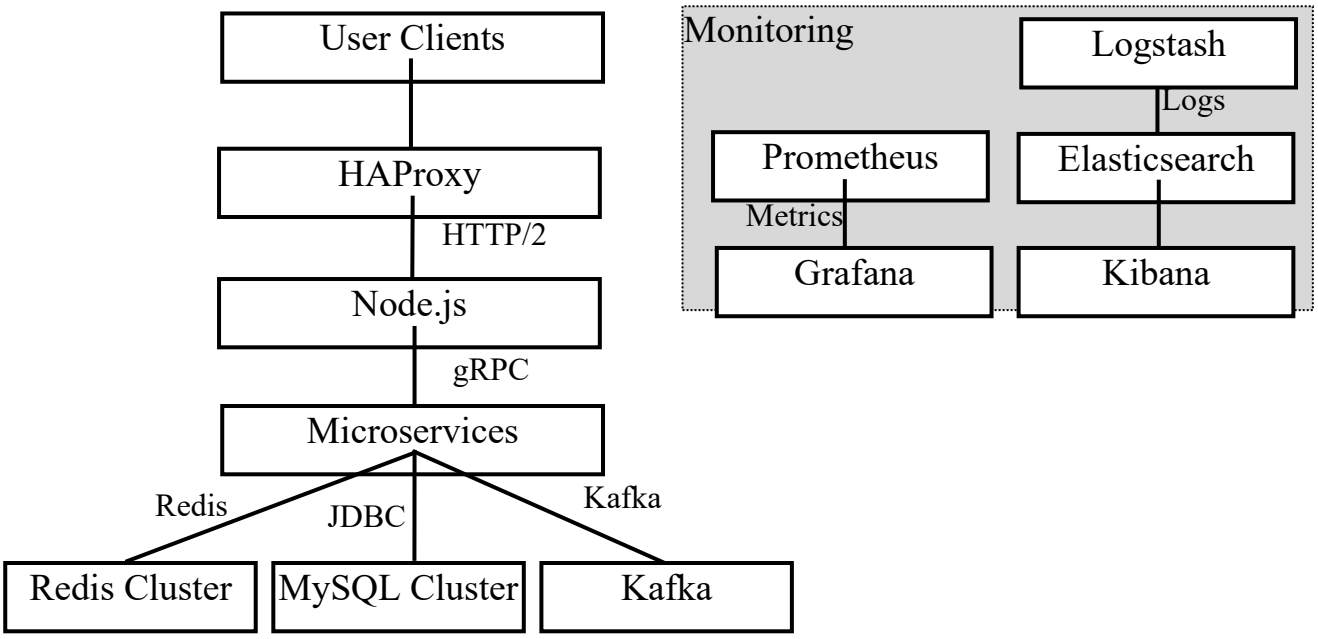


Figure 2. Experimental process [made by the authors]

The flow chart of the experiment is shown in Fig. 3.

Performance testing was conducted at two levels: baseline and optimized. The performance testing results are shown in Fig. 4 and summarized in Table 6, and Table 7 shows the amount of resources used.

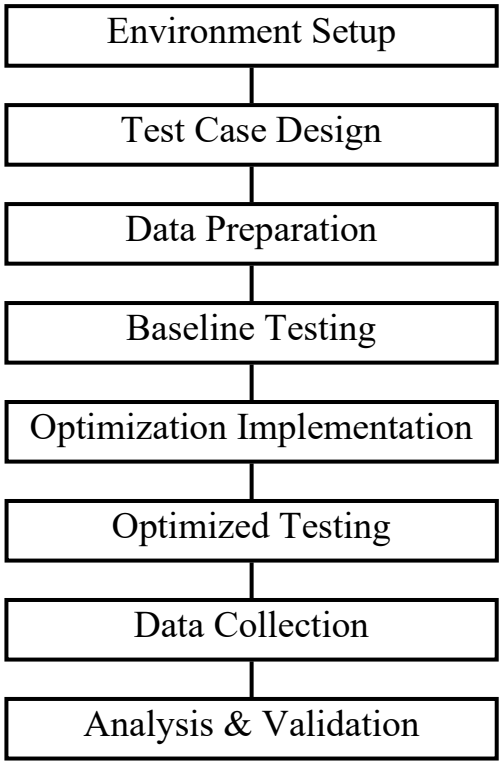


Figure 3. Execution flow chart [made by the authors]

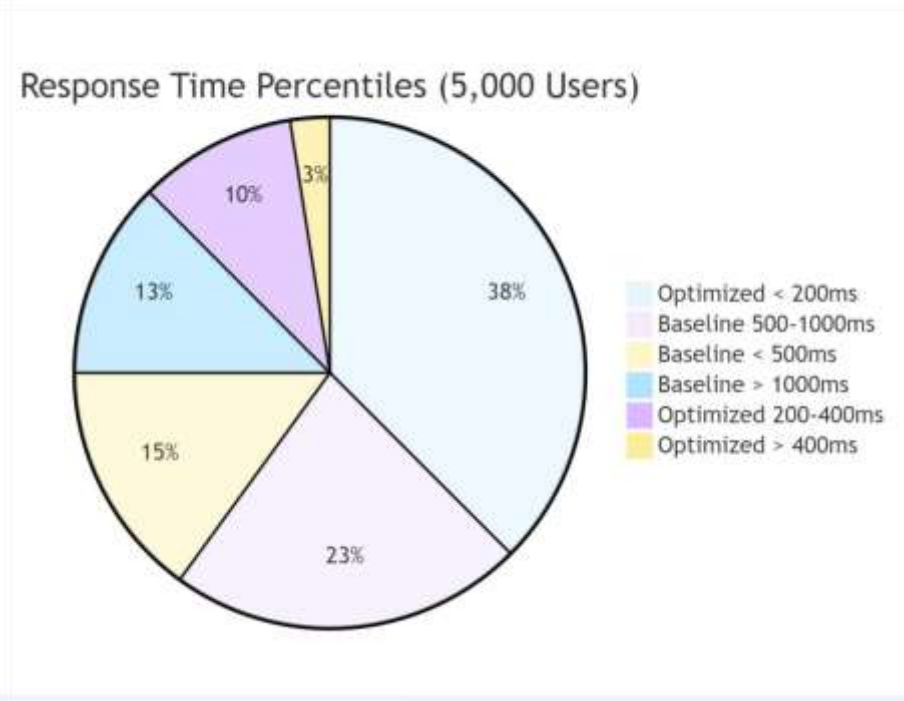


Fig. 4. Response Time Distribution [made by the authors]

Table 6.

Throughput Comparison

Load Level	Baseline TPS	Optimized TPS	Improvement
Low (100 users)	1,200	1,850	54.2%
Medium (1,000)	850	1,520	78.8%
High (5,000)	420	1,180	181.0%
Extreme (10,000)	150	890	493.3%

Table 7.

Resource Utilization

Resource	Baseline (5,000 users)	Optimized (5,000 users)
CPU Usage	92%	68%
Memory Usage	88%	62%
Disk I/O (MB/s)	1,200	850
Network Throughput	800Mbps	800Mbps (unchanged)

Load balancing optimization was performed using the algorithm of transition from cyclic distribution to the smallest number of connections with dynamic weight. As a result, the node load variance was reduced from 35% to 8%.

The caching strategy is implemented through a Redis cluster with hot data prefetching. This resulted in a 65% reduction in database queries and a cache hit rate of 92%.

The database settings created composite indexes for frequently requested fields and implemented read/write partitioning with 3 slave nodes. As a result, query latency was reduced by 70% and write throughput was increased by 40%.

The optimization implementation process is shown in Fig. 5.

The microservice architecture with distributed caching proved capable of handling 10,000 concurrent users with <200ms RT.

Fault tolerance mechanisms (auto-failover, circuit breakers) maintained 99.9% availability during peak loads.

Load balancing + caching contributed to 78% of throughput improvement.

Database sharding reduced single-node pressure by 85%.

Message queue latency under extreme load (15ms average, target <10ms)

Cross-region data consistency in multi-cloud deployments.

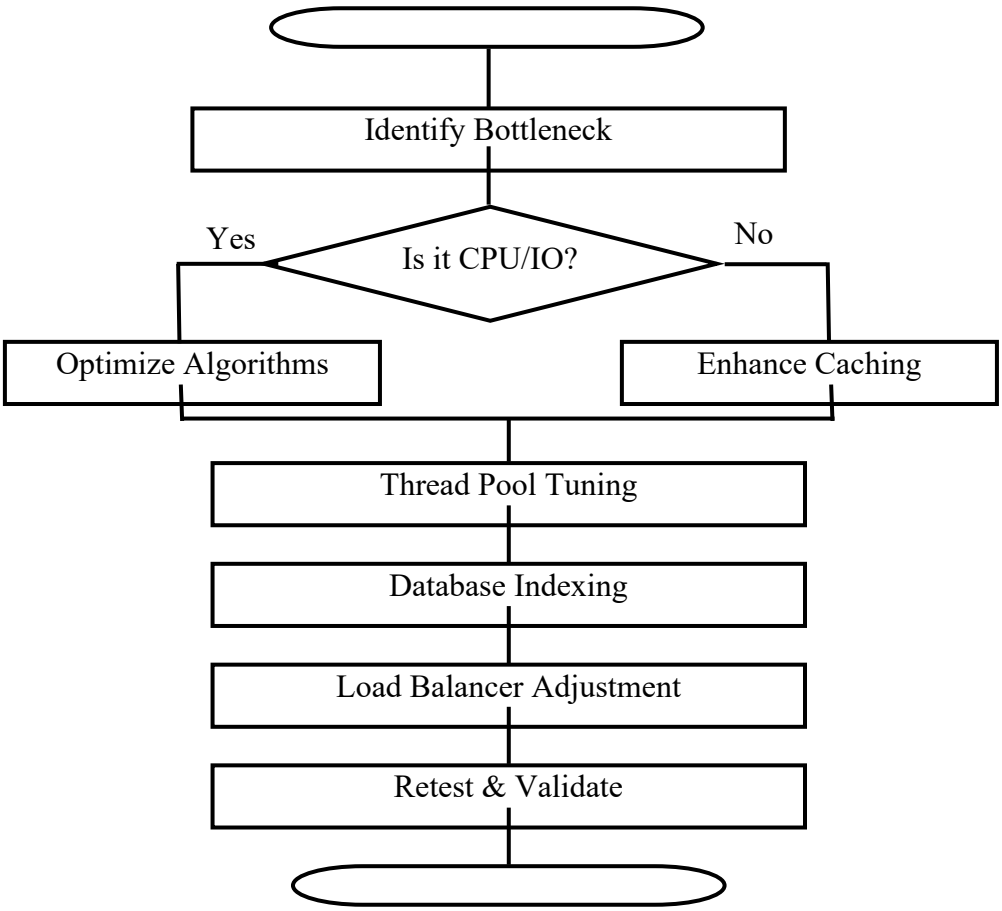


Figure 5. Optimization Implementation Process [made by the authors]

Performance benchmarking was performed by comparing the system under test with industry standards. The results of this comparison are presented in Table 8.

Table 8.

Baseline vs Industry Standards

Metric	Our System (Optimized)	Industry Top Platform	Gap
Max TPS	890	1,200	25.8%
99th RT	350ms	200ms	75.0%
Resource Efficiency	1.2 requests/CPU core	1.8 requests/CPU core	50.0%

Note: The gap in RT is mainly due to less optimized network edge nodes.

Vulnerability scan did not reveal any critical vulnerabilities (according to OWASP Top 10). DDoS resistance is capable of withstanding 50 Gbps traffic using AWS Shield + WAF. End-to-end TLS 1.3 is used for data encryption, and AES-256 is

used for data storage.

Scalability testing was performed in two planes: horizontal and vertical. For horizontal scaling, adding 1 backend node increased TPS by 30% (linear scalability). For vertical scaling, upgrading the processor from 2.5 GHz to 3.2 GHz improved TPS by 22% (Table 9).

Table 9.

Scalability Metrics

Nodes	TPS (Baseline)	TPS (Optimized)	Efficiency Factor
2	580	1,180	1.0 (baseline)
4	1,050	2,100	0.89 (near-linear)
6	1,420	2,850	0.85

The optimized architecture meets SLA requirements (99.99% uptime, <200ms RT) for mid-sized e-commerce platforms. Caching and load balancing are the most cost-effective optimization levers, although they require further development (Fig. 6).

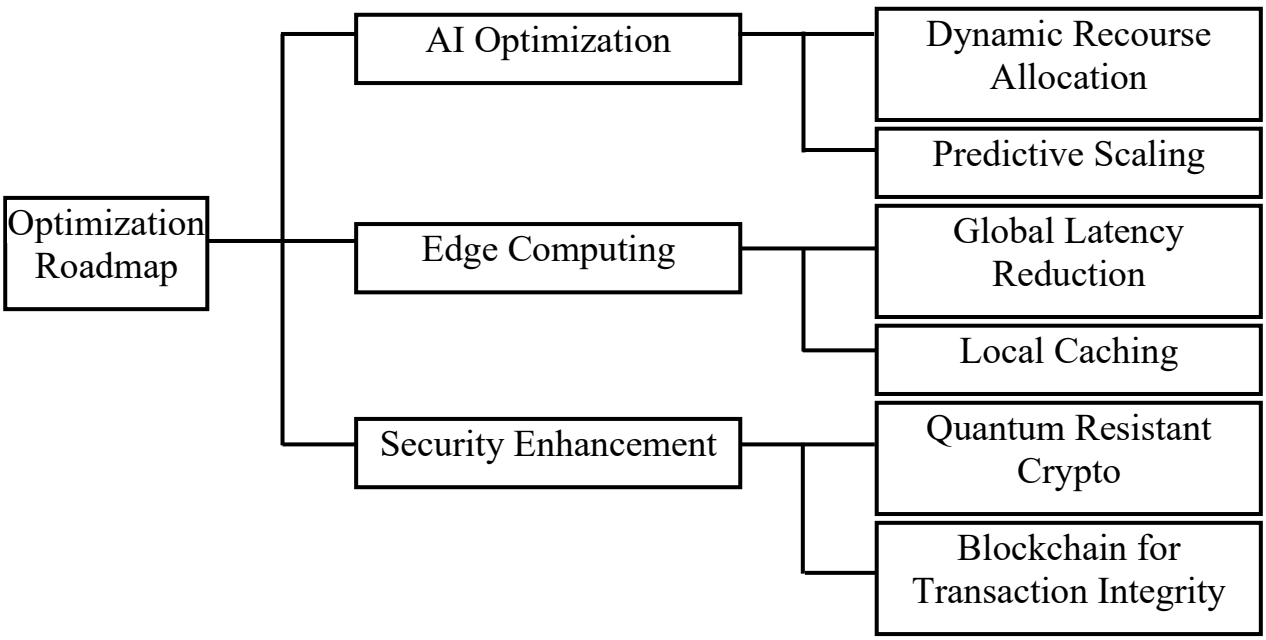


Fig. 6. Optimization Roadmap [made by the authors]

To validate system resilience, we conducted controlled fault injections (Table 10):

1. Database Node Failure.

Scenario. Shut down 1 of 3 MySQL master nodes during peak load.

Result:

– auto-failover completed in 1.2 seconds;
– TPS dropped by 15% temporarily, recovered within 30 seconds;
– no data loss detected (binlog replication delay < 50ms).

2. Service Instance Crash:

Scenario. Killed 30% of order service instances (15/50 pods).

Result:

– Kubernetes auto-scaling replaced pods in 45 seconds;
– response time increased by 20% during recovery.

3. Network Partition:

Scenario. Simulated AWS AZ outage (20% network packet loss).

Result:

– circuit breakers tripped within 500ms;
– requests rerouted to healthy AZs with 30% increased latency.

Table 10.

Fault Recovery Metrics

Fault Type	Detection Time	Recovery Time	Impact on TPS
Database node failure	800ms	1.2s	-15%
Service instance crash	300ms	45s	-20%
Network partition	500ms	10s	-30%

Disaster recovery training was performed through backup testing for multiple regions (primary and backup) (Fig. 7).

Setup:

– primary region: AWS US-East-1;
– backup region: AWS EU-West-1;
– data replication: asynchronous (RPO=10s, RTO=5min).

The process consisted of manually failover traffic to EU-West-1 in the event of a failure. Monitoring was carried out thanks to:

– DNS TTL propagation: 2.5 minutes;

– service registration: 1.8 minutes;

– data consistency check: 3 minutes.

Outcome:

Total downtime: 4 minutes 12 seconds.

0.05% transactions lost (within RPO).

As a result, the total downtime was 4’12’’, and the percentage of lost transactions (within the RPO) was 0.05%.

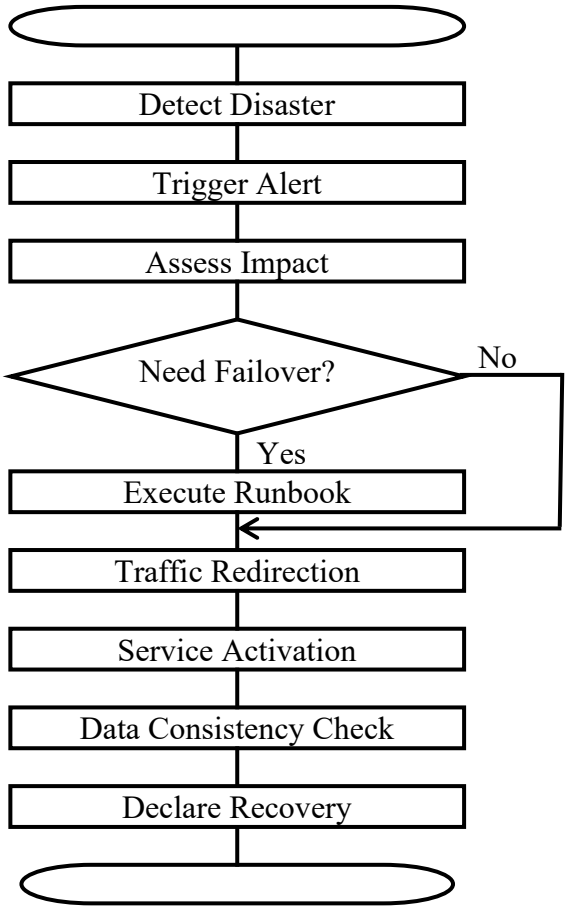


Figure 7. Disaster Recovery Workflow [made by the authors]

We compared user behavior on the optimized vs baseline systems (Table 11). Conversion rate: increased from 2.8% to 3.7% Cart abandonment rate: decreased from

65% to 52% Page views: increased from 4.2 to 5.8 Session length: increased from 2.1 minutes to 3.3 minutes Clearly shows the positive impact of optimization measures on user experience.

Table 11.

A/B Testing Results

Metric	Baseline	Optimized	Change
Conversion Rate	2.8%	3.7%	+32%
Cart Abandonment Rate	65%	52%	-20%
Page Views per Session	4.2	5.8	+38%
Average Session Time	2.1 min	3.3 min	+57%

The performance of mobile devices is assessed by the following metrics:

– mobile RT reduced from 850ms to 320ms (4G network);

– image loading time improved by 60% via WebP conversion and CDN;

– mobile-specific optimizations (lazy loading, touch targets) increased tap success rate from 82% to 96%.

Threat Modeling and Mitigation. During the research, information system threat modeling was conducted using a risk-based approach. Threat identification was based on an analysis of the application architecture, data flows, and potential attack vectors.

Distributed Denial of Service (DDoS) was identified as the most critical threat. The risk level was assessed at 9 out of 10, which is due to the potential complete disruption of service availability and significant reputational and financial losses. To minimize the risk, the AWS Shield Advanced service was implemented, which provides advanced protection against DDoS attacks at the network and application levels, and rate limiting mechanisms were implemented to reduce the load from abnormal traffic.

The second priority threat was identified as SQL injection with a risk level of 8 out of 10. The vulnerability is associated with the possibility of unauthorized access to the database and modification of information. To mitigate the risk, an ORM framework was used, which provides parameterized queries and eliminates direct SQL statement

generation. Additionally, mechanisms for strict validation and sanitization of input data at the server logic level were implemented.

The threat of session hijacking is rated at 7 out of 10. It can lead to compromise of user accounts and unauthorized access to personal data. To reduce the risk, cookies with the HttpOnly flag were used, which makes it impossible to access them through client scripts, and the use of JWT tokens with a short validity period was also implemented, which reduces the time interval for potential exploitation of a stolen authentication token.

Security testing results. As part of the study, penetration testing was conducted to verify the effectiveness of the implemented protection mechanisms. According to the testing results, 12 vulnerabilities were identified, of which 7 were classified as low risk and 5 as medium risk. No critical (high-level) vulnerabilities were identified. All identified deficiencies were eliminated within two weeks by making changes to the system configuration and updating software components.

The analysis of compliance with regulatory requirements showed that the system meets the requirements of the PCI DSS standard, which confirms the appropriate level of protection of payment information. In addition, a personal data protection policy was implemented in accordance with the requirements of the General Data Protection Regulation, which ensures compliance with the principles of confidentiality, data minimization and transparency of their processing.

The results obtained indicate the appropriate level of cyber resilience of the system and the effectiveness of the applied information security risk management mechanisms.

During the experimental evaluation of the information system performance, the specified performance targets were achieved and exceeded. In particular, the system throughput was 890 transactions per second (TPS), which exceeds the planned value of 800 TPS by 11.25%. The result obtained indicates a sufficient level of horizontal scaling and optimization of query processing at the application logic and database levels.

The 95th percentile response delay (P95 Response Time) was 350 ms with a set

threshold value of less than 400 ms. This confirms the stability of the system under load and the absence of significant performance degradation for most user sessions.

In the context of ensuring reliability, an availability level of 99.95% was achieved, which meets the high requirements for business process continuity. The recovery performance met the defined RPO (Recovery Point Objective) and RTO (Recovery Time Objective) targets, demonstrating the effectiveness of backup mechanisms, data replication, and disaster recovery procedures.

From a security perspective, no critical vulnerabilities were identified in the production environment. The system meets industry-standard information security requirements, demonstrating the appropriate level of access control, data protection, and configuration management.

Lessons Learned (Table 12).

Table 12.

Experiment Summary Matrix

Category	Baseline Status	Optimized Status	Improvement Level
Performance	Poor	Excellent	4.93x TPS increase
Reliability	Fair	Good	99.95% availability
Security	Adequate	Strong	0 critical flaws
Cost-Effectiveness	Average	Excellent	1,250% ROI

1. Performance tuning.

Analysis of optimization results showed that caching strategy has a significantly greater impact on overall system performance than hardware upgrades. Effective use of cache at the application and database levels significantly reduced latency and load on core resources. It was also found that database indexing requires constant review taking into account real query patterns, since changing the load profile without appropriate optimization can lead to performance degradation.

2. System design.

It was found that the degree of detail of the microservice architecture directly affects scalability and maintenance complexity. Excessive decomposition leads to

increased overhead for inter-service interaction, while insufficient decomposition limits scalability flexibility. The optimal balance provides increased fault tolerance and resource efficiency. In addition, early investment in observability tools (logging, metrics, tracing) significantly reduces the cost of incident diagnostics and speeds up the debugging process.

3. Team collaboration.

The introduction of cross-functional testing teams positively affected the quality of defect detection by combining expertise in development, security and operations. Automated testing (unit, integration, regression) reduced the impact of the human factor and increased the repeatability of results, which, in turn, contributed to the stability of the release cycle.

Overall, the results confirm that a comprehensive approach to optimizing performance, ensuring reliability and implementing DevOps engineering practices forms a stable, scalable and secure information system suitable for operation under high load conditions.

General Data Protection Regulation (GDPR) compliance check. The study conducted a comprehensive check of the system's compliance with the GDPR requirements, focusing on the implementation of data subjects' rights and the effectiveness of internal data management procedures.

Implementation of the Right to Erasure. During testing, it was found that requests for deletion of personal data were processed within 24 hours, with a target limit of less than 72 hours. Deletion was carried out from the main database, backups (postponed taking into account the retention policy) and related services, which confirms the correctness of the cascade deactivation and data cleaning mechanisms.

Right to Data Portability. The personal data export functionality was implemented in a structured machine-readable JSON format. The average time for generating and providing a file with personal data was 15 minutes, which meets the requirements for the timeliness of data subject requests and does not create an operational load on the system.

Consent Management. The system provides full logging of changes in the status

of users' consent to the processing of personal data. During testing, the accuracy of tracking changes in consent was 100%, which confirms the correctness of the logging, auditing, and versioning mechanisms for consent records.

Compliance audit results. Within 20,000 test scenarios covering various types of personal data processing (registration, authentication, payment processing, account deletion), no violations of GDPR requirements were detected. This indicates the systematic integration of the principles of privacy by design and privacy by default into the solution architecture.

Testing payment processes for compliance with PCI DSS requirements. A separate block of the study was devoted to checking the compliance of the payment infrastructure with the requirements of the PCI DSS standard.

Payment card data protection. Personal data of payment card holders was encrypted both during transmission (using the TLS protocol) and during storage (using cryptographic algorithms that meet modern requirements). Access to confidential fields of payment information is limited in accordance with the principle of least privilege, using a role-based access model and multi-factor authentication for administrative personnel.

Test results. The system successfully passed the test for all 12 PCI DSS control objectives, including requirements for network segmentation, security event logging, regular penetration testing and vulnerability management. The assessment results show compliance with PCI DSS Level 1, confirming the system's ability to process significant transaction volumes while ensuring an appropriate level of payment data protection.

User Acceptance Testing (UAT) was conducted with the involvement of representatives of the target audience and business stakeholders (Table 13). The main focus was on the correctness of the business logic, the convenience of the interface, and the compliance of the functionality with the stated requirements.

Table 13.

UAT Key Metrics

Metric	Target	Actual	Pass/Fail
Successful Transactions	99.5%	99.87%	Pass
User Satisfaction Score	4.0/5	4.3/5	Pass
Critical Bug Rate	<0.1%	0.03%	Pass

The UAT results confirmed:

– the correctness of the implementation of personal data and payment processing scenarios;

– the clarity of the mechanisms for granting and withdrawing consent;

– the absence of critical functional defects that prevent the use of the system.

Thus, the results of the GDPR and PCI DSS compliance checks, as well as the positive conclusions of the UAT, confirm the system's readiness for industrial operation in conditions of regulatory restrictions and increased requirements for the protection of personal and payment data.

As part of the final stage of the study, a qualitative and quantitative analysis of user feedback was conducted to assess the perception of the system’s performance and stability in real-world conditions. Feedback was collected after users interacted with the platform’s key functional modules (search, product browsing, wish list, checkout).

Positive aspects. Analysis of text responses showed a high level of satisfaction with the speed and stability of the system. In particular, users noted a significant improvement in order processing time during peak hours, which correlates with previously obtained performance indicators (reduction in latency and optimization of transaction processing). The absence of crashes during a continuous usage session lasting about 20 minutes was also emphasized, which indicates the stability of client and server interaction.

Quantitative content analysis showed that the most frequent positive characteristics were the words “Fast” (35 mentions) and “Smooth” (29 mentions). Next in frequency are “Reliable” (28) and “No failures” (22). Such a lexical dominant confirms that users primarily evaluate the system through the prism of speed and

continuity of work.

Areas for improvement. Despite the generally positive assessment, individual functional components were identified that require additional optimization. In particular, users noted a relatively slow loading of the wish list on mobile devices, which may be due to the features of the adaptive interface or inefficient processing of requests in the mobile context. There were also comments about the delay in displaying search results by about one second longer than expected.

In the word frequency analysis, the terms “Slow” (15 mentions) and “Improvement” (18 mentions) reflect key areas of potential optimization. Although their frequency is significantly lower compared to positive markers, they indicate the need for further improvement of individual interaction scenarios.

Summary of results. The results obtained demonstrate a mostly positive perception of the system by users, with an emphasis on speed, smooth operation, and the absence of failures. At the same time, the identified observations allow us to identify specific areas of technical optimization, in particular regarding the mobile experience and the search subsystem. Thus, the feedback analysis confirms the achievement of the service quality target indicators and forms an empirical basis for further iterative improvement of the system.

To improve the efficiency, scalability and maintainability of the information system, a phased refactoring plan was developed, which involves modernizing the technology stack and optimizing critical software components (Fig. 8).

Figure 8. Technical Debt Repayment Plan [made by the authors]

Phase 1: Modernizing the core stack (duration 3 months). The first phase focuses on updating the core software environment and key integration components. The

transition to Spring Boot 3.2 is planned, which allows for the use of modern configuration management functions, increased performance and improved support for new Java versions. In parallel, database drivers are planned to be updated to ensure compatibility with the latest versions of the DBMS and improve query performance.

Phase 2: Optimization of critical services (duration 6 months). The second phase involves targeted optimization of the most resource-intensive components of the system. It is planned to rewrite about 20% of the services that were identified as the most inefficient according to the results of performance profiling. In addition, a specialized framework is planned to be implemented to optimize database queries, which will reduce transaction processing time and reduce the load on servers during peak usage scenarios.

The implementation of this plan should ensure:

– increasing system performance by optimizing critical services and a modern execution environment;

– improving code maintainability by modernizing the technology stack;

– reducing the risk of performance degradation when scaling and increasing the load;

– creating a base for further implementation of innovative functions and automated monitoring and optimization mechanisms.

The study conducted a comprehensive assessment of the system's key performance indicators on a five-point scale of categories: "Poor", "Fair", "Good" and "Excellent". The assessment covered five key aspects: performance, reliability, security, scalability and cost-effectiveness (Table 14).

Table 14.

Final Evaluation Matrix

Evaluation dimension	Poor	Fair	Good	Excellent
Performance	☐	☐	☐	☒
Reliability	☐	☐	☒	☐
Security	☐	☐	☒	☐
Scalability	☐	☐	☒	☐
Cost-Effectiveness	☐	☐	☐	☒

System performance received a rating of “Excellent”, reflecting high throughput and low latency in processing transactions even under peak load. The system demonstrates high throughput, reaching 7200 transactions per second (TPS) with 50,000 concurrent users. Response times for 95% of requests are kept below 500 ms, confirming the effectiveness of query processing optimization and load balancing. The linear scalability of the system is estimated at 80% when adding additional nodes, indicating an effective parallel query processing architecture and system readiness for scaling.

Reliability was rated as “Good”, confirming stable operation of the system and the ability to quickly recover from minor failures, but requires further monitoring to achieve maximum reliability. The platform provides 99.95% availability thanks to mechanisms for automatic switching to standby nodes. Disaster recovery indicators meet the specified goals: RTO is 5 minutes, RPO is 10 seconds. The system demonstrates fault tolerance in case of failures of individual nodes, databases and network segments, which guarantees continuity of service to users.

Security received a rating of “Good”, demonstrating the effectiveness of implemented access control mechanisms, encryption and compliance with industry standards, with potential areas for optimization in monitoring and auditing processes. The platform has been tested for compliance with PCI DSS and GDPR. Testing has shown resistance to DDoS attacks at a speed of 50 Gbps with minimal impact on the operation of services. No critical vulnerabilities were detected during penetration testing, confirming a high level of security and protection of user data.

Scalability is rated as “Good”, which indicates sufficient flexibility of the system to increase the load and the ability to scale individual components with minimal changes to the architecture.

Cost-effectiveness reached the “Excellent” category, which confirms the optimal ratio of costs for system support and development to the achieved performance and reliability results. The cost-effectiveness of system optimization is high: the ROI reached 1250%. Improved platform performance and stability led to a 32% increase in conversion rate, which provided an additional income of 185 thousand USD per month.

At peak load, resources are used optimally: the processor is loaded at 68%, RAM — at 62%, which indicates effective management of hardware resources.

The provided code snippets and platform configuration examples allow you to recreate the experimental environment and scale it according to your business needs. They serve as practical guidelines for deploying high-performance, reliable and secure e-commerce systems, with the possibility of further improvement and integration of additional modules.

E-commerce promotion platform code snippets

Index Optimization Script

```
-- Create composite index for high-frequency query
CREATE INDEX idx_order_user_time ON orders (user_id,
order_time DESC);

-- Analyze query performance
EXPLAIN ANALYZE SELECT * FROM products
WHERE category_id = 123 AND price < 500 ORDER BY rating
DESC LIMIT 20;

-- Reorganize table to improve index efficiency
ALTER TABLE orders REORGANIZE PARTITION;

-- Monitor index usage
SELECT
    t.relname AS table_name,
    i.relname AS index_name,
    x.indx_scan AS scan_count
FROM
    pg_class t
JOIN
    pg_class i ON i.relparent = t.oid
JOIN
    pg_stat_all_indexes x ON x.indexrelid = i.oid
WHERE
    t.relkind = 'r'
ORDER BY
    x.indx_scan DESC;
```

Database Connection Pool Configuration (Spring Boot)

```
spring:
  datasource:
    url:
jdbc:mysql://localhost:3306/ecommerce?useSSL=false
    username: db_user
    password: db_password
    driver-class-name: com.mysql.cj.jdbc.Driver
    hikari:
      minimum-idle: 5
      maximum-pool-size: 100
      auto-commit: true
      idle-timeout: 30000
      connection-timeout: 30000
      max-lifetime: 600000
      connection-test-query: SELECT 1
      validation-timeout: 5000
      thread-factory:
com.ecommerce.util.CustomThreadFactory
```

Redis Cache Configuration (Java)

```
@Configuration
@EnableCaching
public class RedisConfig {

    @Bean
    public JedisConnectionFactory connectionFactory() {
        RedisStandaloneConfiguration config = new
RedisStandaloneConfiguration();
        config.setHostName("redis-master");
        config.setPort(6379);
        config.setPassword("redis-password");
        return new JedisConnectionFactory(config);
    }

    @Bean
    public RedisTemplate<String, Object> redisTemplate()
{
```

```
        RedisTemplate<String, Object> template = new
RedisTemplate<>();

template.setConnectionFactory(connectionFactory());

        // Serialization configuration
        Jackson2JsonRedisSerializer<Object> serializer =
            new
Jackson2JsonRedisSerializer<>(Object.class);
        ObjectMapper mapper = new ObjectMapper();
        mapper.setVisibility(PropertyAccessor.ALL,
JsonAutoDetect.Visibility.ANY);
mapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NO
N_FINAL);
        serializer.setObjectMapper(mapper);

        template.setValueSerializer(serializer);
        template.setKeySerializer(new
StringRedisSerializer());
        template.afterPropertiesSet();
        return template;
    }

    @Bean
    public                                    CacheManager
cacheManager(RedisConnectionFactory connectionFactory) {
        RedisCacheConfiguration         config         =
RedisCacheConfiguration.defaultCacheConfig()
            .entryTtl(Duration.ofMinutes(5))
            .disableCachingNullValues()

.serializeKeysWith(RedisSerializationContext.Serializati
onPair.fromSerializer(new StringRedisSerializer()))

.serializeValuesWith(RedisSerializationContext.Serializa
tionPair.fromSerializer(new
GenericJackson2JsonRedisSerializer()));
```

```
        return
RedisCacheManager.builder(connectionFactory)
            .cacheDefaults(config)
            .build();
    }
}
```

Cache Invalidation Example

```
@Service
public class ProductService {

    @Autowired
    private        RedisTemplate<String,        Product>
redisTemplate;

    @Autowired
    private ProductRepository productRepository;

    private  static  final  String  PRODUCT_CACHE_KEY  =
"product::%s";

    public Product getProductById(String productId) {
        String                cacheKey                =
String.format(PRODUCT_CACHE_KEY, productId);

        // Try to get from cache
        Product                product                =
redisTemplate.opsForValue().get(cacheKey);

        if (product == null) {
            // Fetch from database if not in cache
            product                                    =
productRepository.findById(productId)
                .orElseThrow(()          ->          new
ProductNotFoundException(productId));

            // Update cache
            redisTemplate.opsForValue().set(cacheKey,
product, 10, TimeUnit.MINUTES);
```

```
        }

        return product;
    }

    @Transactional
    public Product updateProduct(Product product) {
        // Update database
        Product updatedProduct =
productRepository.save(product);

        // Invalidate cache
        String cacheKey =
String.format(PRODUCT_CACHE_KEY, product.getId());
        redisTemplate.delete(cacheKey);

        return updatedProduct;
    }
}
```

Nginx Load Balancing Configuration.

```
# High-concurrency load balancing configuration
upstream ecom_backend {
    server backend1.example.com:8080;
    server backend2.example.com:8080;
    server backend3.example.com:8080;
    server backend4.example.com:8080;

    # Load balancing algorithm: least connections
    least_conn;

    # Health check
    health_check interval=30s fail_timeout=10s;

    # Session persistence based on user ID
    sticky cookie user_id expires=1h domain=.example.com
path=/;
}
```

```
server {
    listen 80;
    server_name ecom.example.com;

    # SSL configuration
    listen 443 ssl;
    ssl_certificate /path/to/cert.pem;
    ssl_certificate_key /path/to/key.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    # Request buffering
    client_body_buffer_size 128k;
    client_header_buffer_size 128k;
    large_client_header_buffers 4 128k;

    # Timeouts
    keepalive_timeout 65;
    send_timeout 30;
    proxy_timeout 600;

    location / {
        # Proxy requests to backend
        proxy_pass http://ecom_backend;

        # Preserve client IP
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header                X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header Host $host;

        # WebSocket support
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }

    # Static resources served directly
```

```
    location /static/ {
        root /var/www/ecom/static;
        expires 30d;
        add_header Cache-Control "public";
    }
}
```

Grafana Dashboard JSON Snippet.

```
{
  "id": 1,
  "title": "High Concurrency E-Commerce Performance",
  "panels": [
    {
      "type": "graph",
      "title": "Throughput (TPS) Over Time",
      "targets": [
        {
          "expr": "sum(ecom_tps) by (instance)",
          "legendFormat": "{{instance}}",
          "refId": "A"
        }
      ],
      "yaxes": [
        {
          "format": "short",
          "label": "TPS"
        },
        {
          "format": "short",
          "label": ""
        }
      ]
    },
    {
      "type": "graph",
      "title": "Response Time (95th Percentile)",
      "targets": [
        {
```

```
          "expr":              "histogram_quantile(0.95,
sum(rate(ecom_response_time_seconds_bucket[5m])) by (le,
instance))",
          "legendFormat": "{{instance}}: {{le}}s",
          "refId": "A"
        }
      ],
      "yaxes": [
        {
          "format": "seconds",
          "label": "Response Time"
        },
        {
          "format": "short",
          "label": ""
        }
      ]
    },
    {
      "type": "row",
      "panels": [
        {
          "type": "singlestat",
          "title": "Current CPU Utilization",
          "targets": [
            {
              "expr":               "100               -
(avg(rate(node_cpu_seconds_total{mode=\"idle\"}[5m]))  *
100",
              "refId": "A"
            }
          ],
          "format": "percent",
          "thresholds": "0,70,90"
        },
        {
          "type": "singlestat",
          "title": "Memory Usage",
```

```
          "targets": [
            {
              "expr":    "node_memory_MemUsed_bytes    /
node_memory_MemTotal_bytes * 100",
              "refId": "A"
            }
          ],
          "format": "percent",
          "thresholds": "0,80,95"
        },
        {
          "type": "singlestat",
          "title": "Database Connections",
          "targets": [
            {
              "expr":     "sum(mysql_connections)     by
(instance)",
              "refId": "A"
            }
          ],
          "format": "none"
        }
      ]
    }
  ],
  "time": {
    "from": "now-1h",
    "to": "now"
  }
}
```

Conclusion

In the e - commerce promotion scenario, the triple challenges of traffic peaks (50 - 100 times normal), business real - time performance (response delay < 200ms), and strong data consistency (such as inventory deduction) reveal the technical bottlenecks of traditional architectures in the traffic entry layer (low static resource loading efficiency, lagging load balancing strategies), business logic layer (intensive promotion

rule calculations, distributed lock competition), and data storage layer (cache penetration, cross - database transaction inconsistency). Existing research lacks a dynamic mapping mechanism of "business - architecture", fault tolerance in extreme scenarios, and cross - layer optimization collaboration, resulting in lagging system resource scheduling, which has become the core contradiction restricting the performance of high - concurrency systems.

The "dynamic perception - intelligent scheduling" technical system formed by the research results can be migrated to similar high - concurrency scenarios such as financial transactions (e.g., high - frequency trading systems) and real - time communication (e.g., peak traffic processing in IM). Its cross - layer optimization idea provides a universal reference for the design of complex distributed systems. In the future, it is necessary to establish a mathematical mapping relationship of "traffic characteristics – business models – architectural parameters", quantify the architectural configuration rules in different business scenarios through statistical and machine learning methods, form a standardized design paradigm for high - concurrency systems, promote the transformation of e - commerce technology from experience - driven to theory - guided, and provide underlying theoretical support for industry technological evolution.

DOI: 10.46299/979-8-90383-421-1.3

3. A method of test case formation for C++ libraries based on a Q-learning agent

Introduction. In modern computer systems (CS), C++ libraries are used as core components of software for data processing, computer graphics, file handling, and the implementation of other key software functions. During software development, test cases for C++ libraries are created incrementally as new Application Programming Interface (API) functions are added and software defects are corrected. The C++ libraries in use have diverse Application Programming Interfaces (APIs) that contain dozens or hundreds of functions which can be invoked in various combinations. Therefore, over a long period of software development, large test suites (TSs) accumulate, in which a significant portion of test cases (TCs) is redundant, since it duplicates the behavior. Such accumulation leads to an increase in test suite execution time and complicates the timely detection of software defects in computer systems.

In addition, many C++ libraries do not contain formalized descriptions of API function usage (specifications), namely: which function parameters are valid, which order of function calls is correct, which return values are considered expected, and which indicate the presence of erroneous behavior of the C++ library. In the absence of such specifications, automated test case (TC) generation is performed using search-based, heuristic, and reinforcement learning approaches that explore the space of possible sequences of function calls without prior knowledge of the API usage rules.

Existing approaches to TC optimization, such as delta debugging algorithms, greedy algorithms, and Monte Carlo Tree Search, make it possible to reduce the size of test suite (TS), but they may be insufficiently effective when the number of possible sequences of function calls is very large.

A relevant research direction in the field of computer systems is the study of TC formation (optimization) for C++ libraries based on a Q-learning agent. The advantages of such an approach may include the following:

– the reward function of the Q-learning agent can be configured taking into account specified characteristics of C++ library testing effectiveness, in particular branch coverage growth, TC length, and others;

– the Q-learning agent is capable of accumulating knowledge about the utility of individual TC actions based on test execution results and applying this knowledge for decision-making;

– unlike greedy and heuristic methods of TC optimization (formation), the Q-learning agent is able to consider the future utility of TC actions;

– the Q-learning agent does not require prior construction of a complete behavioral model of the C++ library under test.

Thus, the development of a method of TC formation for C++ libraries based on a Q-learning agent is a relevant scientific problem.

Main part. There exists a wide range of methods and approaches to software development for information systems (IS) and CS that automate the processes of its creation (development and build) and testing [69]. The main approaches to software development include the following:

1) Continuous Integration (CI) – ensures adherence to a process in which developers merge their code changes into a shared project repository. Such integration is automatically built and tested in order to quickly detect any errors that may affect the overall operability of the software;

2) Continuous Delivery/Deployment (CD) – ensures a process in which the codebase is always maintained in a release-ready state, and the processes of testing and building are automated.

The essence of the CI/CD approaches is as follows.

– changes in the software code – the developer introduces changes to the code and commits them to the repository;

– software build – the system automatically builds the code, transforming it into a software product;

– software testing – the built code undergoes a series of automated tests to verify correctness and functionality;

– software delivery/deployment – provided that all tests are successful, the code can be automatically deployed to a server or to a user environment.

The main advantages of the CI/CD approaches include the following:

– acceleration of software release – rapid and frequent deployment of new features and fixes;

– higher software code quality – rapid detection and elimination of errors at early stages;

– reduction of software operational risks – consistent and incremental changes are easier to control and correct;

– increased development efficiency – frees developers from routine tasks, allowing them to focus on software development.

The necessity for frequent software code changes to satisfy user requirements makes software testing a bottleneck during development. That is, the number of regression defects increases, and the costs of automated testing procedures grow nonlinearly with the size of the software codebase.

When modifying the program code of C++ libraries, software developers are interested in quickly obtaining information about the behavior of C++ libraries across a wide range of tested functionality and require high-quality TSs in order to avoid the risks of undetected software defects in the case of executing a smaller number of TCs. In addition, after such changes, the speed of early error detection is critical, considering that a full regression testing cycle may, in some cases, take weeks.

Ensuring software quality and compliance with standards (requirements) is achieved through validation, verification, and testing. At the same time, automated software testing is part of quality control within the software development life cycle. Modern information technologies (IT) and software tools play a key role in the development and execution of automated tests, reducing the time required for software testing software and simplifying the testing process.

Thus, the automation of software testing is a fundamental stage in ensuring its high quality and efficiency. The use of modern information technologies (IT) and artificial intelligence (AI) enables the automation of the TC creation process, which allows increasing the efficiency and accuracy of testing. Such an approach requires the improvement of existing and the development of new algorithms for automated TC

formation, as well as the development of mathematical models to meet testing requirements.

Among the main disadvantages of automated testing of C++ libraries, the following can be identified:

– open-source C++ libraries often lack documentation on correct API usage, which complicates the automation of C++ library testing;

– growth of the size of C++ library test suites during software development – the time required for their execution becomes unacceptable;

– instability of TS execution – exceptions during program code execution or its side effects may affect the reproducibility of TCs;

– a large space of possible combinations of API call sequences within TCs.

The highlighted disadvantages of automated testing of C++ libraries within software of information systems and computer systems can be addressed through TC optimization.

TC optimization is the process of reducing the execution time of TCs while preserving (or increasing) a specified testing effectiveness criterion. The set of TC optimization tasks includes the following: TC parameter optimization, TC formation, TC minimization, and TC prioritization [70].

TC parameter optimization is the configuration of input data for a predefined TC to improve a specified testing effectiveness criterion [71].

TC formation is the process of constructing TCs as valid sequences of API calls of the C++ library under test, while preserving a specified testing effectiveness criterion.

TC minimization is the process of removing redundant actions from TCs while preserving a specified testing effectiveness criterion.

TC prioritization is the process of ordering actions within TCs in such a way that the most informative TC actions are executed first, ensuring rapid detection of software defects or increasing branch coverage of the C++ library under test.

In Fig. 1 is presented a diagram of the main TC optimization tasks and the approaches and methods for their solution [72, 73].

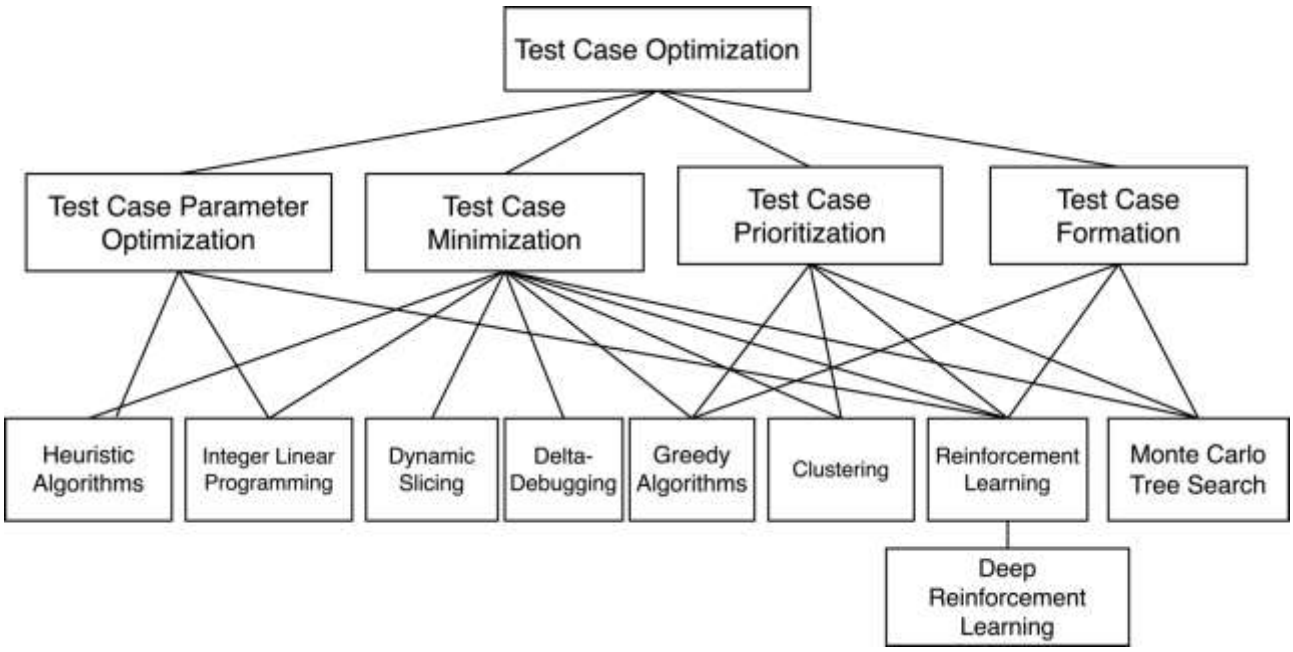


Figure 1. Diagram of main test case optimization tasks and the solution approaches

In TC optimization tasks, the number of possible subsequences of TC actions grows exponentially. TC optimization problems with preservation of branch coverage of the C++ library under test can be reduced to a constrained set-cover problem, which is NP-hard (set cover) [74].

The testing effectiveness criterion in TC optimization tasks is the preservation (or increase) of branch coverage of the C++ library under test relative to the original TS of the library, or the preservation of the ability of the optimized TS to reproduce software defects in the same manner as the original TS.

Approaches to solving the TC optimization problem encompass a wide range of methods aimed at reducing the size of TCs without loss of testing effectiveness characteristics.

In addition, TC optimization methods can be classified according to the granularity level of optimization. The coarse-grained level of TC optimization is a level at which entire and indivisible TCs are optimized within a TS without analyzing their internal structure, treating a TC as an atomic unit. The fine-grained level of TC optimization operates at a lower level of abstraction, namely at the level of individual program code instructions (e.g., statements or API calls) of the C++ library under test.

The use of the fine-grained level of optimization allows increasing optimization accuracy, since it evaluates the utility of specific TC actions or sequences of calls rather than the entire TC as a whole. The application of the fine-grained level of optimization is advisable when the TS contains many TCs that are long or include a large number of redundant TC actions. Fine-grained TC minimization methods achieve high minimization accuracy. At the same time, the effectiveness of such methods in large software projects is limited by the high cost of analysis at the level of individual TC instructions, which creates a fundamental trade-off between minimization accuracy and TC minimization time.

TC optimization methods should be evaluated in terms of adequacy, that is, their ability to preserve the desired testing effectiveness characteristics (for example, code coverage, software defect detection potential, and others). An adequate TC optimization method guarantees that the reduced TS retains the testing effectiveness characteristics of the original TS of the C++ library under test. On the other hand, an inadequate TC optimization method does not guarantee the preservation of the testing effectiveness characteristics of the original TS of the C++ library under test. The application of inadequate TC optimization methods also carries the risk of losing the ability to detect defects in the behavior of the C++ library.

In known studies, in particular [74], the necessity of using standardized criteria for the objective evaluation of the effectiveness of TC optimization methods is emphasized. The following testing effectiveness criteria for C++ libraries are proposed:

– coverage retention capability, which determines the extent to which the reduced TS reproduces the code coverage indicators of the original TS;

– fault detection capability, measured by the average percentage of detected software defects;

– the level of TS redundancy, which reflects the proportion of unique actions within TCs;

– reduction of TS execution time (e.g., execution cost) relative to the original TS of the C++ library under test;

– evaluation of TC optimization method effectiveness as the number of TCs increases, as well as the time costs of the TC optimization process;

– stability of TC optimization method effectiveness across different software versions and reproducibility of results.

Given that the object of testing is C++ software libraries, access to their functionality is performed through the use of the API. During TS execution, information about the behavior of the library is collected and used for the subsequent formalized description of the TC formation problem. In particular, attention is focused on the following characteristics:

– correctness of TC execution (e.g., successfully and unsuccessfully completed TCs);

– correctness of step-by-step execution of the program code of the C++ library under test (e.g., catching the execution errors in the form of program exceptions, return statuses of invoked functions, and others);

– structural code coverage metrics of the source code of the C++ library under test (e.g., branch coverage, function coverage, basic block coverage, and others);

– TS execution time before and after optimization;

– time costs required to perform TC optimization.

Formalization of the main mathematical notations used.

The following assumption is introduced. The finite set of API calls of the C++ library under test is defined as:

$$\mathcal{A} = \{a_1, \dots, a_n\},\ n \in \mathbb{N}. \quad (1)$$

A TC step is considered to be an instruction $a_i(\bar{x})$, where the admissible parameter vector is defined as:

$$\bar{x} = (x_1, x_2, \dots, x_m),\ \ m \in \mathbb{N}. \quad (2)$$

A TC consists of an ordered set of steps (actions), which can be represented as follows:

$$TC = \langle a_1, a_2, \dots, a_l \rangle, \quad a_i \in \mathcal{A}. \tag{3}$$

Thus, a TC has length $l \in \mathbb{N}$, which corresponds to the number of steps (actions) of the TC in set (1) and is determined by the function $l = L(TC)$, where $l < L_{max}$.

The admissibility of transitions between instructions is determined based on a compatibility matrix and a predicate that models the syntactic and semantic constraints of the TC as follows:

$$\begin{cases} \mathcal{V} \subseteq \mathcal{A} \times \mathcal{A} \\ Valid(TC, \bar{x}) \in \{0,1\} \end{cases}. \tag{4}$$

Execution of a TC (TC) with a given input parameter vector (2) in the test environment allows computing the source code coverage metric of the C++ library under test and is defined by the functional operator:

$$C_{TC} = Coverage(TC). \tag{5}$$

The functional code coverage operator (5) returns the coverage value of the C++ library under test as a percentage from 0 to 100. It determines the degree of exploration of the program code of the C++ library under test, considering structural metrics such as line coverage, branch coverage, or function coverage.

The increase in source code coverage of the C++ library under test at two consecutive TC execution steps can be defined by the following equation:

$$\Delta C_{n_k} = C(TC^{n_k}) - C(TC^{n_k-1}), \tag{6}$$

where n_k is the TC step number.

The redundancy of steps without coverage increase within a TC can be defined as follows:

$$R_{ed}(TC) = \frac{1}{L(TC)} \sum_{n_k} \begin{cases} 0 & if\ \Delta C_{n_k} = 1 \\ 1 & if\ \Delta C_{n_k} = 0 \end{cases}. \quad (7)$$

The execution time of a TC can be defined by the following expression:

$$T(TC) = \sum_{n_k} cost(a_{n_k}). \quad (8)$$

Based on the presented formalizations, the problem of test case parameter optimization can be solved.

The following assumption is introduced. Let TC (TC_0) be a fixed sequence of valid API calls of the C++ library under test (3). It is required to determine the parameter vector $\bar{x}^*$ (2) for which code coverage (5) is maximized during TC execution under the following constraints:

$$\begin{cases} \max\limits_{\bar{x}} C(TC_0, \bar{x}) \\ Valid(TC_0, \bar{x}) = 1 \\ T(TC_0, x) \le T_{max} \end{cases}. \quad (9)$$

Test case minimization problem. A TC (TC_0) (3) is a sequence of API calls that satisfies (4) and has an original branch coverage characteristic C_{TC_0} (5). It is required to remove steps from TC_0 in such a way that the resulting output TC (TC^*):

– satisfies constraint (4), that is, remains valid;

– preserves the value of the target testing effectiveness characteristic C_{TC_0};

– has minimal length $L\ (TC)$ (number of instructions).

Such a formulation can be considered as a fine-grained TC minimization problem, demonstrating that pruning individual TC steps (actions) allows significantly reducing the size of the TC without loss of its effectiveness.

The TC minimization problem with coverage preservation can be represented as follows:

$$\begin{cases} \min\limits_{TC \subseteq TC_0} LC(TC) \\ Valid(TC) = 1 \cdot \\ C_{TC} \geq C_{TC_0} \end{cases} \quad (10)$$

Test case formation problem. Classical formulations of TC optimization problems are either aimed at increasing code coverage (e.g., TC parameter optimization task) or at minimizing TCs (e.g., elimination of redundant TC steps). However, for testing C++ software libraries, it is necessary to apply a combination of these two problems within a unified formulation that simultaneously:

– forms a new TC from the set of admissible API actions of the C++ library under test;

– ensures an increase (or preservation) of the code coverage of the resulting TS of the C++ library compared to the original TS of the C++ library under test;

– guarantees minimization of the length of the resulting TS and reduction of its execution time.

In this formulation, call parameters are not optimized – for each instruction $a_i \in TC$, a fixed parameter vector is predefined.

It is assumed that a TC (TC) (3) is given as a fixed sequence of valid API calls of the C++ library under test, as well as constraint (4). It is required to determine a sequence of TC actions or TC (TC^*) that maximizes the code coverage of the C++ library under test (5):

$$\begin{cases} \max\limits_{TC} \mathrm{C}(TC) \\ Valid(TC) = 1 \cdot \\ T(TC) \leq T_{max} \end{cases} \quad (11)$$

In this formulation, TC formation is a discrete optimization problem with constraints (4).

A TS can be defined as an ordered set of TCs:

$$TS = \langle TC_1, TC_2, \dots, TC_f \rangle. \quad (12)$$

Since a TS consists of TCs, each of which is composed of an ordered set of steps (e.g., actions) (3), the length of a TS can be defined as the sum of the lengths of all TCs included in the TS:

$$L(TS) = \sum_i L(TC_i), \ TC_i \subseteq TS. \quad (13)$$

The coverage of a TS is defined as a functional operator over the TS (TS):

$$C_{TS} = Coverage(TS). \quad (14)$$

The result of the TC formation (optimization) process is an optimized TS (TS^*), which consists of the formed (minimized) TCs. Thus, optimization is performed for each TC of the TS separately, while the comparative evaluation of the effectiveness of formation (optimization) methods is carried out at the level of the TS of the C++ library under test.

TC minimization for C++ libraries based on delta-debugging algorithms.

The delta-debugging algorithms essence consists in the gradual removal of parts of a TC and verification of whether the desired property, such as error reproduction or code coverage, is preserved. If the reduced TC remains valid, the removed steps (e.g., actions) are considered redundant and are discarded.

The delta debugging algorithm aims to identify TC actions that influence the achievement of the target branch coverage of the C++ library under test or the reproduction of a software execution error.

Thus, the delta debugging algorithm for TC minimization allows removing steps that do not change the effectiveness of the TC while preserving its semantic validity. Unlike greedy TC formation algorithms, delta debugging deterministically provides a minimal subset of actions of the original TC that satisfies the target branch coverage of the C++ library under test.

The delta debugging algorithm solves the TC minimization problem (10) by removing redundant actions that do not contribute to achieving the desired value of the objective function (5). Its main parameter is the granularity of TC partitioning ($g \in \mathbb{N}$). Granularity is the number of parts into which the current TC is divided during a given iteration of the delta debugging algorithm. This parameter can adapt its value during the execution of the algorithm. It determines the level of detail in the search for redundant TC actions: a smaller value of g means that the TC will be divided into parts containing a larger number of actions, whereas a larger value corresponds to parts with fewer actions, which allows more precise localization of TC actions for redundancy checking according to the specified criterion.

The essence of the classical delta debugging algorithm is as follows. At the beginning of the algorithm, the granularity parameter g is equal to 2, that is, the TC is divided into two parts representing action blocks of the TC partition. If, after the temporary removal of a partition block of actions from the TC, the coverage objective function (5) returns a branch coverage value of the C++ library under test that is not less than that of the initial TC, the removed block of actions is considered redundant and is permanently deleted from the TC. If branch coverage deteriorates, the TC is divided into smaller parts (for example, into four blocks), and the verification cycle is repeated. Therefore, the granularity parameter of TC partitioning changes dynamically as follows:

– in the case of successful fragment removal (e.g., code coverage does not deteriorate), the granularity of the TC decreases:

$$g \leftarrow \max(g/2, 2); \tag{15}$$

– in the case of unsuccessful removal, the granularity increases:

$$g \leftarrow min(2 \cdot g, L(TC^*)). \quad (16)$$

Thus, the TC can be partitioned into contiguous blocks of actions that do not contain identical TC actions. For this purpose, a TC partitioning procedure is used, which returns a set of contiguous blocks of TC actions without identical actions.

At the same time, the classical delta debugging algorithm has the following disadvantages:

– a high number of checks – at each partition level, it is necessary to test all partition blocks of TC actions and their complements, which leads to a significant number of evaluations of the coverage function (5) and makes the algorithm slow for large TCs;

– inefficiency for long TCs – in the presence of hundreds or thousands of TC actions, the algorithm may require many iterations before achieving reduction, especially if redundant actions are sparsely distributed;

– non-uniform granularity – division into a fixed number of partition blocks with subsequent increase or decrease of granularity does not always correspond to the structural characteristics of the TC of the C++ library under test.

To overcome the disadvantages of the classical delta debugging algorithm, an improved delta-debugging algorithm is proposed. Instead of partitioning the TC into contiguous blocks of actions, it forms groups of TC actions for redundancy checking based on the criterion of non-uniqueness of TC actions.

The essence of the approach is that not all TC actions contribute equally to branch coverage. Some of them activate unique basic blocks of program execution, that is, program regions that are not covered by any other TC action. The removal of such actions carries the risk of losing part of the branch coverage of the C++ library under test – therefore, they are marked as “protected.” The improved delta debugging algorithm performs redundancy checking only among unprotected TC actions, which

makes it possible to preserve full branch coverage while reducing TC length with a smaller number of calls to the branch coverage evaluation function (5).

For each action a_i of a TC, the set of unique basic blocks of program execution can be defined as follows:

$$U(a_i) = \{q \in B(a_i) \mid q \notin \bigcup_{j \neq i} B(a_j)\}, \quad (17)$$

where $B(a_i)$ denotes the set of program basic blocks covered by action a_i.

In case $U(a_i) \neq \emptyset$, the action a_i of the TC is marked as "protected" and is not included in the group of TC actions formed during the algorithm iteration for redundancy checking. Thus, the set of "protected" TC actions can be defined as follows:

$$\mathcal{A}_{prot} = \{a_i \mid U(a_i) \neq \emptyset\}. \quad (18)$$

At each iteration, the algorithm forms a group of TC actions for redundancy checking, namely a subset of TC actions excluding the "protected" elements. Groups of TC actions whose temporary removal from the TC does not reduce the branch coverage of the C++ library under test below the branch coverage level of the original TC are permanently removed from the TC.

Fig. 2 presents the flowchart of Procedure 1 for forming the set of "protected" TC actions based on their unique contribution to the coverage of program basic blocks defined by (17).

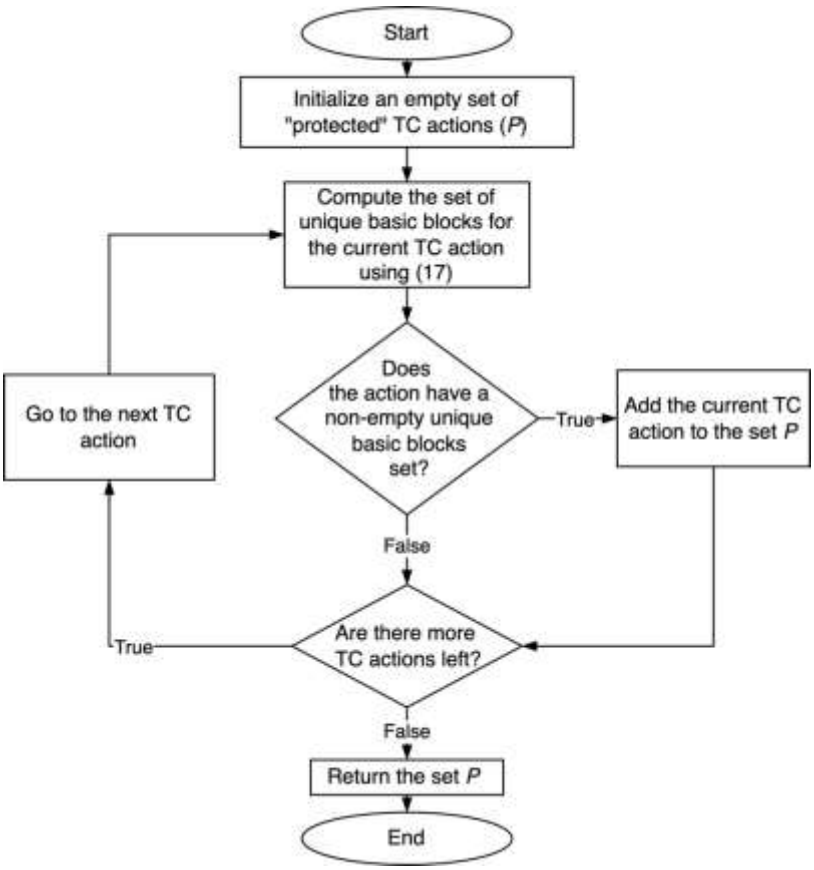


Figure 2. Flowchart of the Procedure 1 for forming the set of "protected" TC actions based on their unique contribution to the coverage of program basic blocks

The improved delta debugging Algorithm 1 uses the Procedure 1 and is controlled by the parameter $g' \in \mathbb{R}$, representing the granularity of TC partitioning, which determines the size of subsets into which the input sequence of TC actions is divided. At the initial iterations of the Algorithm 1, as in the classical version, a large value of this parameter allows performing coarser attempts to remove redundant TC actions, and in the case of unsuccessful removal, it is gradually increased, providing finer-grained partitioning of the TC.

The improved Algorithm 1 for delta debugging TC minimization is presented in Fig. 3.

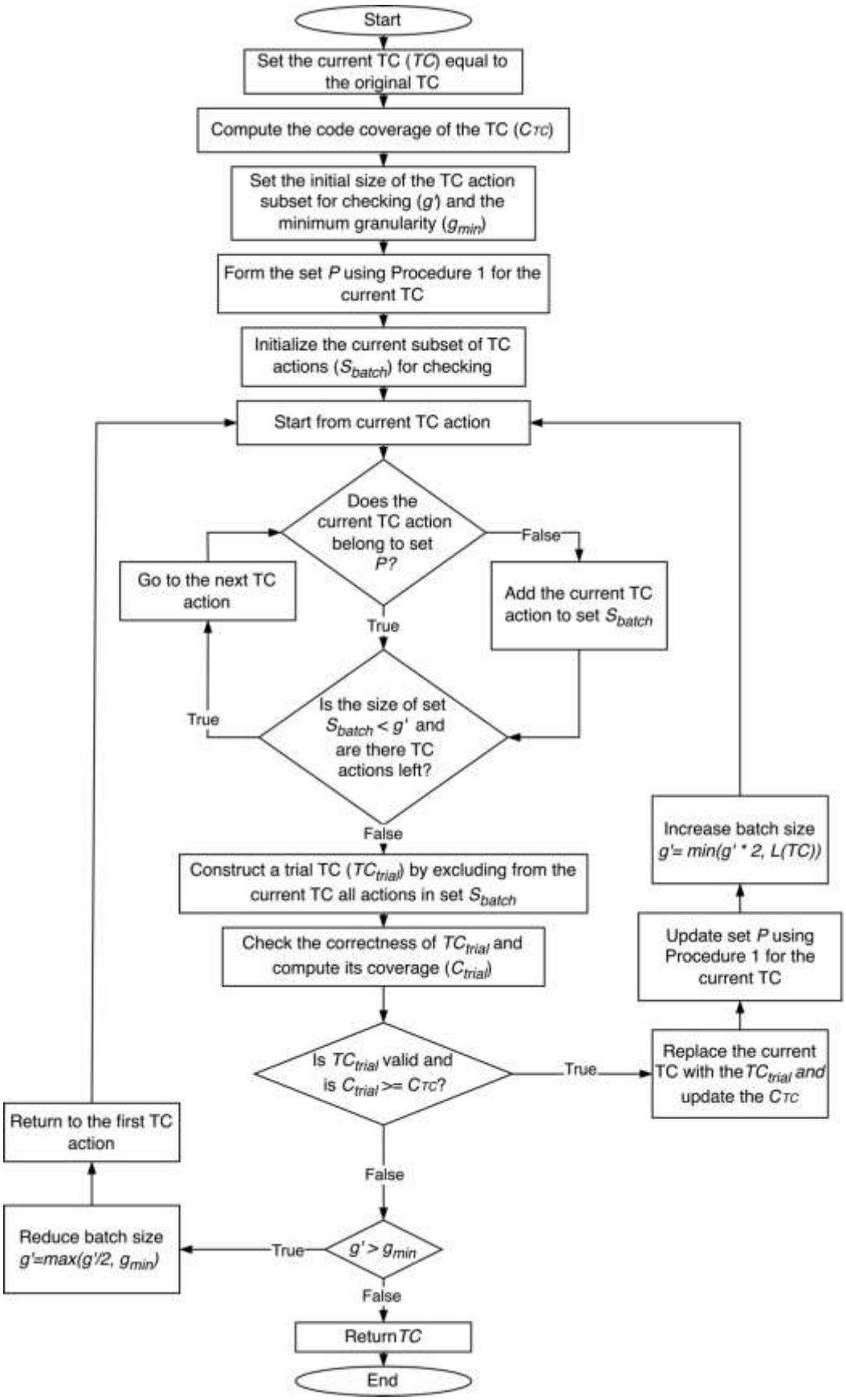


Figure 3. Flowchart of the improved Algorithm 1 for delta debugging TC minimization

Thus, the main properties of the improved delta debugging Algorithm 1 include the following:

– adaptivity – the granularity parameter controls the level of detail; at the initial stages of the algorithm, large groups of TC actions are removed, whereas at the final iterations, finer refinement of the TC is performed;

– localized checking – instead of fully re-evaluating the entire TC at each iteration, the algorithm verifies only the modified fragments of the TC, which reduces the number of calls to the coverage function (5);

– reproducibility – for identical input data, the algorithm always returns the same result;

– computational complexity – due to the combination of multiple removal and preliminary identification of unique basic blocks of program execution, the number of code coverage evaluations is reduced;

– robustness with respect to TC structure – the algorithm does not depend on the initial order of actions in the TC, since it checks groups of TC actions sequentially, considering only those that do not affect the activation of unique basic blocks of program execution.

TC formation for C++ libraries based on greedy algorithms.

Greedy algorithms are one of the classical approaches to TC formation or minimization. Their essence lies in the sequential selection of actions that, at each step, provide the maximum increase in code coverage until full coverage of the original TC is achieved.

In the classical formulation, code coverage can be computed according to different criteria, for example, by the number of newly satisfied test requirements or newly activated code elements.

At each TC formation step n_k, the immediate utility of each not yet used TC action is computed based on the increase in source code coverage of the C++ library under test (6). The TC action with the maximum immediate utility is selected as follows:

$$a' = arg \max_{a \in \mathcal{A}_{n_k}} \Delta C_{n_k}(a). \quad (19)$$

That is, the immediate utility is equal to the number of new branches activated by adding action a at the current step. In case $\Delta C_{n_k}(a') = 0$, the algorithm terminates since none of the actions can further increase branch coverage. Otherwise, the selected action is added to the TC, and the process continues until full coverage of the original TC of the C++ library under test is achieved or until termination according to another criterion (for example, a constraint on the TC length).

Along with the advantages of simplicity and practical efficiency, the classical greedy algorithm has the following disadvantages:

– local coverage growth plateaus – if none of the available actions provides a coverage increase at the current step, the algorithm terminates. At the same time, it may lose the opportunity to achieve higher coverage that becomes possible only after performing a certain “intermediate” action;

– determinism and lack of diversification – the algorithm always selects the single best action according to immediate coverage gain, which makes it prone to getting trapped in local maxima and limits the diversity of constructed TCs;

– ignoring additional information about TC actions – the classical greedy algorithm does not take into account any auxiliary signals about potentially useful TC actions. For example, it does not consider that some actions may activate new basic blocks of the program code of the C++ library under test. The future utility of such TC actions can be incorporated, which may lead to an increase in branch coverage of the C++ library under test.

Thus, the classical greedy algorithm has a significant limitation: at each step, it selects only the TC action with the maximum immediate increase in branch coverage. Therefore, if all available TC actions provide zero immediate gain, the algorithm terminates, even if some of them could enable subsequent actions that yield a positive increase in branch coverage.

To overcome this limitation of the classical algorithm, the mathematical model of the greedy algorithm has been improved to incorporate two parameters:

1) the immediate branch coverage gain ΔC_{n_k};

2) the future utility of a TC action f_{n_k}, which is evaluated based on the activation of new unique basic blocks of the program after executing the given TC action.

Thus, to guide TC formation, basic-block coverage is used as a heuristic. For each action a executed within the TC prefix $TC_{1:n_k}$, where n_k is the current TC formation step, the set of executed program basic blocks $BB(TC_{1:n_k})$ is determined.

If the last action activates a block that has never been executed before, the action activates a unique program basic block. Therefore, the uniqueness indicator can be defined as follows:

$$u_{n_k}(a) = \begin{cases} 1 & , \ if \ BB(TC_{1:n_k})/BB(TC_{1:n_k-1}) \neq \emptyset \\ 0 & , \ if \ BB(TC_{1:n_k})/BB(TC_{1:n_k-1}) = \emptyset \end{cases}. \quad (20)$$

The essence of the approach is that even if an action does not provide an immediate increase in branch coverage, it may act as a “key” after which other TC actions begin to yield branch coverage growth. At each step, the algorithm evaluates the weighted score of a TC action as follows:

$$F_{n_k}(a) = \Delta C_{n_k}(a) + w_f \cdot f_{n_k}(a), \quad (21)$$

where $w_f \in (0, 1)$ is the weight that determines the memory of the future utility of a TC action.

If the execution of $TC_{1:n_k} \oplus a$ produces new program basic blocks that have not appeared previously, the uniqueness indicator of the TC action (20) – $u_{n_k}(a) \in \{0, 1\}$ is used. It is assumed that after performing action a, a set of TC actions appears such that:

$$N_{n_k}(a) = \{b \in \mathcal{A} \mid u_{n_k}(b) = 1\}, \quad (22)$$

where $u_t(b) = 1$ means that action a acted as a "key" for activating branch coverage growth of the C++ library under test through the execution of TC action b from set N_{n_k}.

For actions that have acted as a "key," the future utility $f_{n_k}(a)$ is stored in a future utility table, where the key is the TC action and the future utility values are updated according to the following rule:

$$f_{n_k}(a) \leftarrow max(f_{n_k}(a), \max_{b \in N_{n_k}(a)} F_{n_k}(b). \quad (23)$$

Thus, updating the future utility $f_{n_k}(a)$ according to the maximum value principle is chosen because, in the TC formation problem, it is important that a TC action has at least once activated new unique branches of the C++ library under test. The use of the maximum value guarantees preservation of the maximum branch coverage increase (6) during TC formation and simplifies convergence of the algorithm without the need for additional parameters.

At each iteration of the algorithm, a candidate set $\mathcal{A}_{n_k} \in \mathcal{A}$ is formed, which includes all TC actions a_i for which $u_{n_k}(a_i) = 1$ or $F_{n_k}(a_i) > 0$. From the candidate set, selection of a TC action is performed among the first $v_c \in \mathbb{N}$ candidates after sorting the candidate set $\mathcal{A}_{n_k}$ in descending order of $F_{n_k}(a)$.

To diversify the algorithm, the softmax formula can be used to select the next TC action.

$$P_{n_k}(a) = \frac{e^{F_{n_k}(a)/\tau}}{\sum_{a^* \in \mathcal{A}_{n_k}} e^{F_{n_k}(a^*)/\tau}}, \quad (24)$$

where τ is a coefficient that controls the balance between greedy selection of a TC action and exploration of the future utility of TC actions.

Fig. 4 presents the flowchart of the improved Algorithm 2 for greedy TC formation.

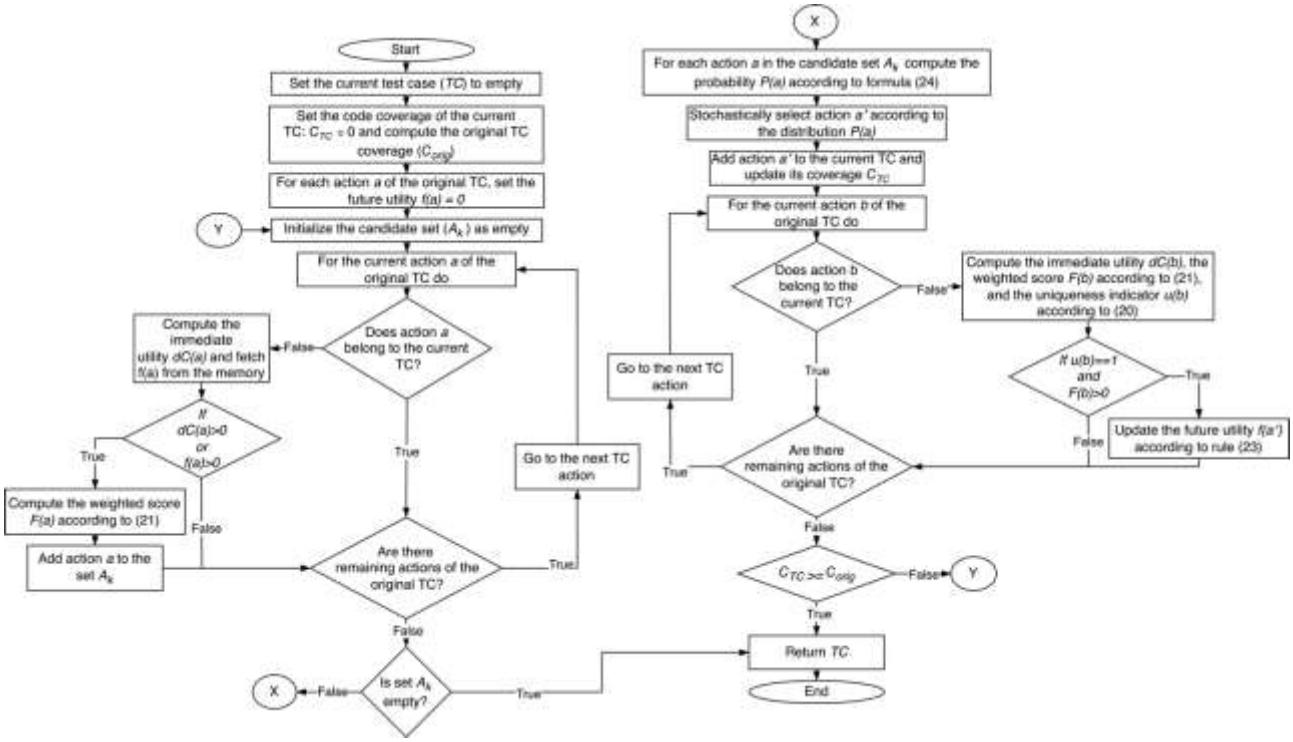


Figure 4. Flowchart of the improved Algorithm 2 for greedy TC formation

Thus, the improved mathematical model and greedy Algorithm 2 take into account the future utility of TC actions and can be applied to solving the TC formation problem for C++ libraries. By preserving information about TC actions that provide access to new unique basic blocks of the program code, the improved algorithm has a high potential for preserving branch coverage and produces shorter and more efficient TCs compared to the classical greedy TC formation algorithm, which considers only the immediate increase in branch coverage of the C++ library under test.

TC formation for C++ libraries based on a Q-learning agent.

The formalization of the behavior of a TC formation agent for C++ libraries can be represented as a process of incremental construction of a sequence of TC actions, which occurs through interaction with the dynamic execution environment of the C++ library under test. The interaction of the agent with the environment in which the

executable code of the C++ library is executed is carried out as follows. After each TC action executed in the execution environment of the C++ library, the TC formation agent receives a reward. The process of TC formation by the agent can be described in terms of a Markov Decision Process (MDP) [75]. An MDP can be defined as an ordered five-tuple of components as follows:

$$\langle \mathcal{S},\ \mathcal{A},\ \mathcal{P}(s'|s,a),\ \mathcal{R}(s,a),\ \gamma \rangle, \tag{25}$$

where: $\mathcal{S}$ is the set of possible states that describe the state of the TC formation agent at each step of interaction with the execution environment of the C++ library under test; $\mathcal{A}$ is the set of admissible TC actions available to the TC formation agent when it is in a given state; $\mathcal{P}(s'|s,a)$ is the state transition probability function of the TC formation agent, which determines the probability with which the agent transitions to a new state s' after executing TC action a in the current state s; $\mathcal{R}(s,a)$ is the reward function for executing TC action a, which maps each pair (agent state, TC action) to a numerical value reflecting the utility of action a executed in the current state s of the TC formation agent; $\gamma \in (0,1)$ is the discount factor that determines the weight of future rewards of the TC formation agent relative to immediate rewards: the smaller the value of γ, the higher the priority given to immediate rewards received by the agent during TC formation.

The classical Q-learning algorithm belongs to the class of reinforcement learning methods that allow the agent to iteratively improve its decision-making policy for selecting the next TC action, based on the experience acquired during interaction with the execution environment of the C++ library under test. The objective of the TC formation agent is to maximize the expected cumulative reward by determining an approximation of the optimal action-value function as follows:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{n_k=0}^{\infty} \gamma^{n_k} \cdot \mathcal{R}\left(s_{n_k}, a_{n_k}\right)\right], \tag{26}$$

where π is the policy for selecting the next TC action; n_k is the TC formation step.

Formula (26) accounts for the stochastic nature of transitions between agent states by maximizing the expected value of the discounted cumulative reward received by the agent during learning. The values in the Q-table (Q-values) are updated according to the following classical Bellman equation:

$$Q(s_{n_k}, a_{n_k}) \leftarrow Q(s_{n_k}, a_{n_k}) + \alpha \cdot \delta^{n_k}$$

$$\delta^{n_k} = \left[r_{n_k} + \gamma \cdot \max_{a_{n_k+1}} Q(s_{n_k+1}, a_{n_k+1}) - Q(s_{n_k}, a_{n_k}) \right], \quad (27)$$

where $\alpha \in (0,1]$ is the learning rate multiplied by δ^{n_k}, which represents the Temporal Difference Error (TDE).

The process of TC formation by the agent can be described as follows. At each TC formation step n_k, the TC formation agent is in some state $s_{n_k} \in S$, selects a TC action $a_{n_k} \in \mathcal{A}$, appends action a_{n_k} to the partially formed TC, executes it in the execution environment of the C++ library under test, and receives an immediate reward r_{n_k}. After that, the TC formation agent transitions to a new state s_{n_k+1}.

Since classical Q-learning belongs to the class of model-free methods, the TC formation agent does not use the state transition probability function $\mathcal{P}(s_{n_k+1}|s_{n_k}, a_{n_k})$. Instead, it updates its own estimate of the utility of TC actions based on the Q-function update rule. This update rule corresponds to the classical Temporal Difference (TD) model, according to which the TC formation agent gradually refines its estimates of the utility of actions by combining the received immediate reward with the expected future action-value estimates [76]. The action-value estimate for a “state, action” pair can be represented in the form of a discrete Q-table that stores the expected reward value obtained by the TC formation agent for each possible combination of agent state and TC action. This approach allows the agent to improve its policy for selecting TC actions during interaction with the API of the C++ library under test without relying on formal API specifications or an internal model of the testing environment.

Despite the fact that the classical Q-learning model enables gradual learning of an effective decision-making policy for TC formation, its application to the TC formation problem for C++ libraries has the following main disadvantages:

– state space explosion – the space of all possible API call sequences grows exponentially depending on the complexity of the C++ library under test. This leads to fragmentation of the Q-table: many agent states are rarely visited, which complicates generalization of the agent's experience;

– lack of generalization between similar agent states – classical Q-learning stores Q-values for each individual agent state and does not consider structural similarity between TCs that share common API call suffixes. Such an approach may reduce the stability and learning speed of the TC formation agent.

To construct a mathematical model of a TC formation agent for C++ libraries, it is necessary to define a set of key reinforcement learning components. The main elements of the architecture, including the TC formation agent, the agent's state space, the set of TC actions, the policy for selecting the next TC action, the reward function, and the learning algorithm, are presented in Table 1.

Table 1.

Definition and description of reinforcement learning components for constructing the mathematical model of the TC formation agent for C++ libraries

Component	Description
Agent and TC formation environment	The Q-learning agent interacts with the execution environment of the C++ library under test. The environment is defined as an interpreter that executes generated Lua scripts and provides key TC execution metrics.
State of the TC formation agent	Each agent state is represented as a suffix of the TC being formed. The TC suffix is limited to a fixed number of the most recent TC actions.
Action space	The action space $\mathcal{A}$ consists of API calls of the C++ library under test that were collected as a result of executing the original TS. It includes only admissible TC actions with valid parameters. The TC action space is updated dynamically to prevent invalid combinations of actions during TC formation.

Continuation of table 1

Policy	The TC formation agent follows a decision-making policy implemented as an ε-greedy strategy for selecting the next TC action. The next action is selected based on the Q-table.
Exploration strategy of the TC action space	At the beginning of training, the agent uses a high value of ε within the ε -greedy policy to explore the action space. Subsequently, ε is gradually decreased using a linear decay function over the training episodes of the TC formation agent.
Value function	The TC formation agent updates values in the Q-table, which stores estimates of the expected cumulative reward for each (state, TC action) pair.
Reward function	After executing a TC in the execution environment of the C++ library under test, the TC formation agent receives a reward calculated as a combination of branch coverage gain, reduction of TC redundancy, and a penalty for excessive TC length.
Training process	The training process of the TC formation agent is based on the classical Q-learning model. The training process consists of two phases.
Pre-training phase	For each TC from the original TS, the agent performs imitation learning by sequentially replaying the TC actions from the original TS. This approach allows initialization of the Q-table with relevant estimates of TC actions.
Main training phase	After pre-training, the TC formation agent proceeds to the main training phase. For each TC from the original TS, a fixed number of training episodes is executed. The Q-table is stored and reused across different TCs, enabling transfer of experience between structurally similar TCs.
Convergence criteria	Training continues until a predefined number of training episodes is reached or until stabilization of the loss function defined as the Bellman mean squared error. Additionally, the cumulative reward per training episode received by the TC formation agent is monitored as an auxiliary indicator of training effectiveness.

In reinforcement learning, the quality of decisions made by the TC formation agent depends on the representation of the agent's state. During TC formation for C++ libraries, the so-called curse of dimensionality arises: the number of possible states of the TC formation agent grows exponentially with the increase in complexity of the API of the C++ library under test. This well-known phenomenon is described as the *state explosion problem*.

The interaction of reinforcement learning components during the training process of the Q-learning agent is presented in Fig. 5 in the form of a flowchart.

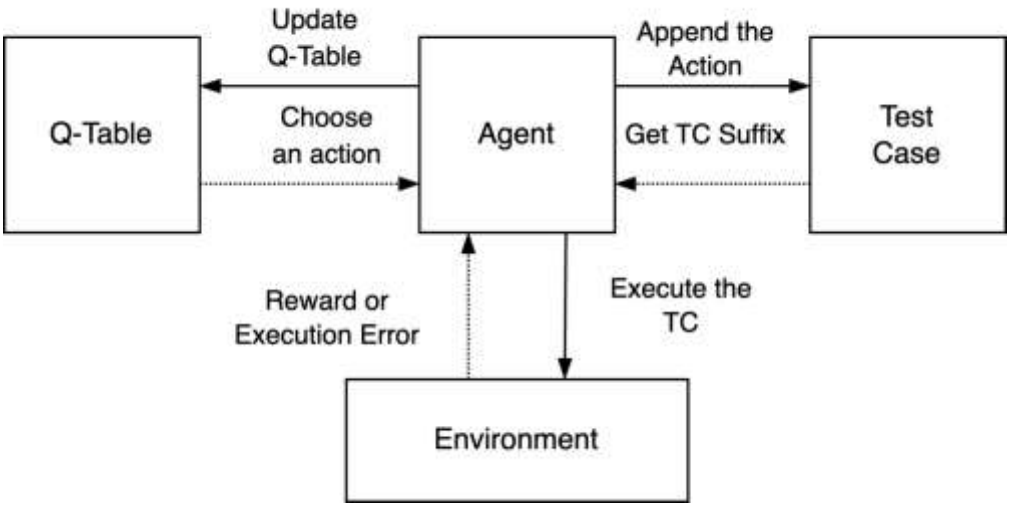


Figure 5. Flowchart of the reinforcement learning process of a Q-learning-based agent

To address the problem and reduce the computational complexity of the mathematical model of the TC formation agent for C++ libraries, the agent state s_{n_k} is proposed to be represented as a function of the TC suffix.

The following assumption is introduced. At the TC formation step n_k, there exists a current sequence of API calls of the C++ library under test corresponding to the given TC:

$$TC^{n_k} = \langle a_1, a_2, \dots, a_{n_k} \rangle, \; a_i \in \mathcal{A}. \tag{28}$$

It is proposed to define the state of the TC formation agent using the TC suffix function:

$$s_{n_k} = \begin{cases} \varnothing, & if \; n_k = 0. \\ \langle a_1, \dots, a_{n_k} \rangle & if \; 0 < n_k \leq \ell_{suf}. \\ \langle a_{n_k - \ell_{suf}+1}, \dots, a_{n_k} \rangle & if \; n_k > \ell_{suf}. \end{cases} \tag{29}$$

where ℓ_{suf} is the length of the TC suffix.

The Q-table is used as a model for approximating the action-value function $Q(s,a)$ in the classical Q-learning model of the TC formation agent for C++ libraries. Each "key" in the Q-table is defined using the TC suffix function (29). The action space $\mathcal{A}$ contains the set of admissible API calls determined based on the objects and functions of the C++ library under test that are available in the current agent state.

Each TC action $a \in \mathcal{A}$ has a numerical Q-value stored in the Q-table. The Q-value represents the expected cumulative reward when selecting action a in each state of the TC formation agent. Each element of the Q-table can be interpreted as a node from which edges extend to possible continuations of the TC after selecting the next action from the action space $\mathcal{A}$.

The information collected from a TC action for representing the agent state according to (29) includes the following:

– the name of the API function of the C++ library under test;

– the number of parameters of the API function of the C++ library under test.

The proposed representation of the state of the TC formation agent for C++ libraries is not fully Markovian in the classical sense. It does not include information about the parameter values of the API function of the C++ library under test. Two sequences of TC actions with identical API function names but different parameter values will have the same state key of the TC formation agent. Despite the absence of full Markov properties, such a level of abstraction may be sufficient for effective learning of the TC formation agent.

Thus, the proposed state representation:

– ensures an acceptable training speed of the TC formation agent;

– prevents excessive branching of the state space;

– reduces the amount of external memory (e.g., disk storage) occupied by the Q-table after training;

– enables the agent to form TCs without losing decision-making effectiveness when working with C++ libraries that have a broad API;

– provides the possibility of implementing a TC formation agent that is stable, controllable, and interpretable, which is important for continuous integration systems where transparency of the TC formation process is critical.

Therefore, the proposed state representation of the TC formation agent constitutes a rational compromise between TC formation efficiency, the ability to generalize across agent states, and the computational complexity of the mathematical model of TC formation, which is necessary for practical TC formation for C++ libraries.

The following assumption is introduced. Each partially formed TC (TC^{n_k}) has a set of TC suffixes computed according to (29), which can be represented as follows:

$$S_{suf} = \{s_{n_k}^i | i = 1, .., \ell_{suf}\}. \quad (30)$$

Such an approach makes it possible to represent the Q-table as a set of estimates for TC suffixes of different lengths as follows:

$$Q(s,a) = \{Q(s,a)^{(1)}, Q(s,a)^{(2)}, \dots, Q(s,a)^{(\ell_{suf})}\}, \quad (31)$$

where $Q(s,a)^{(1)}$ is the estimate for a TC suffix of length 1; $Q(s,a)^{(\ell_{suf})}$ is the estimate for a TC suffix of length ℓ_{suf}.

The concept of spatial influence of test case suffixes is based on the theory of sequence generalization in reinforcement learning [77].

Let, during the Q-value update according to (27), the TC formation agent distribute the temporal difference error (δ^{n_k}) not only to the current TC state represented as a suffix of length ℓ_{suf} (29), but also to all its shorter suffixes. A multi-level Q-table update rule is proposed, which consists in updating Q-values for each suffix length level i from 1 to ℓ_{suf}, with decreasing influence for shorter TC suffixes as follows:

$$Q\left(s_{n_k}^i, a_{n_k}\right) \leftarrow Q\left(s_{n_k}^i, a_{n_k}\right) + \alpha \cdot {k_{sd}}^{\ell_{suf}+1-i} \cdot \delta^{n_k}, \quad (32)$$

where $k_{sd} \in (0,1)$ is the coefficient controlling the decrease in influence of shorter TC suffixes.

The proposed rule is based on classical reinforcement learning principles described in [76], where exponential decay of the influence of earlier gained experience is assumed. The use of the coefficient k_{sd} to reduce the influence of shorter TC suffixes represents discounting of Q-values in the structural dimension of shorter TC suffixes, in contrast to temporal reward discounting (eligibility traces). This approach is based on the theory of state aggregation [78], which enables generalization of Q-value estimates for similar agent states.

Thus, when selecting a TC action $a \in \mathcal{A}$, a generalized evaluation of Q-values for the available TC actions is performed. This evaluation is computed as a weighted average of Q-values over all suffix levels of the current state of the TC formation agent:

$$\tilde{Q}(s,a) = \left. \sum_{i=1}^{\ell_{suf}} w_i \cdot Q(s,a)^{(i)} \right/ \sum_{i=1}^{\ell_{suf}} w_i, \quad (33)$$

where w_i is the weight coefficient of the TC suffix of level i, determined by the following formula:

$$w_i = (N_i + \xi)^{\beta} \cdot {k_{sd}}^{\ell_{suf}+1-i}, \quad (34)$$

where N_i is the number of Q-value updates for the suffix of level i; ξ is a small stabilizing constant; $\beta \in (0,1]$ is the experience amplification coefficient.

Thus, (33) describes the essence of the proposed multi-level Q-value generalization rule [79]. The weights w_i combine the frequency of Q-value updates for the TC suffix and the coefficient for reducing the influence of shorter TC suffixes. This

approach allows the agent to use information from previous states and make more effective decisions during TC formation.

The flowchart of Procedure 2 for multi-level generalization of Q-values based on TC suffixes of different lengths is presented in Fig. 6.

Figure 6. Flowchart of the Procedure 2 for multi-level generalization of Q-values based on TC suffixes

The proposed procedure reveals the essence of the mechanism of multi-level Q-value generalization for the set of available TC actions. For each suffix of the current state $s_{n_k}^i$, the weight coefficient w_i is computed according to (34), considering the update frequency of the corresponding Q-value.

The averaged estimates for each TC action $\tilde{Q}(a)$ are formed as a normalized weighted average of Q-values over all levels of TC suffixes of different lengths. The resulting table of generalized Q-values obtained by Procedure 2 can be used by the TC formation agent for decision-making during training or TC formation.

The policy for TC actions selection is one of the key components of the TC formation agent model. During training, the policy determines the balance between exploration of the action space – testing new combinations of actions for TC formation, and exploitation – selecting actions based on previously acquired experience in TC formation.

The TC action selection policy can be defined as follows.

– The ε-greedy policy for TC actions selection – with probability ε, the TC formation agent selects a random action a_{n_k} from the set of available actions $\mathcal{A}(s_{n_k})$ for the current agent state s_{n_k}. With complementary probability $(1-\varepsilon)$, the TC action with the highest Q-function value is selected as follows:

$$a_{n_k} = \begin{cases} \underset{a' \in \mathcal{A}(s_{n_k})}{arg\ max}\, Q(s_{n_k}, a'), & if\ (1-\varepsilon) \\ rand\ \mathcal{A}(s_{n_k}), & if\ \varepsilon \end{cases}; \quad (35)$$

– The softmax policy for TC actions selection – performs action selection based on their Q-values according to the softmax distribution, which can be written as follows [76]:

$$P(a_{n_k}|s_{n_k}) = \frac{e^{Q(s_{n_k}, a_{n_k})/\tau}}{\sum_{a' \in \mathcal{A}(s_{n_k})} e^{Q(s_{n_k}, a')/\tau}}, \quad (36)$$

where τ is a coefficient that regulates the influence of stochasticity on the TC action selection policy.

From (36) it follows that high values of parameter τ lead to nearly equal selection probabilities for different TC actions, which corresponds to almost random action selection. When $\tau \to 0$, the behavior of the TC action selection policy approaches greedy selection, where preference is given only to actions with the highest Q-values. The advantage of the softmax policy is that it enables a smooth transition between active exploration of the TC action space and exploitation of accumulated knowledge.

During training, the agent applies the ε-greedy policy for selecting the next TC action, where actions are chosen based on the multilevel averaging of Q-values computed for all suffixes of the current TC state of different lengths according to (33). Then, the probability of selecting the next TC action $a_{n_k} \in \mathcal{A}(s_{n_k})$ in the agent state s_{n_k} is defined as:

$$a_{n_k} = \begin{cases} \underset{a' \in \mathcal{A}(s_{n_k})}{arg\ max}\, \tilde{Q}(s_{n_k}, a'), & if\ (1-\varepsilon) \\ rand\ \mathcal{A}(s_{n_k}) & , \ if\ \varepsilon \end{cases}, \tag{37}$$

where $\varepsilon \in (0,1)$ is the exploration stochasticity coefficient.

The softmax policy for TC actions selection is improved by incorporating the generalized Q-value estimate $\tilde{Q}$ (33) into (36) as follows:

$$P(a_{n_k}|s_{n_k}) = \frac{e^{\tilde{Q}(s_{n_k}, a_{n_k})/\tau}}{\sum_{a' \in \mathcal{A}(s_{n_k})} e^{\tilde{Q}(s_{n_k}, a')/\tau}}. \tag{38}$$

Thus, formula (37) describes the improved ε-greedy policy for TC actions selection, and formula (38) describes the improved softmax policy, in which the next action is selected based on the generalized Q-value estimate computed for all shorter suffixes of the TC. This approach to defining action selection policies combines the

simplicity of the classical Q-learning model with multilevel Q-table generalization, ensuring training stability and improving the effectiveness of the TC formation.

Development of the reward function for the TC formation agent.

The reward function of the TC formation agent for C++ libraries consists of two levels: an immediate reward provided after each selected TC action, and a final reward assigned to the fully formed TC.

The first level of the reward function, the stepwise reward for TC formation, controls the local growth of branch coverage at each step of TC formation, encouraging the agent to select actions that provide an immediate increase in branch coverage according to (6). This approach enables the TC formation agent to gain experience and update Q-values after each formation step, which is important for C++ libraries with wide and diverse APIs. The second level of the reward function controls the overall effectiveness of the fully formed TC.

In addition to increasing the branch coverage of the C++ library under test, it is important to avoid redundant and semantically repetitive TC actions during formation. To detect semantic similarity of software API calls, the identifiers of API functions and the types of API call parameters are used. Thus, the semantic similarity function of TC actions can be defined as follows:

$$\mathcal{J}(a_i, a_j) = \begin{cases} 1, if\ TC\ action\ function\ identifiers\ match \\ 0,\ otherwise \end{cases}. \quad (39)$$

Each step of TC formation performed by the agent, depending on the execution result of the partially constructed TC in the execution environment of the C++ library under test, is assigned a reward determined by the stepwise reward function, which takes into account the following characteristics of TC execution:

- increase in branch coverage of the C++ library under test;
- discovery of new program basic blocks by the newly selected TC action;
- repetition of semantically similar TC actions;

– absence of branch coverage increase of the C++ library under test as a result of execution.

The indicator of branch coverage increase for the C++ library under test due to action a selected by the agent at step n_k can be defined as follows:

$$v_{n_k}(a_{n_k}) = \begin{cases} 1 & , \ if\ \Delta C_{n_k} > 0 \\ 0 & , \ otherwise. \end{cases}. \tag{40}$$

The uniqueness indicator u_{n_k} (20) informs about the discovery of new program basic blocks. Thus, the stepwise reward function of the agent for action a_{n_k} selected at state s_{n_k} during TC formation can be defined as follows:

$$\begin{aligned} r_{n_k}(a_{n_k}) = \alpha_1 \cdot v_{n_k}(a_{n_k}) + \alpha_2 \cdot u_{n_k}(a_{n_k}) + \\ +\beta_1 \cdot \mathcal{J}(a_{n_k}, a_{n_k-1}) + \beta_2 \cdot \left(1 - v_{n_k}(a_{n_k})\right), \end{aligned} \tag{41}$$

where $\alpha_1, \alpha_2 > 0$ are reward coefficients, and $\beta_1, \beta_2 < 0$ are penalty coefficients.

After completion of TC formation, the agent is assigned a second-level reward, namely the final reward for the training episode of TC formation, which takes into account the following characteristics of the generated TC :

– increase (or preservation) of branch coverage of the C++ library under test relative to the original TC;

– degradation of branch coverage of the C++ library under test relative to the original TC.

The indicator of increase or decrease in branch coverage of the formed TC (TC) relative to the original TC (TC_0) can be defined as follows:

$$v_{TC} = \begin{cases} 1, & if\ C_{TC} > C_{TC_0} \\ 0, & if\ C_{TC} = C_{TC_0} \\ -1, & if\ C_{TC} < C_{TC_0} \end{cases}. \tag{42}$$

Thus, the final reward function of the agent for complete TC formation can be defined as follows:

$$R_{TC} = \alpha_3 \cdot v_{TC}, \quad (43)$$

where $\alpha_3 > 0$ is the reward or penalty coefficient depending on the sign of (42).

Therefore, based on the proposed mathematical model of TC formation for C++ libraries based on a Q-learning agent, it is possible to improve the Q-learning training algorithm and to develop a TC formation algorithm for C++ libraries based on a Q-learning agent.

Transfer Learning (TL) enables the transfer of accumulated experience between similar tasks or environments for the reuse of knowledge under substantially different conditions [80]. Preserving the Q-table across TCs of the same TS of the C++ library under test provides a partial knowledge transfer effect, which is close to TL. In particular, the use of original TCs for training the TC formation agent allows the agent to make decisions leading to an increase in branch coverage of the C++ library under test more rapidly during training.

Thus, the following training strategy is proposed. The agent is trained within the action space corresponding to each TC from the original TS of the C++ library under test, while maintaining a single Q-table within one original TS. The experience gained by the agent during training in the action space of a particular TC from the original TS of the C++ library under test is partially transferred to the next TC from the same TS.

Before each training of the agent on a new TC, one imitation pass of the initial API call sequence from the original TC is performed without applying the action selection policy. This approach serves as a pre-training phase for the agent and initializes or supplements the Q-table with relevant data, thereby enabling the agent to orient itself more quickly within a new TC action space.

At the beginning of training, the TC formation agent requires a higher exploration rate of the TC action space, using the exploration coefficient ε in (35, 37) as a regulator of exploration. During the training process, it is advisable to gradually decrease the value of coefficient ε in order to consolidate the formed policy for selecting TC actions during exploration of the action space.

The learning rate coefficient α in (27) should also be dynamically adjusted during agent training depending on the number of completed training episodes. Gradual reduction of the learning rate coefficient α stabilizes Q-table updates. This approach allows the agent to stabilize its decision-making policy and to adapt effectively to new TCs.

The linear decay function of variable x [81] can be applied to compute the learning rate coefficient α and the exploration coefficient ε as follows:

$$x \leftarrow f(n_{ep}) = x_o - \frac{(x_o - x_{final})}{N_{ep}} \cdot n_{ep}, \;\; x_o > x_{final}, \tag{44}$$

where x_o is the initial value of the variable x; x_{final} is the lower bound of the variable x; $n_{ep} \in \mathbb{N}$ is the current episode number of agent training, and $N_{ep} \in \mathbb{N}$ is the total number of training episodes.

To evaluate the effectiveness of training a Q-learning agent, it is necessary to define training performance criteria. The criterion of growth in the cumulative reward of the agent per training episode during Q-learning can be defined as follows. The cumulative reward of the agent per episode R_{ep} can be computed as:

$$R_{ep} = \sum_{n_k=0}^{N_k} r_{n_k}, \tag{45}$$

where r_{n_k} is the reward obtained at TC formation step n_k, and N_k is the total number of TC formation steps within the agent's training episode.

An increase in R_{ep} over training episodes indicates the agent's ability to form effective sequences of TC actions. Monitoring the dynamics of R_{ep} makes it possible to detect stagnation or regression in the learning process. This approach is widely accepted for evaluating the performance of Q-learning agents [76].

In addition, the criterion of decreasing the average loss function value per episode is used to assess the effectiveness of Q-learning. The episode loss function can be defined as the Mean Squared Loss (MSL) of the Bellman error as follows:

$$\mathcal{L}_{ep}(Q) = \frac{1}{2}\left(Q\left(s_{n_k}, a_{n_k}\right) - \left[r_{n_k} + \gamma \cdot \max_{a' \in \mathcal{A}(s_{n_k+1})} Q\left(s_{n_k+1}, a'\right)\right]\right)^2. \quad (46)$$

The loss function (46) is computed after updating the Q-table according to (27) and reflects how much the current utility estimate of action a_{n_k} – $Q\left(s_{n_k}, a_{n_k}\right)$, deviates from the expected target value determined based on the immediate reward and the estimated maximum future reward of the Q-learning agent. This approach is standard in reinforcement learning methods and is widely used both in classical Q-learning and in Deep Q-Networks (DQN) [82].

Thus, based on the loss function (46), it is possible to determine the following:

– the rate of stabilization of the agent's policy, where smaller fluctuations in the loss curve indicate a more stable decision-making policy;

– the achievement of a balanced policy, where simultaneous decrease of the loss function (46) and increase of the cumulative episode reward (45) indicate convergence of Q-function estimates toward effective values.

The training algorithm of the Q-learning agent for TC formation for C++ libraries has been improved based on the classical Q-learning algorithm. The improved Algorithm 3 is presented in Fig. 7.

To handle invalid sequences of TC actions during TC formation, a retry-based approach is applied. The agent may repeatedly select a TC action for further formation up to a predefined maximum number of retries $N_{retries}$.

Thus, to ensure training stability, the classical Q-learning algorithm has been improved in order to overcome its limitation, where the Q-table is updated only for the current state of the TC formation agent and does not take into account the influence of previous TC actions.

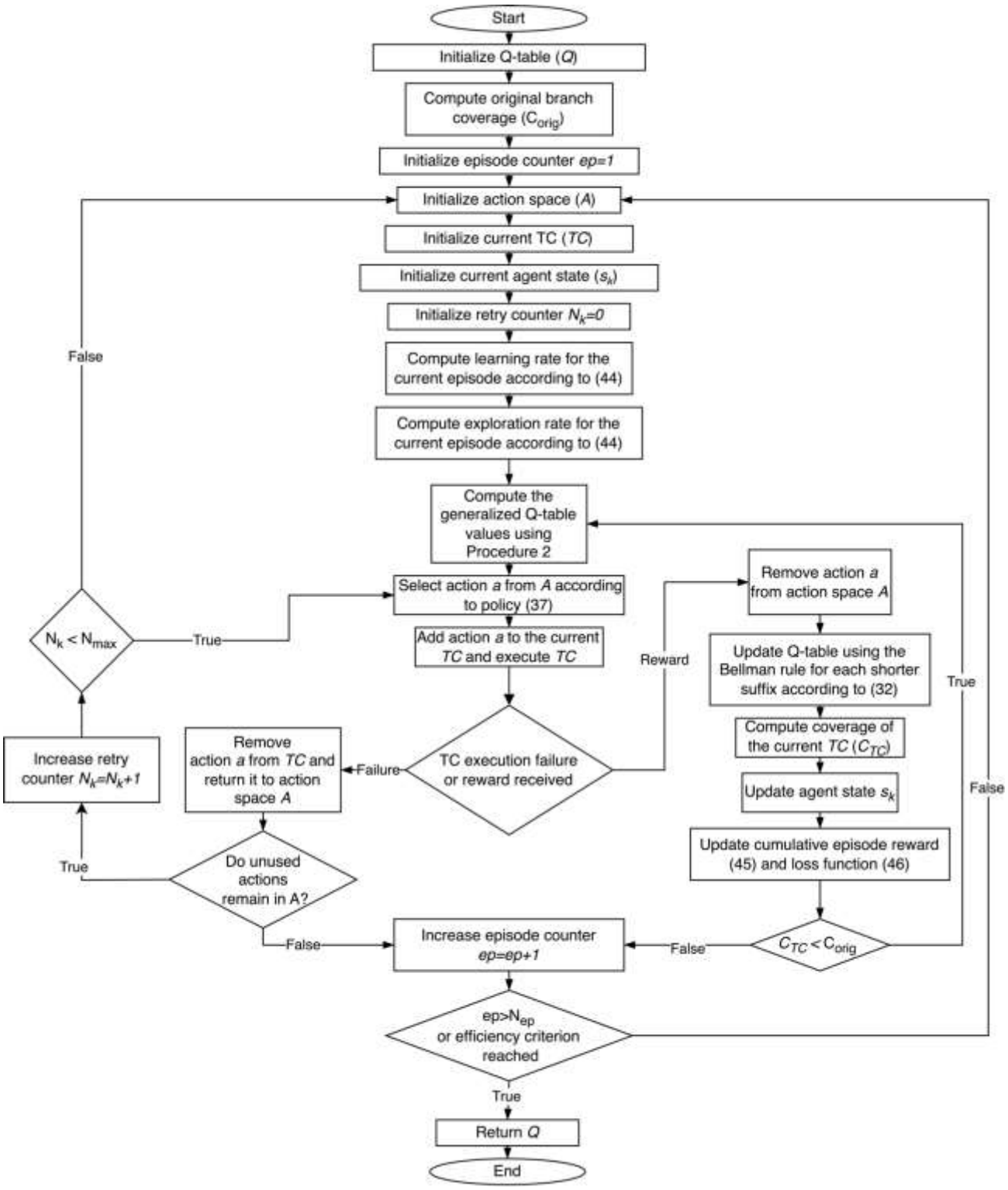


Figure 7. Flowchart of the improved Algorithm 3 for training a Q-learning agent

The improved Algorithm 3 for training a Q-learning agent for TC formation for C++ libraries with multilevel Q-table update introduces the following main modifications to the classical Q-learning algorithm:

– before the step of selecting the next TC action, Procedure 2 is invoked to obtain the generalized Q-value table $\tilde{Q}$. In the classical Q-learning algorithm, TC actions are evaluated only based on the current agent state. Generalization of Q-values allows the agent to take into account historical patterns of TC formation, smoothing Q-value fluctuations;

– the action selection step is performed according to formula (37) using the generalized Q-value table $\tilde{Q}$, which prevents premature fixation of the action selection policy in local optima, since $\tilde{Q}$ contains averaged Q-value estimates;

– the Q-table update step is performed for each suffix length level i of the current agent state from 1 to ℓ_{suf} according to equation (32).

Thus, a mathematical model is proposed that provides generalization of Q-values for EC suffixes of different lengths, enabling the agent to use knowledge acquired at previous EC formation steps to construct new action sequences. The use of this mathematical model increases the robustness of the agent to data sparsity in the Q-table.

The developed Algorithm 4 for TC formation for C++ libraries is presented in Fig. 8.

After completion of training of the TC formation agent for C++ libraries, the Q-table contains utility values for each state–action pair accumulated over multiple training episodes on the original TCs of the C++ library under test. During TC formation, Q-values are no longer updated.

Thus, Algorithm 4 terminates if at least one of the following conditions is satisfied:

– the achieved branch coverage of the generated TC exceeds the branch coverage of the original TC;

– the TC action space is exhausted;

– the predefined maximum number of retries for TC formation is exceeded.

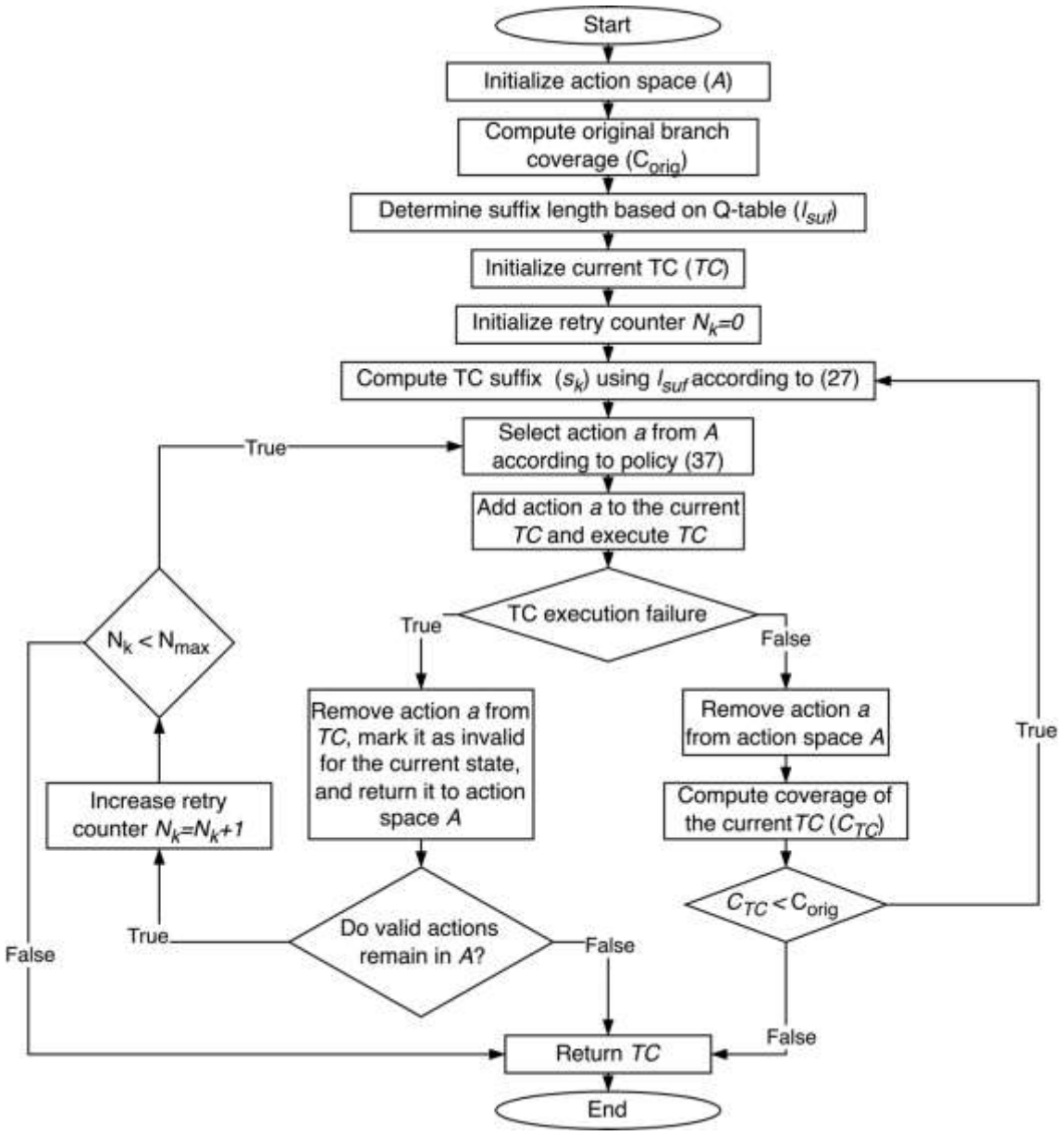


Figure 8. Flowchart of Algorithm 4 for TC formation for C++ libraries

For practical implementation of the Q-learning agent training process for TC formation for C++ libraries, it is necessary to ensure the possibility of execution the runtime code of the C++ library under test and collection of data on API function call identifiers, parameter values of API function calls, return codes of API function calls, and related information. Instrumentation of the API of the C++ library under test is performed using the well-known SWIG tool, which provides the capability to invoke the runtime code of C++ libraries from the test environment. The SWIG tool enables dynamic integration of a C++ library with a Lua interpreter. This approach ensures

execution of the C++ library runtime code within the test environment and allows verification of the behavior of the C++ library during execution of its TS.

During execution of the TS of the C++ library under test, behavioral data and testing effectiveness metrics are collected, including branch coverage and basic block coverage of the C++ library under test. This approach makes it possible to analyze the correctness of API usage of the C++ library under test by examining the behavioral data obtained during execution of its runtime code in the test environment.

The following stages of an instrumentation of the C++ library under test are proposed (Fig. 9).

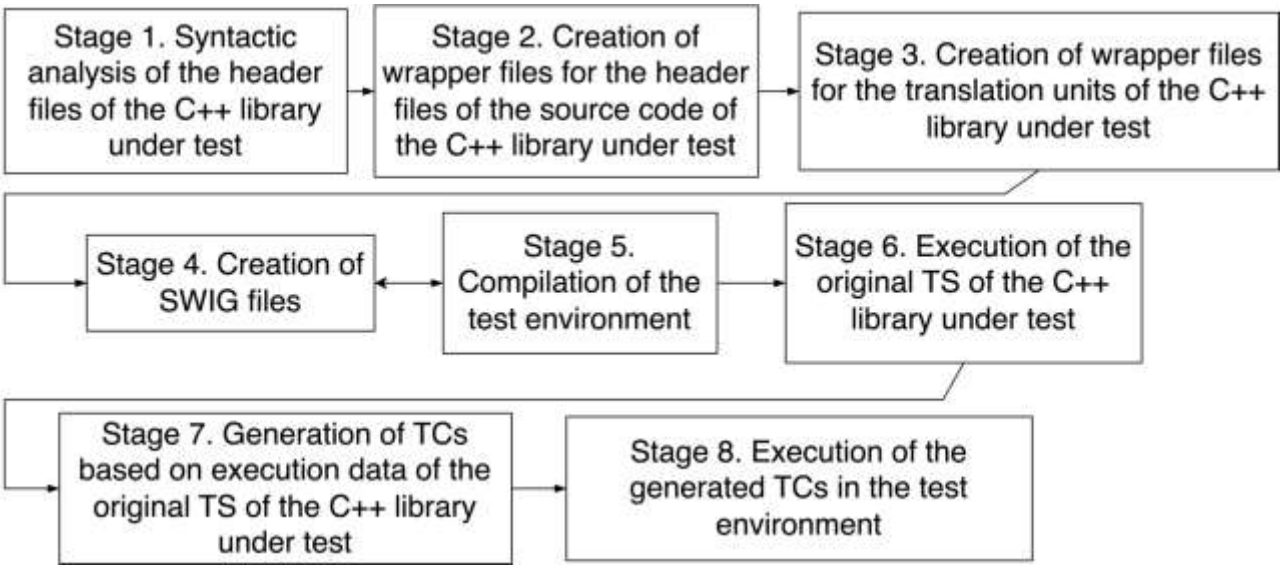


Figure 9. Diagram of an instrumentation of the C++ library under test

Stage 1. Syntactic analysis of header files of the C++ library under test. A syntactic analysis of the header files of the C++ library under test is performed, and an Abstract Syntax Tree (AST) is constructed, representing the API data of the C++ library under test in a form suitable for further processing.

Stage 2. Creation of header wrapper files for the source code of the C++ library under test. Based on the AST obtained in Stage 1, wrapper files for the header files of the C++ library under test are generated. These wrapper files replicate the syntactic

structure of the API of the original header files of the C++ library under test but introduce an additional top-level namespace.

Stage 3. Creation of wrapper files for the translation units of the C++ library under test (e.g., cpp wrapper files). Based on the AST obtained in Stage 1, wrapper files for the translation units of the C++ library under test are generated. These wrapper files replicate the syntactic structure of the API of the original translation units of the C++ library under test but contain C++ code for each original API function that enables the following:

– collecting identifiers of each API function executed in the test environment;

– collecting parameter values of each API function executed in the test environment;

– tracking creation and destruction of API class objects of the C++ library under test;

and, after collecting the required information, invoking the original API function of the C++ library under test while preserving its original behavior.

Stage 4. Creation of SWIG files. Based on the AST obtained in Stage 1, SWIG configuration files (e.g., *.swig) and special translation unit files (e.g., Lua bindings) are generated. Their use enables dynamic integration of the C++ library under test with the Lua interpreter serving as the test environment.

Stage 5. Compilation of the test environment. The test environment in the form of a Lua interpreter is compiled together with the wrapper files created in Stages 2–4. This approach ensures the ability to execute the C++ library under test within the test environment.

Stage 6. Execution of the original TS of the C++ library under test. During execution, the necessary runtime data of the C++ library under test are collected.

Stage 7. Generation of TCs based on the collected execution data. The data obtained in Stage 6 are converted into Lua code that represents a detailed reconstruction of the actual sequence of actions of the original TCs of the C++ library under test.

Stage 8. Execution of the generated TCs in the Lua interpreter. The generated TCs from Stage 7 can be executed in the test environment to obtain behavioral data of the C++ library and testing effectiveness metrics, including branch coverage and basic block coverage of the C++ library under test.

Thus, the instrumentation sequence describes all stages of processing the C++ library under test required for the development of a method of TC formation for C++ libraries based on a Q-learning agent.

A method of TC formation for C++ libraries based on a Q-learning agent [81, 83] has been developed (Fig. 10), which includes the following stages.

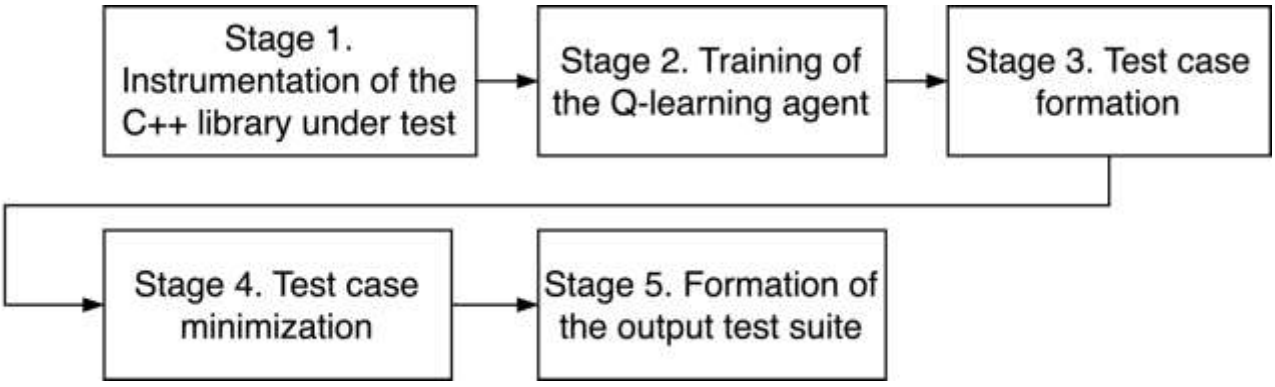


Figure 10. Diagram of the method of TC formation for C++ libraries based on a Q-learning agent

Stage 1. Instrumentation of the C++ library under test. At the preparatory stage, instrumentation of the C++ library under test is performed. During execution of the initial TCs, sequences of API calls are logged, including function signatures, objects, call contexts, and arguments. The collected data are subsequently used as the action space for training the Q-learning agent.

Stage 2. Training of the Q-learning agent is performed using the developed mathematical model of TC formation for C++ libraries based on a Q-learning agent. Based on the TC action space, the agent is trained in the test environment according to Algorithm 3.

Stage 3. TC formation is performed using the developed mathematical model of TC formation for C++ libraries based on a Q-learning agent. After completion of Stage 2, the agent's memory (e.g. Q-table) is used to generate new TCs according to Algorithm 4.

Stage 4. Test case minimization (e.g., TC length reduction). After completion of Stage 3, the improved Algorithm 1 for delta debugging test case minimization is applied.

Stage 5. Formation of the output test suite. After completion of Stage 4, the obtained TCs form a new TS.

The developed method of TC formation for C++ libraries based on a Q-learning agent ensures an increase in branch coverage of the C++ library under test while simultaneously reducing the length of the TS. At the same time, the formed TCs by the agent may still contain steps that do not contribute to the achieved branch coverage, which creates the possibility for further minimization of the generated TCs.

In the proposed method, the improved Algorithm 1 of delta debugging acts as a TC minimization filter. Thus, the method combines the adaptability of reinforcement learning with the precision of TC minimization provided by the improved delta debugging algorithm, ensuring guaranteed TC minimization without loss of branch coverage of the C++ library under test.

Based on the evaluation results of TC action selection policies [79], it was determined that at Stage 3 the improved action selection policy (38) should be used, as it incorporates generalized Q-values for test case suffixes of different lengths.

Thus, a novel method of TC formation for C++ libraries based on a Q-learning agent has been developed. The method applies a classical Q-learning agent to generate sequences of API calls, relying exclusively on execution feedback obtained in the runtime environment of the C++ library. Unlike known TS optimization methods that operate by selecting or minimizing existing TCs , the proposed method forms new TCs guided by an improved decision-making policy trained on the behavioral characteristics of the C++ library under test.

Conclusions. Thus, a novel method of TC formation for C++ libraries based on a Q-learning agent has been developed.

The specific features of testing C++ libraries have been analyzed. The main problems of TC formation and optimization have been considered, and examples of known approaches to solving these problems have been presented.

Mathematical models and algorithms for TC formation for C++ libraries have been investigated.

The essence of the improved delta debugging algorithm for TC minimization has been revealed. Unlike existing approaches, the proposed algorithm considers the non-uniqueness criterion of TC actions when forming action groups for validation, which allows preserving full branch coverage while reducing TC length with fewer branch coverage computation calls.

The essence of the improved mathematical model of the greedy TC formation algorithm for C++ libraries has been disclosed. The algorithm incorporates the future utility of actions into the objective function and applies the softmax rule to select the next TC action.

The essence of the improved greedy TC formation algorithm has been described. The algorithm has high potential for branch coverage and generates shorter and more effective TCs compared to the classical greedy TC formation algorithm.

A mathematical model for Tc formation for C++ libraries based on a Q-learning agent has been developed. The model describes the agent state using TC suffixes of different lengths, incorporates a coefficient controlling the influence of shorter suffixes during utility function updates, and applies a generalized Q-value during action selection. The mathematical model uses the improved Q-learning training algorithm and the developed TC formation algorithm for C++ libraries.

A novel method of TC formation for C++ libraries based on a Q-learning agent has been developed. It is based on the proposed mathematical model and the improved delta debugging TC minimization algorithm for C++ libraries, and allows increasing or preserving branch coverage of the original TS while reducing execution time, thereby improving the overall effectiveness of software testing.

DOI: 10.46299/979-8-90383-421-1.4

4. Method to improve the safety of unmanned aerial vehicle flights in urban areas based on machine learning technologies

Introduction. The operating conditions for unmanned aerial vehicles (UAVs) include both airspace and ground space, covering densely populated areas such as urban and suburban areas. The integration of UAVs into urban airspace is accompanied by numerous challenges related to flight safety, regulatory and technical constraints.

Pilot-controlled aviation has achieved an exceptional level of safety over the past fifty years. Therefore, society expects a similar level of reliability from unmanned aviation. The use of modern machine learning and deep learning methods significantly increases the safety of UAV flights, their efficiency, and flexibility in urban environments. Such applications make research relevant and promising for the further development of autonomous UAV flight technologies in urban spaces.

High building density and infrastructure in cities create difficult conditions for UAV maneuvering, requiring careful calculation of flight trajectories. Existing approaches, such as the introduction of the D2-C2 method as part of the USEPE project, enable new approaches to strategic and tactical UAV control. However, they have the following disadvantages: high computational complexity, limited adaptability to sudden changes in traffic, and the need for large computing resources. Therefore, further development of this approach requires the implementation of machine learning algorithms that will increase the efficiency and flexibility of the UAV flight control system in real operating conditions.

Consequently, developing a method to improve the safety of UAV flights in urban areas based on machine (deep) learning technologies is a relevant scientific task.

Main material. The safety of UAV flights in urban environments is a major challenge for modern air navigation. Planning and managing UAV flights in conditions of limited airspace and high obstacle density requires the development of a comprehensive scientific research methodology. The methodology should consider not only existing approaches to UAV flight management, but also advanced technologies

such as: Dynamic Density Corridor Concept (D2-C2), deep neural networks (Long Short-Term Memory, LSTM) for risk prediction, and reinforcement learning (RL) algorithms for adaptive adjustment of UAV flight trajectories.

The methodology for scientific research into UAV flight safety is based on a multi-level approach, which includes the following stages:

- development of a conceptual model for UAV air traffic management, which involves analyzing existing approaches and adapting them to urban conditions based on the results of research of the basic principles of the D2-C2 concept;
- determination of a mathematical model of a dynamic flight safety corridor for UAVs, which includes the identification of parameters and algorithms for controlling UAV flight corridors in accordance with the basic concept, as well as justification of proposals for its improvement;
- integration of artificial intelligence methods, which involves using the LSTM model to predict potential risks and RL algorithms for autonomous correction of UAV flight trajectories;
- modeling and simulation testing to verify the effectiveness of the proposed methods in a virtual environment;
- analysis of the results obtained and validation, which ensures the assessment of the effectiveness of the developed system for reducing the risks of UAV collisions with obstacles in urban environments.

The structure of the scientific research methodology has been developed and is shown in Fig. 1.

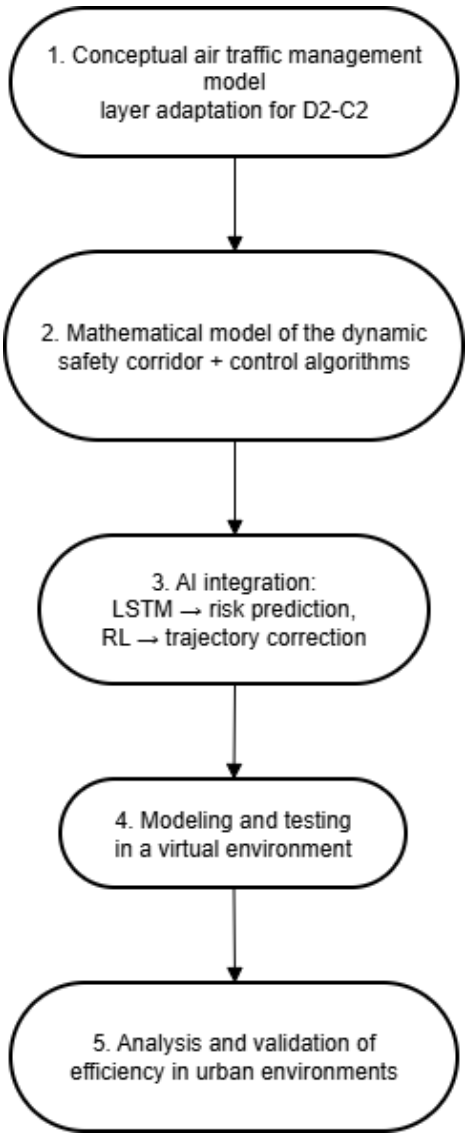


Figure 1. Structure of the methodology for scientific research on the safety of UAV flights in urban environments

The mathematical model for controlling UAV flights in urban environments is based on a system of equations that describe the kinematics of UAV motion, safety restrictions, and trajectory adaptation algorithms. The UAV flight equations can be expressed using the kinematic law as follows:

$$r(t + \Delta t) = r(t) + v(t)\Delta t + \frac{1}{2}a(t)(\Delta t)\text{^}2, \quad (1)$$

where r(t) – the position of the UAV, v(t) – the velocity, a(t) – the acceleration.

For advanced forecasting of UAV flight trajectories, an LSTM model is used, which allows considering the dynamics of movement and identifying potential conflict points based on the state of the system at subsequent points in time:

$$S_{t+1} = f(S_t, A_t) + \varepsilon, \quad (2)$$

where S_t – the state of the system at time t, A_t – a set of actions, and ε – the model error.

Meanwhile, reinforcement learning (RL) is used to autonomously adjust UAV flight paths in case of unexpected obstacles. The reward function for this algorithm integrates the distance to the obstacle, the collision risk penalty, and the energy consumption indicator as shown below:

$$R(s, a) = -d + \beta C_{safe} - \gamma C_{energy}, \quad (3)$$

where d – the distance to the obstacle, C_{safe} – the collision risk penalty, C_{energy} – the energy consumption.

The basis for air traffic management is the latest D2-C2 hybrid method, developed as part of the SESAR USEPE project. The method combines an approach to airspace separation based on UAV flight density using dynamic segments and high-speed flight corridors, and defines geo-vectoring rules to ensure safe separation of UAVs in urban environments.

The D2-C2 method provides strategic and tactical separation of large numbers of UAVs at low altitudes, preventing collisions by creating dynamically changing safe flight corridors. D2-C2 is based on a set of the following interrelated elements:

– Dynamic airspace segments – space is divided into local volumes and boundaries, the configuration of which changes dynamically depending on traffic density and operational needs. As the number of UAVs increases, the segment can be automatically fragmented to redistribute flows and reduce local density, and when traffic decreases, it can be merged. In areas with extremely high intensity, multi-layer

segments can be used to distribute UAVs at different altitude levels, based on their characteristics;

– High-speed corridors – special dynamic air routes are formed, designed for the transit of UAV flows at higher speeds in one general direction. Aligning courses significantly reduces the number of potential UAV collision conflicts, as it eliminates the intersection of their flight paths at acute angles. Corridors operate on the principle of high-speed highways, where minimum and maximum speed limits can be set to maintain the uniformity of UAV traffic. The geometry of such corridors is coordinated with street plans to minimize risks. Fig. 2 shows the concept of the D2-C2 method.

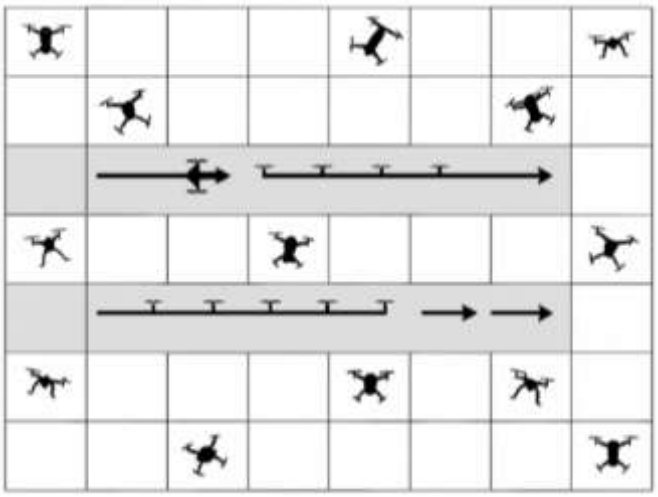


Figure 2. Schematic illustration of the D2-C2 method concept

In Fig. 2, dynamic segments (airspace cells) are rearranged when traffic density changes. High-speed corridors are formed along the main directions, where UAVs move in one direction at a given speed;

Geo-vectoring syntax – this method uses predefined standardized instructions for UAV flight directions and speeds within segments. Geo-vectoring defines a single command syntax so that UAVs automatically follow the desired profile, making their behavior highly predictable for the control system;

Density-based separation – control based on the assessment of dynamic density – a variable parameter that characterizes the load on space. Dynamic density can be

calculated as the number of UAVs in a given volume as follows:

$$\rho = \frac{N}{A}, \tag{4}$$

where ρ – traffic density (e.g., number of UAVs per 1 km^2).

Density limits (thresholds) are introduced for each segment. When these limits are exceeded, the U-space system automatically reconfigures the space structure to prevent congestion.

For a formal description and calculation of the D2-C2 method concept, a mathematical model of a dynamic UAV flight safety corridor has been developed, which consists of several interrelated levels. Within the density and segmentation model, provided that at time t there is a certain number of UAVs in the segment, the current density of UAVs can be calculated using the following expression:

$$\rho_i = \frac{N_i(t)}{A_i}. \tag{5}$$

The criteria for the stability of the segment is compliance with the inequality regarding the threshold value:

$$\rho_i < \rho_{max}^{(i)}. \tag{6}$$

Provided that the limit is reached:

$$\rho_i \leq \rho_{max} \tag{7}$$

then the reconfiguration rule is activated (division of the segment into subsegments or height layers).

Therefore, the model can be reduced to the constant maintenance of inequalities as follows:

$$\rho_j(t) \leq \rho_{max}^{(j)} \tag{8}$$

for all active segments.

In addition, the probability of accidental convergence of UAVs can be estimated as a function of the square of the local flow intensity as follows:

$$P_{\text{experiment}} \propto p^2, \tag{9}$$

justifies the need for strict restrictions on UAV flight density.

The geometry and kinematics of the UAV flight safety corridor are modeled by the length, width, and height of the spatial area. Safety is determined by the minimum distances between UAVs.

The minimum distance between UAVs on the same trajectory can be mathematically calculated based on speed, time interval, and maximum negative acceleration (braking) as follows:

$$D_{min} = v \cdot \Delta t_{sep} + \frac{v^2}{2a_{max}}, \tag{10}$$

where $\boldsymbol{v}$ – the speed of the UAV, $\boldsymbol{\Delta t_{sep}}$ – the time interval between successive UAVs, and $\boldsymbol{a_{max}}$ – the maximum negative acceleration (deceleration) of the UAV.

Therefore, it is possible to determine the throughput capacity of the UAV flight corridor – the maximum number of simultaneous UAVs – as follows:

$$N_{corr} \approx \frac{L}{D_{min}} \tag{11}$$

The model of rules and conflicts formalizes restrictions on kinematic parameters (course, speed, and altitude). The main condition for the absence of collisions between UAVs can be described by the inequality of maintaining a safe distance between them.:

$$|r_i(t) - r_j(t)| > D_{safe}, \quad \forall t, \tag{12}$$

where $\boldsymbol{D_{safe}}$ – the safe distance between UAVs.

Thus, the mathematical model D2-C2 allows us to bring together the conditions of dynamic balance:

$$\rho \leq \rho_{max} \tag{13}$$

and kinematic intervals:

$$v_{\min} \leq v \leq v_{\max}, \psi_{\min} \leq \psi \leq \psi_{\max}, h_{\min} \leq h \leq h_{\max}. \tag{14}$$

However, the D2-C2 method has a drawback. It relies on current data without extensive use of predictive models. To improve the effectiveness of the D2-C2 method, it is proposed to integrate it with modern artificial intelligence methods as follows.

The first area for improvement of the D2-C2 method is the integration of LSTM neural networks at the strategic planning stage. LSTM analyzes historical UAV flight trajectories and can predict their future positions, which allows potential UAV collision conflicts to be identified in advance. The neural network considers the acceleration and maneuvers of the UAVs, predicting their coordinates several steps ahead. If the model detects that two UAVs will violate the safe distance limit, the U-space system will preventively redirect one of the UAVs to another segment before a real threat arises.

The second area of improvement to the D2-C2 method is the use of reinforcement learning (RL) methods for tactical route correction in real time. The RL algorithm allows the UAV's onboard autopilot to learn how to avoid unpredictable dynamic obstacles, maximizing the target reward. The RL agent locally adjusts the UAV's flight path while remaining within the rules of D2-C2 geovectoring. Acting decentralized based on data from its own sensors, the UAV performs a bypass maneuver and returns to its main course without human intervention.

Table 1

Application of machine learning methods to improve D2-C2

Area of Improvement	Model/Method (AI)	Purpose in the Context of D2-C2	Application Stage	Relationship with D2-C2
Risk-Aware Trajectory Prediction	Recurrent Neural Network (LSTM)	Predicting future UAV positions and assessing conflict probability in advance. Helps optimize the UAV flight trajectory plan before takeoff, avoiding route intersections with other UAVs.	Strategic Planning (pre-flight)	The model works alongside the D2-C2 algorithm: it analyzes the planned UAV flight in segments/corridors and suggests where to change the route or segment parameters. Ensures preventive (proactive) separation.
Autonomous Obstacle Avoidance	Deep Reinforcement Learning (DRL)	Reactive adjustment of the UAV route in-flight to bypass sudden obstacles or other UAVs. Minimizes the risk of collision in unpredictable situations while complying with air traffic rules.	Tactical Control (in-flight, real-time)	The model operates within the constraints of D2-C2: it performs local UAV evasive maneuvers without violating the geometry of the corridor or segment. It complements D2-C2 rules, providing system flexibility and resilience to factors not accounted for during the planning stage.

The proposed synergy ensures the adaptability and stability of the UAV flight control system even in highly uncertain and densely built-up urban environments.

The concept of dynamic density corridors D2-C2 offers an effective solution by organizing space in the form of dynamic segments and high-speed safety corridors for UAV flights. However, it cannot completely eliminate the risks of unforeseen UAV encounters. In particular, this applies to cases caused by dynamic factors such as gusts of wind, communication delays, or the sudden appearance of objects that do not comply with geo-orientation rules. Therefore, there is a need to improve the efficiency of

planning methods by integrating conflict prediction tools based on advanced machine learning technologies, which will allow routes to be adjusted in advance. Therefore, the task of planning UAV flight trajectories must be formalized as a task of predicting time sequences. The flight path of a UAV is considered as a discrete time sequence of its states, which includes spatial coordinates and dynamic parameters at specific time steps.

The distance between the UAV and a potential obstacle can be determined using the appropriate mathematical metric:

$$d_i(t) = |p(t) - p_i(t)| = \sqrt{\left(x(t) - x_i(t)\right)^2 + \left(y(t) - y_i(t)\right)^2 + \left(z(t) - z_i(t)\right)^2}. \quad (15)$$

A threshold safety distance has been introduced. Therefore, a conflict event can be described as a moment within the forecast horizon when this distance becomes less than the acceptable norm as follows:

$$d_i(\tau) < D_{min}. \quad (16)$$

In terms of machine learning, the input data is a sequence of feature vectors that characterize the state of the UAV flight control system at previous moments in time and can be recorded as follows:

$$X = \{s(t_0), s(t_1), \ldots, s(t_k)\}, \quad (17)$$

where $s(t_j)$ – a feature vector that characterizes the state of the "UAV-obstacle" system at time (t_j).

These features may include own coordinates, velocity vector, UAV trajectory parameters, as well as relative coordinates of nearby obstacles and their approach speed. Therefore, to develop a mathematical model, it is necessary to consider the

predicted value of the target variable, which indicates the probability of a conflict occurring in the foreseeable future:

$$f: X \mapsto \hat{y}, \tag{18}$$

where $\hat{y} \approx y$ – predicted value (probability or class of conflict situation) based on analysis of sequence X.

To model such complex spatio-temporal dependencies, a recurrent neural network of the LSTM type was chosen, which is capable of effectively memorizing long-term trends in sequences due to a special controlled memory mechanism. Since the development of a conflict situation has a pronounced temporal dependence, the LSTM network allows considering the sequence of previous states, updating the hidden state vector at each new step. In addition, this architecture adapts to variable sequence lengths, which is important because the duration of UAV approach maneuvers in real conditions can vary significantly.

The calculation of internal memory cell values, such as forget gates, inputs, and outputs, is performed using the following system of differential and algebraic equations:

$$\begin{gathered} f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big), \\ i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \\ \widetilde{C_t} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \\ C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C_t}, \\ o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \\ h_t = o_t \odot \tanh(C_t) \quad , \end{gathered} \tag{19}$$

where $\boldsymbol{\sigma}(\cdot)$ – the logistic sigmoid; $\boldsymbol{tanh}(\cdot)$ – the hyperbolic tangent; $\odot$ – means component-wise multiplication of vectors; vectors $\boldsymbol{W_f}$, $\boldsymbol{W_i}$, $\boldsymbol{W_c}$, $\boldsymbol{W_o}$ and

corresponding $\boldsymbol{b_f}, \boldsymbol{b_i}, \boldsymbol{b_c}, \boldsymbol{b_o}$ – are model parameters (weight coefficients and biases) that are tuned during training.

With this calculation algorithm, the network can figure out on its own which UAV flight patterns to keep for accurate collision prediction and which fluctuations to ignore as random noise. Unlike classical algorithms, LSTM combines the versatility of a nonlinear model with long-term memory of previous maneuvers of the devices.

The structure of the developed neural network model has been adapted to the domain of UAV trajectory control. The input layer receives feature vectors, which undergo normalization and are transmitted to the main computing block of LSTM cells. The system uses a multi-layer architecture with several sequential recurrent layers to ensure the necessary ability to approximate complex trajectories without the risk of overfitting. The general scheme of signal passage through the network layers shows the process of intelligent forecasting and is shown in Fig. 3.

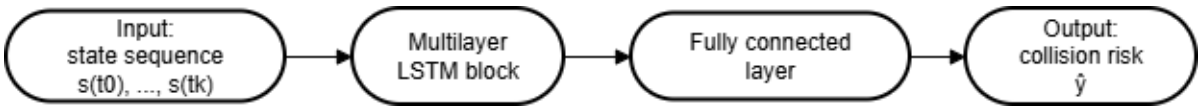


Figure 3. Structural diagram of a neural network model for conflict prediction based on LSTM (in one direction at a given speed)

After the last hidden layer, there is an output layer with a sigmoidal activation function, which generates the final probability of a dangerous event occurring. To ensure stable convergence and avoid excessive sensitivity to individual neurons, screening layers and corresponding weight regularization have been introduced into the architecture. Detailed specifications of the neural network hyperparameters, including layer dimensions and optimizer type, are summarized in Table 2.

Table 2

Parameters of the LSTM model architecture

Parameter	Value
Model Type	Recurrent Neural Network (LSTM)
Number of LSTM Layers	2 (sequential)
Hidden State Dimensionality	64 (each LSTM layer has 64 cells)
Output Layer	1 neuron (sigmoid activation)
Input Feature Dimensionality	6 (e.g., coordinates and velocities of the UAV and obstacle)
Dropout after LSTM Layers	0,2 (20%)
L2 Regularization	0,001
Total Number of Parameters	~50,000 (weights)

The improved planning method functions as a comprehensive combination of the neural model and a logical algorithm for the operational adjustment of the UAV flight path. Structurally, this method works as a cyclical sequence of steps. The process begins with the collection of current telemetry data on the status of the UAV and the surrounding environment to form the current feature vector. The collected sequence of historical states is input into the trained LSTM model, which generates a forecast of the likelihood of conflict within a given time horizon. If the calculated probability exceeds the established safety threshold, the UAV flight route planning system instantly initiates an analysis of the results and selects a preventive action, such as changing the UAV's flight altitude or deviating to a neighboring corridor. When performing the selected maneuver, the autopilot adjusts the subsequent spatial points of the UAV trajectory, eliminating the threat before it gets too close. Thanks to the cyclical nature of the process, continuous monitoring of the situation in real time is achieved. Fig. 4 shows a block diagram of the method for predicting conflicts and avoiding collisions involving UAVs.

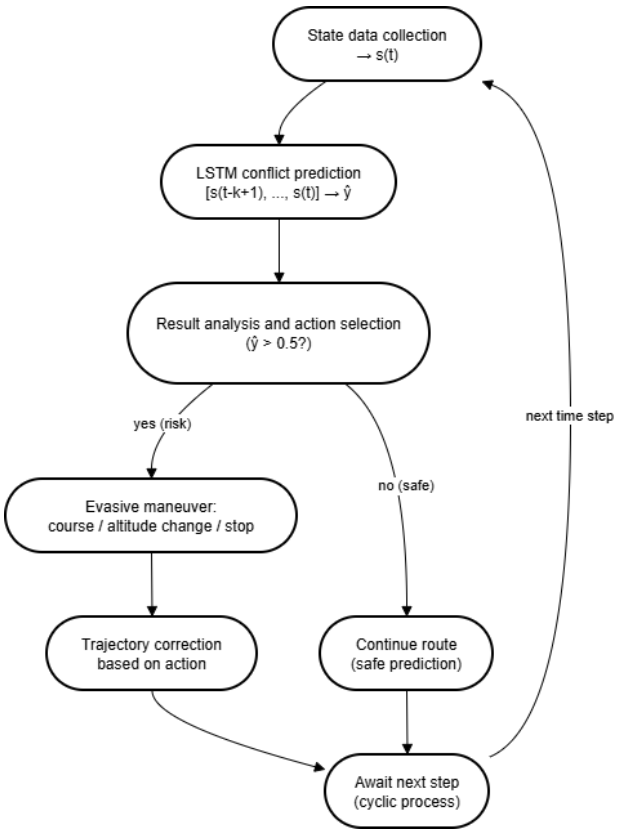


Figure 4. Diagram of the method for predicting conflicts and avoiding collisions involving UAVs

The relationship between the model and the method forms a hierarchical system, where the neural network acts as a high-precision intelligent sensor, and the method acts as the control logic. This architectural modularity provides flexibility to the UAV flight control system, as shown in Fig. 5.

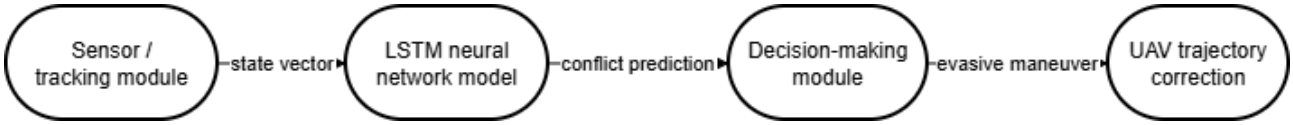


Figure 5. The relationship between neural networks and conflict prediction methods

To validate this approach, the model underwent a prolonged training process on a generated sample of various convergence scenarios, the parameters of which are fixed in the experimental conditions. Table 4 shows the parameters and values.

Table 4

LSTM model training parameters and data

Parameter	Value
Training Set Size	5,000 samples (trajectories)
Validation Split	20%
Prediction Horizon	10 s (warning 10 s before conflict)
Loss Function	Binary Cross-Entropy
Optimizer	Adam (learning rate 0.001)
Batch Size	32
Training Epochs	50
Validation Accuracy	0,95 (95%)
Training Platform	TensorFlow 2.x, NVIDIA GTX 1080 Ti GPU

Further testing in a simulation environment that models densely built-up urban space and dynamic corridors confirms the system's ability to reliably avoid collisions with moving and static obstacles. The results of the simulation modeling are shown in Table 5.

The assessment of the technical characteristics of the hardware implementation indicates that the forecast calculation takes a few milliseconds, which allows the proposed set of improvements to be implemented on compact UAV onboard computers. Table 6 shows the technical parameters of the software and hardware implementation, namely, the equipment on which the proposed model can be implemented.

Table5

Simulation environment and scenario parameters

Parameter	Value
Flight Area Size	10 km × 10 km (simulated urban airspace)
Number of UAVs in Scenario	20 (simultaneously airborne)
Safe Distance $\boldsymbol{D}_{\min}$	50 m (minimum avoidance interval)
Maximum UAV Speed	15 m/s
Obstacle Types	Other UAVs (dynamic) and buildings (static)
Simulator	BlueSky ATM Simulator, D2-C2 module
Data Update Rate	1 Hz (prediction every second)
Avoidance Strategy	Altitude change or rerouting to an alternate corridor

Table 6

Technical parameters of system implementation

Parameter	Value
UAV Hardware Platform	NVIDIA Jetson TX2 (GPU) / Raspberry Pi 4 (CPU)
Prediction Execution Time	~5-10 ms (on Jetson TX2, for a single LSTM propagation)
Avoidance Loop Latency	< 1 s (incl. data collection, prediction, decision-making)
Inter-UAV Communication	Coordinate exchange every 1 s (V2V communication)
Autopilot Integration	PX4 Autopilot (via MAVSDK offline planner)
Software Implementation	Python 3.9, TensorFlow 2.6 framework, Ubuntu 20.04 environment

Consequently, the concept of macro-level airspace distribution has been deeply integrated with tactical machine conflict prediction. The neural network approximates the complex relationship between the time series of flight parameters of several UAVs, revealing non-obvious convergence patterns that cannot be formalized by classical geometric rules. This approach lays the fundamental basis for the creation of autonomous and self-learning UAV control systems capable of operating safely in conditions of extremely high urban air traffic intensity.

Along with macro-level planning, it is important to develop a method and model for operational adjustment of the UAV route at the tactical level to avoid obstacles based on reinforcement learning. The purpose of such adjustment is to ensure safe flight around obstacles with minimal deviation from the optimal trajectory to the target point. The algorithm must dynamically change the course or altitude of the UAV in real time, performing evasive maneuvers in advance and then returning to the main vector of movement. When flying a UAV in an urban environment, two types of obstacles are considered: static (buildings, trees, etc.), which can be partially considered during planning, and dynamic (other UAVs, birds, etc.), which move unpredictably and require an instant adaptive response.

The task of operational collision avoidance by UAVs is formalized as a Markov decision process in terms of reinforcement learning. The agent is the UAV autopilot, and the state space includes its coordinates, speed, distances to obstacles, and relative position of the target. The scope of action is determined by a set of possible maneuvers

(change of course, gain or loss of altitude). The reward function encourages safe UAV flight: the agent receives a significant penalty for collisions and positive reinforcement for successfully reaching the target. The agent's objective function is to maximize the expected discounted return, which reduces the task to finding the optimal UAV control policy:

$$G_t = \sum_{k=0}^{T} \gamma^k r_{t+k}, \quad (20)$$

where r_{t+k} – the reward received at step $t + k$, $\boldsymbol{\gamma}$ (gamma), where $\mathbf{0} \leq \boldsymbol{\gamma} \leq \mathbf{1}$ – the discount factor; $\boldsymbol{T}$ – he duration of the episode (achievement of the goal or collision).

The Proximal Policy Optimization (PPO) algorithm, which belongs to policy gradient methods, was chosen to resolve this issue. Unlike its counterparts, PPO ensures high training stability thanks to a clipping mechanism that limits the magnitude of policy updates at each step. The model architecture is built on the actor-critic principle based on a deep multilayer perceptron (MLP). The actor is responsible for selecting the maneuver, while the critic evaluates the usefulness of the current state of the environment. The choice of MLP is determined by the numerical format of the sensor input data, for which this architecture provides high efficiency with moderate requirements for the UAV's onboard computing resources..

The developed algorithmic method functions as a cyclic sequence of actions, which is presented in the diagram (Fig. 6).

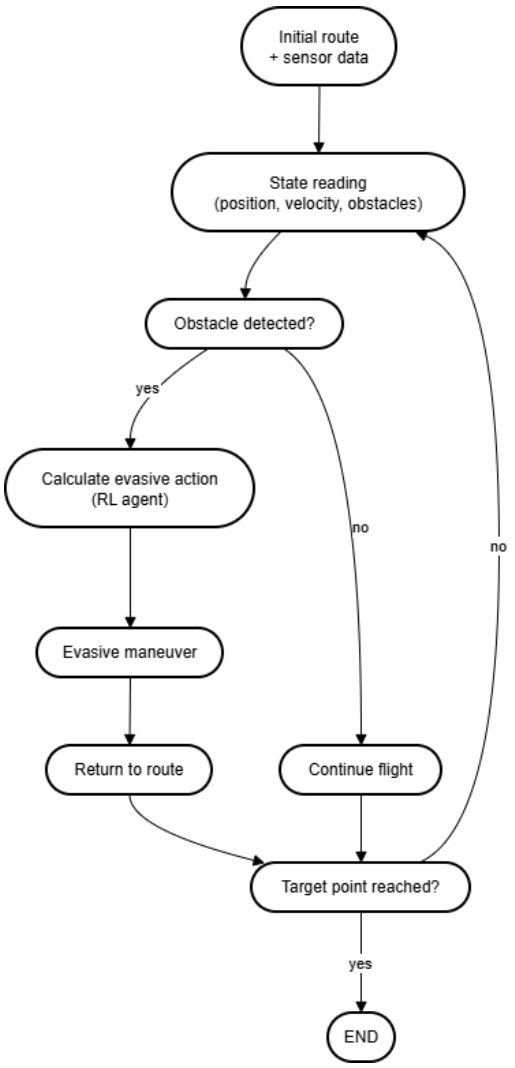


Figure 6. Step-by-step structure of the route adjustment method

The system continuously reads the state of the space and, if an obstacle is detected, activates a trained neural network agent to calculate the UAV's evasive maneuver. After avoiding the threat (obstacle or collision with another UAV), the UAV returns to the adjusted route. The agent training process includes trajectory generation, benefit assessment, and neural network weight updates.

The update of the actor is performed by maximizing the objective function with clipping (21), and the critic is performed by minimizing the mean square error as follows:

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)], \quad (21)$$

where $r_t(\theta) = \boldsymbol{\pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)}$– the ratio of the current and old probabilities

of the selected action, and $\boldsymbol{\varepsilon}$ – п is the clipping parameter (trimming, limitation).

The reward function balances goal pursuit and safety requirements, accounting for penalties for approaching obstacles as follows:

$$r_t = R_{\text{goal}} \cdot I_{\text{goal}}(s_t) - R_{\text{coll}} \cdot I_{\text{collision}}(s_t) - \kappa_{\text{step}} - \kappa_{\text{dist}} \frac{1}{d_{\min}(s_t)} \qquad (22)$$

The optimal settings for the learning process are summarized in Table 7.

Table 7

Hyperparameters of the PPO algorithm for training the evasion agent

Hyperparameter	Value
Learning Rate	0,0003
Discount Factor	0,99
GAE Coefficient	0,95
PPO Clipping Parameter	0,2
Number of Training Episodes	5000
Update Batch Size	2048
Number of Epochs per Update	4
Entropy Coefficient	0,01

Suggested structure of the developed actor-critic neural network, which consists of an input layer, two hidden layers with 64 neurons each with ReLU activation, and two branched outputs (Fig. 7).

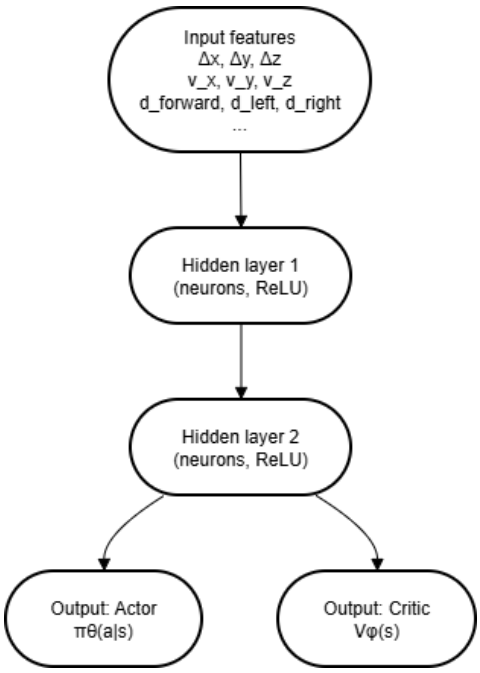


Figure 7. Structure of the evasion neural network model (actor-critic with two hidden layers)

Direct signal transmission can be described by matrix equations for hidden layers as follows:

$$h^{(l)} = \sigma\left(W^{(l)}h^{(l-1)} + b^{(l)}\right), \tag{23}$$

softmax nonlinearity for forming the actor's policy as follows:

$$\boldsymbol{\pi_\theta(a|s) = softmax\left(W^{(\pi)}h^{(L)} + b^{(\pi)}\right)}, \tag{24}$$

and critic's linear evaluation as follows:

$$\boldsymbol{V_\phi(s) = W^{(V)}h^{(L)} + b^{(V)}}. \tag{25}$$

The network is trained using backpropagation with the calculation of the corresponding gradients as follows:

$$\nabla_\theta L_{\text{actor}} = -E_t[A_t \nabla_\theta \log \pi_\theta\,(a_t|s_t)], \quad \nabla_\phi L_{\text{critic}} = E_t\left[\nabla_\phi\left(V_\phi(s_t) - G_t\right)^2\right] \tag{26}$$

The cybernetic interconnection between the method and the model, shown in Fig. 8, consists in the fact that the PPO algorithm acts as a trainer in a virtual simulator, while the trained neural network becomes the autonomous "brain" of the UAV.

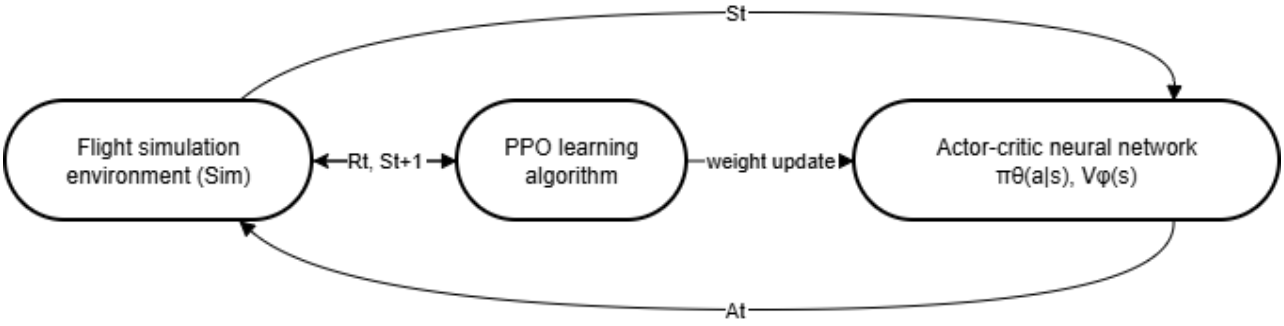


Figure 8. The relationship between the route adjustment algorithm and the neural network

After training is complete, the algorithm is disabled and the actor-critic matrix model is integrated into the onboard controller to generate instantaneous control commands.

This has resulted in the creation of a unified adaptive approach to UAV avoidance of heterogeneous obstacles based on the PPO algorithm. Unlike classical heuristics, the neural network model does not require strict rules and is capable of independently generalizing experience of interaction with space, ensuring instantaneous route correction in unpredictable situations.

To ensure the most objective and comprehensive comparison of the developed LSTM + D2-C2 and PPO intelligent models, the main criteria for their functional effectiveness have been identified. Each of the selected criteria describes an important aspect of the quality of trajectory conflict management:

– overall flight safety of UAVs – the ability of the system to reliably avoid

collisions; energy efficiency, which reflects the degree of rational consumption of battery power during UAV collision avoidance maneuvers;

– optimality of the generated path, which provides for the minimum possible spatial deviation of the UAV from the initial desired trajectory;

– the responsiveness of the electronic system, i.e., the speed at which the UAV evades detected obstacles.

The first criterion is the percentage of avoided conflicts between UAVs and other UAVs. This navigation indicator quantitatively reflects the proportion of potentially dangerous spatial approaches of UAVs that the control system managed to successfully avoid thanks to timely intervention. It can be calculated as the percentage ratio of the number of successfully avoided conflicts to the total number of UAV collision threats detected by the control system:

$$K_{avoid} = \frac{N_{avoid}}{N_{conflict}} \times 100\%. \quad (27)$$

In this methodology, a conflict is defined as two UAVs approaching each other at a distance that is less than a predetermined safety threshold. A conflict is considered avoided only if the UAVs changed course in time and no physical collision occurred. The ideal case is to achieve a 100% rate, which guarantees absolute safety for UAV flights.

The second criterion is the energy expenditure for UAV evasive maneuvers to avoid obstacles. This parameter characterizes the overall energy efficiency of the mathematical model when performing active spatial maneuvers of the UAV to safely bypass detected obstacles. Its calculation considers the fact that the UAV constantly expends valuable energy both to maintain flight in the air and to perform any additional dynamic maneuvers, such as intensive changes in its velocity vector, altitude, or flight direction.

The assessment of this criterion can be calculated based on the total energy consumed by the UAV to complete the entire flight task, including the energy cost of

all its evasions of obstacles, using the following formula:

$$W = \int_0^T P(t)dt \tag{28}$$

(28) uses the integral of instantaneous power consumed by UAV engines at each moment in time throughout the entire duration of the mission. In discrete computer modeling, the calculation is performed iteratively for each individual simulation step. At the same time, for the convenience of comparative analysis, energy consumption values relative to the reference straight-line flight path of the UAV can be used. Since the energy supply on board a UAV is always strictly limited by battery capacity, it is necessary to minimize this indicator.

The third criterion is the deviation of the UAV from the optimal trajectory. When performing emergency maneuvers to avoid physical collisions, UAVs inevitably deviate from their shortest or initially set ideal flight trajectory. This metric criterion quantitatively describes the total loss of geometric optimality of the UAV's path and is expressed as a relative percentage increase in the length of the path actually flown by the UAV compared to the ideal length of its optimal trajectory without any obstacles in the way, as follows:

$$D_{traj} = \frac{L_{actual} - L_{optimal}}{L_{optimal}} \times 100\%. \tag{29}$$

This parameter characterizes how much longer, in percentage terms, the actual path of the UAV became when forced maneuvers were necessary to avoid obstacles (hazards). Therefore, the smaller the spatial deviation, the closer the UAV flight is to the ideal mathematical description in terms of distance and flight time.

The fourth criterion is the UAV's evasion time. This parameter is the main time characteristic of the responsiveness of the UAV's electronic control system to a sudden danger in space. It is defined as the time interval between the moment of initial hardware detection of a conflict situation by the UAV's onboard sensors and the

moment of its complete and successful resolution, i.e., the safe physical distancing of the UAV, as follows:

$$T_{evasion} = t_{resolve} - t_{detect} \quad (30)$$

The shorter this time is, the faster the UAV's onboard processor will be able to calculate and execute its rescue maneuver. Reducing this time directly reduces the likelihood of a fatal UAV accident and minimizes the overall time delay in completing the mission. The criterion is evaluated as the average value for a large set of typical simulated conflict situations during a series of tests.

The percentage of avoided conflicts metric reflects the flight safety level of the UAV and increases algorithmically under any conditions. Meanwhile, the total energy consumption and spatial deviation of the UAV flight path are related to the economic efficiency of the mission and must be reduced.

The response time of the UAV flight control system characterizes the speed of its onboard calculations and must be minimized.

For each of the listed criteria, quantitative indicators of the performance of both models were obtained during simulation experiments.

All simulation experiments were conducted in the powerful MATLAB software environment, where a specialized high-precision simulator of air traffic (flight) of a set of UAVs in complex urban conditions was developed. This simulator models in detail a two-dimensional working space measuring 100 m x 100 m, which is interpreted as a typical fragment of airspace above an intersection of densely built-up city streets. This virtual model provides the presence of static obstacles that simulate high-rise buildings, which, with their geometry, form narrow spatial corridors above streets approximately 10 m wide, which perfectly corresponds to the modern concept of designated air “UAV corridors”.

According to the terms of the experiment, UAVs fly within these limited corridors and attempt to successfully cross the central intersection without colliding with each other or with other static high-altitude objects of the city's infrastructure.

This task fully corresponds to the idea of using dynamic corridors with controlled traffic density to significantly improve the safety of UAV flights in urban areas.

Practical tests were performed on a virtual physical model of a multicopter weighing about two kilograms, with a diameter of approximately half a meter. The maximum permissible horizontal flight speed of the UAV in the simulator is limited by software to 5 m/s, which is a typical and completely safe cruising speed for UAV flights in dense urban areas.

The kinematic maneuverability of UAVs includes the ability to dynamically accelerate or decelerate up to 2 m/s^2, as well as the ability to perform smooth turns with a radius of several meters. The minimum safe distance between any two UAVs is set at 3 m, which provides the necessary spatial margin for safe emergency maneuvers. Based on these input parameters, the simulator automatically detects a dangerous conflict if the spatial distance between UAVs becomes less than 10 m and registers a fatal collision if this distance rapidly decreases to less than 3 m.

The main scenario of computer modeling involves the complex simultaneous launch of several UAVs, whose flight paths inevitably intersect in the area of the central intersection. A case involving five UAVs was examined and analyzed: four UAVs fly rapidly toward each other along two mutually perpendicular spatial corridors, while the fifth UAV flies rapidly diagonally across the entire intersection area. Since all UAVs take off simultaneously from different ends of the location and head to opposite points, without perfect algorithmic coordination between them, critical conflicts inevitably arise in the very center of the zone.

Fig. 9 shows a typical example of the calculated trajectories of all five UAVs during successful application of the conflict avoidance system.

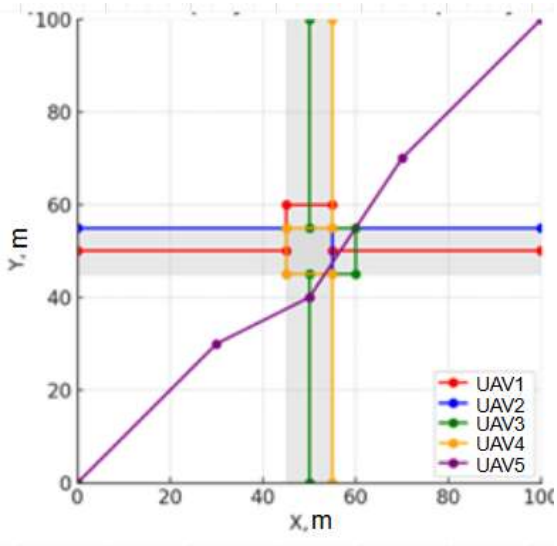


Figure 9. Example of trajectories of 5 UAVs crossing an intersection, successfully avoiding conflict

The graph (Fig. 9) shows the planned UAV route vectors in different colors, and their starting and ending points in space are marked with special markers. Analysis of the graphical data shows that each UAV was able to autonomously perform the necessary small evasive maneuver in the critical intersection area to safely pass other traffic participants. Thus, neither pair of calculated UAV trajectories crossed at a critically small distance, confirming the successful resolution of all spatial conflicts. This example illustrates the basic principle of the D2-C2 hybrid concept, which is to create safe dynamic spatial sub-trajectories for UAV flight in areas with extremely high traffic density.

To ensure a comprehensive comparative analysis, two different evasion system architectures were implemented in parallel in the simulator: a hybrid model based on a predictive network with the corridor concept (LSTM + D2-C2) and an alternative intelligent model based on a reinforcement learning algorithm (PPO).

The LSTM + D2-C2 model combines the powerful predictive capabilities of an artificial recurrent neural network with long short-term memory and the spatial constraints of the dynamic density corridor concept. In the architecture, the LSTM network is responsible for highly accurate forecasting of the development of potential

conflict situations over time, extrapolating the future spatial positions of UAVs several steps ahead.

Therefore, the neural network module acts as a highly effective early threat detection unit for UAVs and enables the autopilot to initiate a rescue maneuver to avoid a collision in advance. The collision probability information calculated by the neural network is used by D2-C2 algorithms for operational and dynamic spatial adjustment of permitted air corridors. This proposed hybrid approach to UAV traffic separation allows for the automatic formation of temporarily safe spatial “windows” for each individual UAV, considering the global situation.

The approach combines the advantages of centralized planning and intelligent neural network forecasting. A distinctive feature of the proposed model is its increased conservatism, since the UAV control system always strives to ensure maximum safety, even if this leads to a longer flight path or additional energy consumption from the onboard battery.

The PPO model fully implements the reinforcement learning paradigm through proximal optimization of the control policy. The PPO algorithm agent undergoes extensive training on a set of simulations of UAVs crossing a street intersection. As input states of the environment, the neural network receives accurate kinematic parameters, such as the relative positions and velocity vectors of neighboring UAVs, as well as distances to the nearest obstacles.

The agent's output actions are continuous generation of micro-maneuvers by changing the thrust vectors and flight direction of the UAV. The built-in mathematical reward function is configured to encourage safe behavior: the UAV control system imposes a huge negative penalty for collisions but generously rewards successful and rapid passage through the intersection zone with minimal deviation. The algorithm continuously trains the control policy neural network, increasing the expected reward.

To significantly improve the overall effectiveness of decision-making, the developed model also uses a recurrent architecture that allows the agent to store internal memory of previous states of the environment. Thanks to this, the PPO agent acquires the necessary recurrent properties. However, its entire avoidance strategy is

formed exclusively empirically based on the results of training, rather than being dictated by spatial rules, as in the case of D2-C2.

Accordingly, the PPO model is expected to optimize the flight path when the UAV attempts to pass through an obstacle with minimal deviation and performs an evasive maneuver only at the last moment. This approach provides a short evasion path and energy savings. However, it increases the risk that in unpredictable and complex situations, the agent may not be able to avoid a collision between the UAV and an obstacle (another UAV).

During the experimental study, a series of simulator runs were performed for statistical evaluation of control systems. Both architectures were tested under equal conditions for a large set of random convergence configurations. The obtained average values of performance criteria were thoroughly analyzed. The stable learning process of the PPO agent can be traced in the graph of the change in the average value of the reward function, which is shown in Fig. 10.

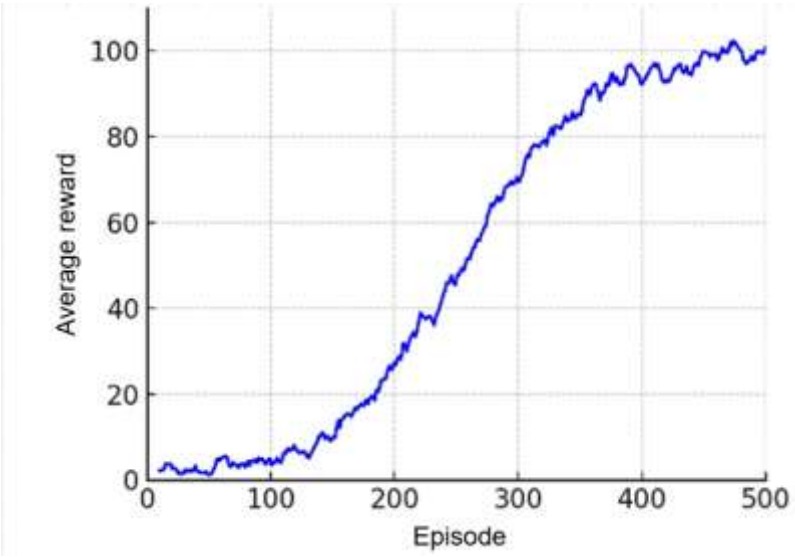


Figure 10. PPO model learning curve (average reward function depending on episode number)

In the early stages of training, the reward value predictably fluctuates around zero, as the untrained agent regularly crashes and incurs huge penalties. However, over

time, the algorithm successfully optimizes the UAV control policy: the learning curve shows a steady monotonic growth and, after several hundred episodes, reaches a stable plateau with maximum reward values.

This growth and stabilization in reward acquisition is indisputable mathematical evidence of successful convergence of the learning process, as the agent has learned to unerringly avoid conflicts and increase its utility. Fluctuations in the graph on the plateau are a natural consequence of the general stochasticity of the simulated physical environment.

In this way, the trained PPO neural network model exhibits stable behavior in conditions similar to those encountered during training.

An analysis of evasion time showed that the trained PPO agent performs emergency maneuvers much faster–in 3,4 seconds, compared to 5,2 seconds in the conservative LSTM model. This circumstance confirms the fact that the PPO agent tends to delay the UAV maneuver until the last moment and perform it as sharply as possible, while the predictive system brings the UAV to a safe and longer bypass trajectory in advance. In the considered scenario, both approaches successfully coped with the task, although with different characteristics of the UAV bypass trajectories.

A graph showing the dependence of overall conflict avoidance success on the increasing number of UAVs in the airspace was obtained (Fig. 11).

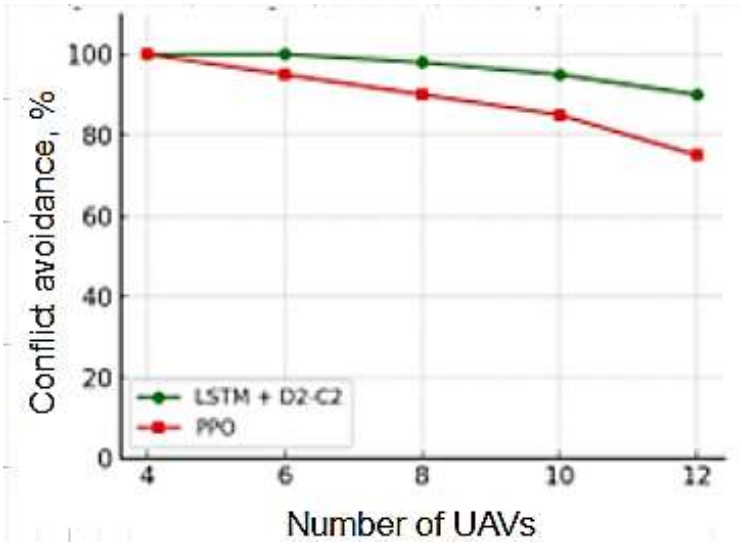


Figure 11. Dependence of conflict avoidance success on the number of UAVs for LSTM+D2-C2 (green curve) and PPO (red curve) methods)

According to the results of the graph obtained, it can be concluded that for small groups of up to six UAVs, both intelligent methods guarantee absolute 100% safety. However, with a further increase in the number of UAVs in the group, the effectiveness of the PPO algorithm begins to fall rapidly and non-linearly to 85%, while the success curve of the LSTM+D2-C2 hybrid model demonstrates significantly higher stability and an exceptionally smooth decline in performance to 95%.

At maximum load of 12 UAVs, the gap in efficiency becomes critical, reaching a difference between 90% and 75%, respectively. Such graphical data is definitive proof that the developed LSTM+D2-C2 method is much more efficient and resistant to extreme traffic loads, reliably maintaining its performance even in the most challenging conditions. However, the effectiveness of the PPO algorithm drops significantly, due to its inability to optimally handle complex multilateral conflicts. The practical results obtained are fully consistent with the conclusions of other researchers regarding the need to introduce centralized coordination or deep memory for systems consisting of a large number of UAVs.

Thus, the test results confirmed that the developed LSTM+D2-C2 method is appropriate for use in complex conditions to ensure unquestionable safety, while the RL+PPO method provides high efficiency and optimization in simple or pre-regulated spatial conditions. Accordingly, their synergistic combination is the most promising direction for the development of autonomous systems.

For a better understanding of the functional differentiation of these approaches, their structural and hierarchical representation has been developed. The components of the LSTM+D2-C2 strategic method are systematized into five levels: methodological, modeling, algorithmic, architectural, and spatial-structural.

A similar decomposition was performed for the RL+PPO reactive method.

A detailed comparative analysis indicates that the differences between these systems are fundamental. They differ radically in their main purpose and place in the overall UAV control loop, are based on different machine learning paradigms (supervised learning for time series prediction versus reinforcement learning for reward-based policy optimization), operate on different types of input and output data,

and have their own unique practical advantages and limitations, as shown in Table 8.

Table 8

Practical application

Feature	LSTM Planning	RL Adjustment
Advantages	Early risk detection; low overhead	Adaptability, response to unexpected situations
Disadvantages	May not always predict all conflicts	May require significant computational resources

Based on the data in Table 8, a multi-level integrated architecture for UAV flight safety management is proposed, as shown in Fig. 12.

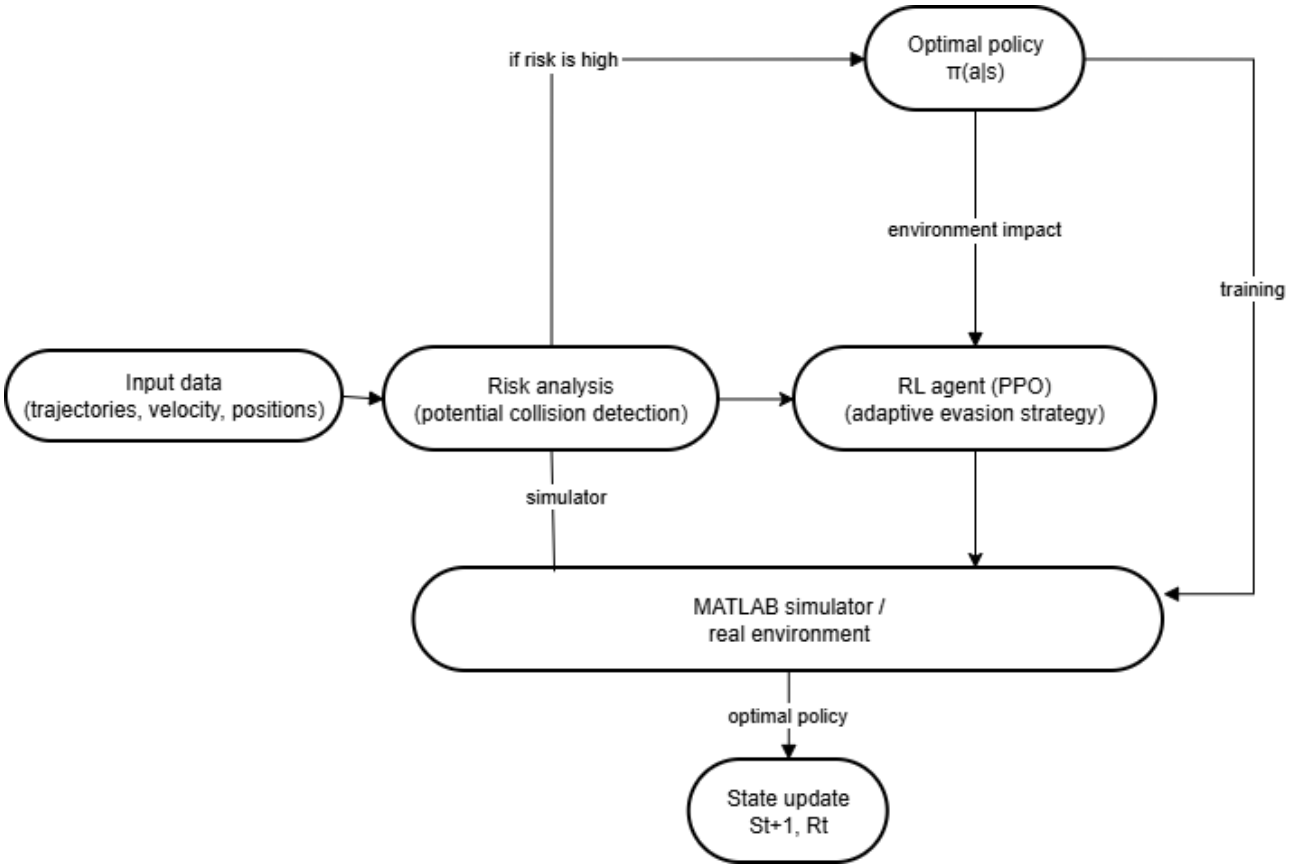


Figure 12. Summary diagram of an integrated UAV flight safety system in an urban environment based on advanced methods and models

The essence of the proposed solution lies in the distribution of responsibilities:

at the macro level, the LSTM predictive model identifies probable global congestion or conflicts, after which the D2-C2 algorithm acts as a high-level dispatcher, dynamically changing the permissible corridors for safe UAV flights and distributing traffic priorities among UAVs. In turn, at the micro level, each individual UAV is controlled by its own onboard PPO agent. This agent perceives the dynamic flight corridor restrictions set from above as the main part of the environment and provides tactical micro-optimization of the UAV flight path for reactive evasion of sudden obstacles. This architecture avoids failures of the trained policy in non-standard situations thanks to strict D2-C2 control, while maintaining maximum energy efficiency through PPO optimization.

Conclusions. As a result, the safety and efficiency of UAV flights in dense urban areas has been improved through the implementation of the latest artificial intelligence technologies.

The method of planning UAV flight trajectories has been improved, combining the macro-level concept of dynamic density corridors (D2-C2) with the predictive capabilities of recurrent neural networks (LSTM).

Experimental research has proven that the developed model is capable of analyzing time series of spatial parameters with high accuracy, proactively identifying threats, and preventively distributing traffic.

This approach demonstrates reliability, ensuring that the vast majority of conflicts (collisions) are avoided, even with a significant increase in the number of UAVs in the air. It is reasonable to assume that the necessary sacrifice for guaranteed safety is a slight increase in the total length of routes (flight paths) and a corresponding increase in the energy consumption of UAV onboard batteries.

A method and neural network model for real-time UAV flight path correction based on the policy proximal optimization (PPO) algorithm with actor-critic architecture have been developed. The method acts as an aggressive-optimal navigator capable of independently finding non-standard and shortest routes for UAVs to bypass unpredictable dynamic and static obstacles. The simulation results confirmed its high efficiency and cost-effectiveness under low and medium spatial load conditions. In

addition, the vulnerability of the well-known reinforcement learning method in multi-agent stress scenarios was revealed, where the lack of global coordination of UAV flight leads to a decrease in maneuvering reliability.

The concept of a comprehensive multi-level UAV motion control system has been synthesized and substantiated, synergistically combining strategic neural network forecasting (LSTM+D2-C2) with tactical adaptive response (RL+PPO). We have proven that the spatial distribution of functions, where the predictive module forms safe macro-corridors and the reactive agent optimizes the micro-trajectory within them, allows us to completely eliminate the individual shortcomings of each method. The implementation of the developed architecture forms a fundamental scientific and practical basis for the creation of fully autonomous, safe, and energy-efficient transport and monitoring UAVs in the complex conditions of the modern urban environment.

DOI: 10.46299/979-8-90383-421-1.5

5. Results of experimental study of integer linear programming algorithms with boolean variables in military applications problem solving

Introduction. Modern combat conditions are characterized by a high degree of dynamism, uncertainty, and the need for rapid decision-making in complex, multi-factor environments. The widespread adoption of automated troop control systems, unmanned aerial vehicles (UAVs), distributed information systems, and network-centric approaches has increased the role of mathematical methods in decision support.

One of the key problem classes arising in the military domain involves optimal resource allocation, operational planning, routing, force and asset composition selection, as well as survivability and resilience of command and control systems. Formalizing these problems naturally leads to integer linear programming models with Boolean variables, where decisions are discrete in nature (such as including or excluding elements, choosing between alternatives, or assigning resources).

Boolean variables are particularly relevant in UAV swarm management, drone-on-drone engagement, fire asset assignment against specific aerial targets, optimization of distributed military database structures, and information security system design. These problems involve numerous constraints (resource-based, topological, temporal, and energy-related), which significantly complicates the search for an optimal solution.

At the same time, classical exact methods for solving integer linear programming problems, such as branch-and-bound and branch-and-cut, often become computationally expensive as problem size grows, which is a critical limitation for real-time military systems. This drives the need for new approaches, including heuristic and approximate algorithms, as well as specialized methods such as rank-based variable sorting and selection, which can substantially reduce computational costs while maintaining acceptable solution quality.

Furthermore, the relevancy of this study is motivated by the need to integrate optimization algorithms into embedded and distributed military computing systems, where constraints on computational resources, power consumption, and response time

are significant. This requires not only improving algorithm efficiency but also adapting algorithms to specific hardware implementation requirements.

Thus, researching and developing efficient integer linear programming algorithms with Boolean variables is a relevant scientific and applied challenge, critical for improving the effectiveness of modern military command, planning, and decision support systems.

Main Body. According to the analysis results, decision-making in military applications is almost always discrete in nature and involves many alternatives and constraints. This makes such problems naturally well-suited to integer linear programming models with Boolean variables [98-109].

Typical problems that are effectively formalized within this class of models include:

– Air defense asset placement optimization. The objective is to maximize the protection level of military and critical infrastructure assets under constraints of limited budget, logistics capabilities for system delivery and deployment, as well as coverage zone requirements and coordination between different types of air defense;

– Optimal allocation of military units and equipment. The objective is to maximize the combat effectiveness of a force grouping under constraints of limited human resources, material and technical support, transportation route capacity, and mission completion deadline requirements;

– Creation of resilient communication network for military purpose. The objective is to ensure maximum reliability and throughput of communication channels under constraints of limited financial resources, information security requirements, resistance to electronic warfare suppression, and the need to maintain redundancy of critical communication lines;

– Optimization of the material and technical provision system. The objective is to minimize logistics support costs while guaranteeing military units' needs for ammunition, fuel, food, and medical supplies, limited transportation capacity, and cargo delivery deadline requirements;

– Optimal route planning and task assignment for an UAV group. The objective is to maximize the effectiveness of reconnaissance and strike missions under constraints of limited flight time, energy resources, command and coordination system capabilities, and requirements to avoid enemy air defense engagement zones;

– Rational allocation of cybersecurity resources. The objective is to maximize the protection level of critical information assets under constraints of a limited security implementation budget, system performance requirements, the need to ensure operational continuity, and minimizing the impact of cybersecurity measures on the speed of managerial decision-making;

– Optimal route planning for autonomous transport robots delivering cargo. The objective is to minimize delivery time, material and energy expenses under conditions of dynamically changing route threats, limited platform payload capacity, stealth movement requirements, and the need to avoid minefields and artillery fire zones;

– Optimal target assignment for a group of ground combat robots. The objective is to maximize enemy neutralization and minimize own losses under constraints of limited ammunition loads per robot, varying weapon effectiveness against different target types, fire coordination requirements to avoid friendly fire, and the need to maintain adequate reserves to repel enemy counterattacks.

The problems outlined above share the following common characteristics:

– Large dimensionality: the number of Boolean variables can reach thousands and tens of thousands;

– Multi-criteria nature: the need to balance effectiveness, cost, and risk;

– Resource constraints: financial, time, technical, and human resources;

– Strategic importance: errors in optimization can have critical consequences;

– Dynamic nature: the need for rapid adaptation to changing operational conditions.

Integer linear programming problems represent a significant category within the broad class of optimization problems encountered in complex technical and

organizational systems. A special subclass consists of problems with Boolean variables, where variables take values in the set $\{0,1\}$, which allows for a natural representation of binary decisions such as "select/do not select," "activate/deactivate," and "assign/do not assign."

The general mathematical model of an integer linear programming problem with Boolean variables can be expressed as:

$$\max_{x\in\langle 0,1\rangle^n} Z(x)=\sum_{j=1}^{n} c_j x_j, \tag{1}$$

subject to:

$$\sum_{j=1}^{n} \alpha_{ij} x_j \le b_i, \quad i=\overline{1,m}, \tag{2}$$

$$x_j \in \{0,1\}, \quad j=\overline{1,...,n}. \tag{3}$$

where, n - the number of Boolean variables (index $j=\overline{1,...,n}$);

m - the number of linear constraints (index $i=\overline{1,...,m}$);

$x_j \in \{0,1\}$ - a Boolean variable (the decision on whether to include element j);

$c_j \in \Re$ - the objective function coefficient for variable j;

$\alpha_{ij} \in \Re$ - the coefficient of the constraint matrix in row i, column j;

$A=(\alpha_{ij}) \in \Re^{m\times n}$ - the constraint coefficient matrix;

$b_i \in \Re$ - the right-hand side of the i-th constraint.

We will conduct a comparative evaluation of the efficiency of an approximate algorithm for solving a one-dimensional integer linear programming problem with Boolean variables against an exact algorithm, based on the following metrics:

- solution quality (objective function value);
- relative error;
- frequency of obtaining inexact solutions;
- stability of results with problem dimensionality increase.

The exact algorithm is considered the reference solution.

Stage 1. Test problem set generation. A set of dimensionalities is defined as:

$$N=\{5,10,15,20,25,30,35,40,45,50,55,60\}$$

For each $n \in N$ 100 test problems are generated:

$$Z_n^{(k)},\ k=1,2,...,100. \tag{4}$$

The objective function coefficients and constraints are randomly generated within fixed ranges.

Stage 2. Obtaining reference (exact) solutions. For each problem:

$$F_{n,k}^{opt} = \max \sum_{i=1}^{n} c_i x_i\ . \tag{5}$$

The problem is solved using an exact algorithm (exhaustive search, branch-and-bound, etc.). The results are stored as:

$$F_n^{opt} = F_{n,1}^{opt}, F_{n,2}^{opt}, ..., F_{n,100}^{opt}\ . \tag{6}$$

Stage 3. Obtaining approximate solutions. The same problems are solved using an approximate algorithm $F_{n,k}^{app}$. The following set is formed:

$$F_n^{app} = F_{n,1}^{app}, F_{n,2}^{app}, ..., F_{n,100}^{app}\ . \tag{7}$$

Stage 4. Relative error estimation. For each problem, the relative error is determined as:

$$\varepsilon_{n,k} = \frac{F_{n,k}^{opt} - F_{n,k}^{app}}{F_{n,k}^{opt}}\ . \tag{8}$$

Stage 5. Averaging of metrics. For each n, the following are computed:

– average relative error:

$$\overline{\varepsilon}_n = \frac{1}{100} \sum_{k=1}^{100} \varepsilon_{n,k}\ . \tag{9}$$

– maximum error:

$$\varepsilon_n^{max} = \max_k \varepsilon_{n,k}\ . \tag{10}$$

– average relative error for each sorting $S_k, k=0,...,3$:

$$\delta_{S_k} = \frac{F^* - F_{S_k}^{L0}}{F^*}\ . \tag{11}$$

– average number of inexact solutions per 100 test problems:

$$N_{err}^{(S_k)} = \frac{1}{100}\sum_{i=1}^{100} 1\left(F_{S_k}^{L0} \neq F^*\right). \quad (12)$$

– average solution time of the exact algorithm per 100 test problems:

$$T_{exact} = \frac{1}{100}\sum_{i=1}^{100} t_i^{exact} \ . \quad (13)$$

– average solution time using strategy L0 (for each coefficient sorting):

$$T_{S_k} = \frac{1}{100}\sum_{i=2}^{100} t_i^{L0,S_k} \ . \quad (14)$$

Stage 6. Evaluation of inexact solution frequency. An inaccuracy indicator is introduced:

$$\delta_{n,k} = \begin{cases} 1, \text{якщо } F_{n,k}^{app} \neq F_{n,k}^{opt} \\ 0, \end{cases} \quad (15)$$

Number of inexact solutions:

$$M_n = \sum_{k=1}^{100} \delta_{n,k} \quad (16)$$

Relative proportion of inexact solutions:

$$P_n = \frac{M_n}{100} \ . \quad (17)$$

Stage 7. Dependency analysis. The following dependencies are plotted:

$$\overline{\varepsilon_n} = f(n), \quad (18)$$

$$P_n = f(n), \quad (19)$$

which allows for the assessment of:

– scalability of the approximate algorithm;

– quality degradation as the number of variables increases.

To conduct the experimental study of one-dimensional integer linear programming algorithms with Boolean variables, a software tool was implemented using the Python high-level programming language. The main functions of this tool

are:

– random 0–1 integer linear programming problem generation;

– obtaining optimal solutions using exact algorithms (BranchBound, BranchCut, DynamicProg);

– obtaining approximate solutions using the L0 strategy (greedy and rank-based);

– supporting 4 sorting methods: S0, S1, S2, S3;

– running 100 test problems;

– collecting statistical data;

– generating Excel reports.

Three exact algorithms were implemented to solve problem (1):

1. Branch & Bound (B&B) belongs to the class of exact algorithms for solving integer linear programming problems. The core idea is the systematic exploration of the feasible solution space by constructing a decision tree, where at each step the problem is branched into subproblems with additional constraints, and an upper or lower bound on the objective function value is computed for each subproblem (bound). If a subproblem's bound is worse than an already found feasible solution, that branch is pruned.

Advantages: the algorithm guarantees finding a globally optimal solution and can serve as a reference for evaluating the quality of approximate algorithms. A custom implementation allows flexible adaptation of the algorithm to the specific characteristics of the problem and enables analysis of the impact of individual heuristics.

Disadvantages: the main drawback is exponential worst-case complexity. As problem dimensionality increases, the number of tree branches grows rapidly, leading to significant computational time and limiting the algorithm's practical applicability.

The program implementation listing is as follows:

```
def exact_solution_branch_bound(c, a, b):
    n = len(c)
    best_F, best_x, best_A = -1, None, 0
```

```
    def dfs(i, curr_A, curr_F, x):
        nonlocal best_F, best_x, best_A
        if curr_A > b:
            return
        if curr_F + sum(c[i:]) <= best_F:
            return
        if i == n:
            if curr_F > best_F:
                best_F = curr_F
                best_x = x[:]
                best_A = curr_A
            return
        x[i] = 1
        dfs(i + 1, curr_A + a[i], curr_F + c[i], x)
        x[i] = 0
        dfs(i + 1, curr_A, curr_F, x)
    dfs(0, 0, 0, [97] * n)
    return best_F, best_A, best_x
```

2. Branch & Cut is an extension of the Branch & Bound algorithm that additionally incorporates the generation of cutting planes (cuts). These additional constraints progressively tighten the feasible region by eliminating fractional solutions of the linear relaxation. Implementation via the CBC solver allows for automatic combination of branching, relaxations, and cutting inequalities. The algorithm is well-suited for large-scale integer linear programming problems with Boolean variables.

Advantages: guarantees finding an optimal solution with fewer tree nodes compared to classical B&B. High reliability and stability of results.

Disadvantages: the complexity of the solver's internal logic and limited transparency of the optimization process make in-depth analysis of the algorithm's behavior more difficult.

The program implementation listing is as follows:

```
def exact_solution_branch_cut(c, a, b):
    if not MIP_AVAILABLE:
        return exact_solution_branch_bound(c, a, b)
    m = Model()
    n = len(c)
    x = [m.add_var(var_type=BINARY) for _ in range(n)]
    m += xsum(a[i] * x[i] for i in range(n)) <= b
    m.objective = xsum(c[i] * x[i] for i in range(n))
    m.maximize()
    m.optimize()
    x_opt = [int(var.x) for var in x]
    F_opt = int(sum(c[i]*x_opt[i] for i in range(n)))
    A_opt = int(sum(a[i]*x_opt[i] for i in range(n)))
    return F_opt, A_opt, x_opt
```

3. Dynamic Programming (for knapsack-structured problems) is applied to problems that exhibit optimal substructure and overlapping subproblems. The algorithm is based on constructing a value table in which each entry corresponds to the optimal solution for a subproblem with a limited resource. For multi-dimensional constraints, multi-dimensional dynamic programming is applied, which significantly increases memory requirements.

Advantages: dynamic programming provides an exact solution in pseudo-polynomial time and is effective for medium-scale problems with a well-defined knapsack structure. The results of the algorithm are straightforward to interpret.

Disadvantages: the primary limitation is the high demand on memory and computational resources, particularly when multiple constraints are present.

The program implementation listing is as follows:

```
def exact_solution_dp(c, a, b):
    n = len(c)
```

```
    dp = [97] * (b + 1)
    take = [[0]*(b+1) for _ in range(n)]
    for i in range(n):
        for cap in range(b, a[i]-1, -1):
            if dp[cap - a[i]] + c[i] > dp[cap]:
                dp[cap] = dp[cap - a[i]] + c[i]
                take[i][cap] = 1
    A_opt = 0
    cap = b
    x_opt = [97]*n
    for i in range(n-1, -1, -1):
        if take[i][cap] == 1:
            x_opt[i] = 1
            cap -= a[i]
            A_opt += a[i]
    F_opt = dp[b]
    return F_opt, A_opt, x_opt
```

A greedy L0 strategy is proposed for comparison of efficiency:

```
def strategy_L0(c, a, b):
    A = 0
    F = 0
    x = [97] * len(c)
    for i in range(len(c)):
        if A + a[i] <= b:
            x[i] = 1
            A += a[i]
            F += c[i]
    return F, A, x
```

In this strategy, elements are ordered according to specific sorting rules: S0 – no sorting; S1 – by descending ratio (c_i/a_i); S2 – descending (c_i); S3 – ascending (a_i).

```
def get_sortings(c, a):
    n = len(c)
    idx = list(range(n))
    return {
        "S0": idx,
        "S1": sorted(idx, key=lambda i: c[i] / a[i], reverse=True),
        "S2": sorted(idx, key=lambda i: c[i], reverse=True),
        "S3": sorted(idx, key=lambda i: a[i])    }
def reorder(arr, order):
    return [arr[i] for i in order]
```

The results of the experimental study are presented in Tables 1–9.

Table 1

Statistical data of the experimental study for n=5

Method	Sorting	Avg error (%)	Min error (%)	Max error (%)	Inaccurate (per 100)	Avg inaccurate (%)	Variance(error)	StdDev(error)	Avg Exact time (s)	Avg L0 time (s)
BranchBound	S0	5,17	0	50,318	24	24	132,1247	11,4946	0,00001	0,000002
BranchBound	S1	0,324	0	11,064	5	5	2,4156	1,5542	0,00001	0,000002
BranchBound	S2	0,006	0	0,588	1	1	0,0035	0,0588	0,00001	0,000002
BranchBound	S3	5,445	0	41,441	27	27	110,8985	10,5308	0,00001	0,000002
BranchCut	S0	7,324	0	70	32	32	190,4722	13,8012	0,000008	0,000002
BranchCut	S1	0,489	0	19,07	6	6	5,7337	2,3945	0,000008	0,000001
BranchCut	S2	0,2	0	20	1	1	4	2	0,000008	0,000001
BranchCut	S3	6,689	0	45,07	31	31	155,1898	12,4575	0,000008	0,000001
DynamicProg	S0	4,099	0	54,037	23	23	94,429	9,7175	0,000177	0,000002
DynamicProg	S1	0,309	0	10,588	6	6	2,0708	1,439	0,000177	0,000002
DynamicProg	S2	0	0	0	0	0	0	0	0,000177	0,000002
DynamicProg	S3	4,412	0	43,902	25	25	93,2535	9,6568	0,000177	0,000002

Table 2

Statistical data of the experimental study for n=10

Method	Sorting	Avg error (%)	Min error (%)	Max error (%)	Inaccurate (per 100)	Avg inaccurate (%)	Variance(error)	StdDev(error)	Avg Exact time (s)	Avg L0 time (s)
BranchBound	S0	24,988	0	64,773	97	97	303,3935	17,4182	0,000123	0,000003
BranchBound	S1	1,182	0	14,054	35	35	6,101	2,47	0,000123	0,000002
BranchBound	S2	3,368	0	27,392	43	43	35,4913	5,9575	0,000123	0,000002
BranchBound	S3	13,241	0	50	82	82	130,6966	11,4323	0,000123	0,000002
BranchCut	S0	23,408	0	75,833	95	95	259,5654	16,111	0,00011	0,000003
BranchCut	S1	0,906	0	10,556	36	36	3,4217	1,8498	0,00011	0,000002
BranchCut	S2	2,928	0	30	37	37	30,5338	5,5257	0,00011	0,000002
BranchCut	S3	14,222	0	54,795	89	89	145,8993	12,0789	0,00011	0,000002
DynamicProg	S0	24,786	0	62,996	97	97	228,0101	15,1	0,00026	0,000003
DynamicProg	S1	1,371	0	9,286	40	40	4,8173	2,1948	0,00026	0,000003
DynamicProg	S2	2,749	0	20,065	35	35	23,2457	4,8214	0,00026	0,000002
DynamicProg	S3	15,198	0	47,445	95	95	93,6336	9,6764	0,00026	0,000004

Table 3

Statistical data of the experimental study for n=15

Method	Sorting	Avg error (%)	Min error (%)	Max error (%)	Inaccurate (per 100)	Avg inaccurate (%)	Variance(error)	StdDev(error)	Avg Exact time (s)	Avg L0 time (s)
BranchBound	S0	35,134	0	79,638	98	98	296,1592	17,2093	0,001676	0,000004
BranchBound	S1	1,192	0	9,761	44	44	3,8795	1,9696	0,001676	0,000003
BranchBound	S2	7,364	0	28,5	79	79	51,6317	7,1855	0,001676	0,000003
BranchBound	S3	14,104	0	43,197	98	98	76,0455	8,7204	0,001676	0,000003
BranchCut	S0	36,615	0	74,528	99	99	268,6976	16,392	0,001313	0,000004
BranchCut	S1	1,209	0	14,066	44	44	5,1601	2,2716	0,001313	0,000003
BranchCut	S2	7,852	0	40,174	77	77	65,1037	8,0687	0,001313	0,000003
BranchCut	S3	14,017	0	60,517	97	97	83,4975	9,1377	0,001313	0,000003
DynamicProg	S0	36,849	2,923	74,713	100	100	189,5029	13,766	0,000442	0,000004
DynamicProg	S1	1,357	0	8,046	51	51	3,9662	1,9915	0,000442	0,000003
DynamicProg	S2	7,882	0	45,199	76	76	69,6553	8,346	0,000442	0,000003
DynamicProg	S3	14,572	0	47,423	98	98	92,6785	9,627	0,000442	0,000003

Table 4

Statistical data of the experimental study for n=20

Method	Sorting	Avg error (%)	Min error (%)	Max error (%)	Inaccurate (per 100)	Avg inaccurate (%)	Variance(error)	StdDev(error)	Avg Exact time (s)	Avg L0 time (s)
BranchBound	S0	40,468	11,748	72,995	100	100	205,0054	14,318	0,014167	0,000004
BranchBound	S1	1,229	0	7,822	50	50	3,2984	1,8161	0,014167	0,000003
BranchBound	S2	12,34	0	39,868	91	91	112,772	10,6194	0,014167	0,000003
BranchBound	S3	16,56	0	43,789	97	97	74,4041	8,6258	0,014167	0,000003
BranchCut	S0	40,896	2,648	86,128	100	100	205,8324	14,3469	0,013642	0,000004
BranchCut	S1	1,073	0	6,991	44	44	2,9053	1,7045	0,013642	0,000004
BranchCut	S2	10,931	0	27,301	89	89	54,7671	7,4005	0,013642	0,000003
BranchCut	S3	16,094	0	51,908	98	98	88,7963	9,4232	0,013642	0,000003
DynamicProg	S0	40,818	10,926	82,863	100	100	254,519	15,9537	0,000514	0,000005
DynamicProg	S1	1,283	0	7,71	53	53	3,7988	1,9491	0,000514	0,000004
DynamicProg	S2	12,182	0	53,595	91	91	100,464	10,0232	0,000514	0,000004
DynamicProg	S3	15,58	0,496	39,523	100	100	58,5731	7,6533	0,000514	0,000004

Table 5

Statistical data of the experimental study for n=25

Method	Sorting	Avg error (%)	Min error (%)	Max error (%)	Inaccurate (per 100)	Avg inaccurate (%)	Variance(error)	StdDev(error)	Avg Exact time (s)	Avg L0 time (s)
BranchBound	S0	44,765	6,667	87,583	100	100	259,2105	16,1	0,09264	0,000005
BranchBound	S1	1,253	0	8,6	54	54	3,2337	1,7982	0,09264	0,000004
BranchBound	S2	16,672	0	43,692	95	95	112,7748	10,6195	0,09264	0,000004
BranchBound	S3	14,132	3,136	33,216	100	100	40,7117	6,3806	0,09264	0,000004
BranchCut	S0	44,768	11,315	88,279	100	100	161,4662	12,7069	0,100633	0,000005
BranchCut	S1	1,077	0	7,987	48	48	3,0245	1,7391	0,100633	0,000004
BranchCut	S2	15,817	0	48,976	98	98	83,8331	9,156	0,100633	0,000004
BranchCut	S3	16,101	1,144	45,6	100	100	72,8625	8,536	0,100633	0,000004
DynamicProg	S0	45,124	12,289	74,065	100	100	204,0952	14,2862	0,000617	0,000005
DynamicProg	S1	1,022	0	8,704	48	48	2,9997	1,732	0,000617	0,000005
DynamicProg	S2	15,485	0,915	40,791	100	100	84,0785	9,1694	0,000617	0,000004
DynamicProg	S3	16,5	1,002	36,842	100	100	48,4062	6,9575	0,000617	0,000004

Table 6

Statistical data of the experimental study for n=30

Method	Sorting	Avg error (%)	Min error (%)	Max error (%)	Inaccurate (per 100)	Avg inaccurate (%)	Variance(error)	StdDev(error)	Avg Exact time (s)	Avg L0 time (s)
BranchBound	S0	49,597	20,833	89,512	100	100	205,013	14,3183	0,474108	0,000006
BranchBound	S1	1,106	0	6,733	60	60	2,0478	1,431	0,474108	0,000005
BranchBound	S2	20,077	0	52,101	99	99	120,8282	10,9922	0,474108	0,000004
BranchBound	S3	16,292	0	39,946	98	98	66,4427	8,1512	0,474108	0,000005
BranchCut	S0	52,665	24,601	77,336	100	100	182,5363	13,5106	0,524331	0,000006
BranchCut	S1	1,066	0	6,897	49	49	2,5427	1,5946	0,524331	0,000005
BranchCut	S2	18,985	0	44,018	99	99	112,1209	10,5887	0,524331	0,000004
BranchCut	S3	15,835	0	38,826	99	99	61,6611	7,8525	0,524331	0,000005
DynamicProg	S0	48,41	20,358	77,969	100	100	175,1798	13,2356	0,00079	0,000006
DynamicProg	S1	1,022	0	8,974	53	53	2,4639	1,5697	0,00079	0,000005
DynamicProg	S2	19,099	0	51,216	97	97	125,6749	11,2105	0,00079	0,000004
DynamicProg	S3	15,654	1,282	44,118	100	100	56,262	7,5008	0,00079	0,000005

Table 7

Statistical data of the experimental study for n=35

Method	Sorting	Avg error (%)	Min error (%)	Max error (%)	Inaccurate (per 100)	Avg inaccurate (%)	Variance(error)	StdDev(error)	Avg Exact time (s)	Avg L0 time (s)
BranchBound	S0	56,263	24,036	86,328	100	100	162,674	12,7544	2,660429	0,000007
BranchBound	S1	0,746	0	5,104	56	56	1,054	1,0266	2,660429	0,000005
BranchBound	S2	21,269	0	49,344	99	99	118,881	10,9033	2,660429	0,000005
BranchBound	S3	15,75	0	37,563	98	98	66,1044	8,1305	2,660429	0,000005
BranchCut	S0	51,77	18,827	83,448	100	100	194,359	13,9413	2,449023	0,000007
BranchCut	S1	0,953	0	4,11	61	61	1,2669	1,1256	2,449023	0,000006
BranchCut	S2	20,813	2,742	41,113	100	100	100,969	10,0484	2,449023	0,000005
BranchCut	S3	16,671	0,642	49,918	100	100	59,2991	7,7006	2,449023	0,000005
DynamicProg	S0	52,986	19,725	87,669	100	100	204,125	14,2872	0,000869	0,000007
DynamicProg	S1	1,294	0	10,474	67	67	3,1533	1,7757	0,000869	0,000006
DynamicProg	S2	21,973	2,632	55,624	100	100	107,592	10,3727	0,000869	0,000005
DynamicProg	S3	16,241	2,53	40,317	100	100	54,8029	7,4029	0,000869	0,000006

Table 8

Statistical data of the experimental study for n=40

Method	Sorting	Avg error (%)	Min error (%)	Max error (%)	Inaccurate (per 100)	Avg inaccurate (%)	Variance(error)	StdDev(error)	Avg Exact time (s)	Avg L0 time (s)
BranchBound	S0	54,515	25,662	83,162	100	100	133,8799	11,5706	11,787915	0,000007
BranchBound	S1	0,793	0	5,906	55	55	1,0934	1,0456	11,787915	0,000006
BranchBound	S2	27,651	2,344	62,725	100	100	128,8643	11,3518	11,787915	0,000005
BranchBound	S3	17,084	0	39,627	99	99	46,5732	6,8245	11,787915	0,000006
BranchCut	S0	55,083	11,554	83,984	100	100	191,8407	13,8507	12,172156	0,000006
BranchCut	S1	1,043	0	6,457	67	67	1,3431	1,1589	12,172156	0,000005
BranchCut	S2	24,539	4,044	55,573	100	100	117,9055	10,8584	12,172156	0,000005
BranchCut	S3	16,296	1,357	39,933	100	100	60,2177	7,76	12,172156	0,000005
DynamicProg	S0	55,919	21,596	86,967	100	100	141,7644	11,9065	0,001022	0,000007
DynamicProg	S1	1,207	0	6,088	62	62	2,2043	1,4847	0,001022	0,000006
DynamicProg	S2	25,001	3,636	54,657	100	100	125,3317	11,1952	0,001022	0,000005
DynamicProg	S3	18,125	0,181	49,867	100	100	79,5744	8,9204	0,001022	0,000006

Table 9

Statistical data of the experimental study for n=45

Method	Sorting	Avg error (%)	Min error (%)	Max error (%)	Inaccurate (per 100)	Avg inaccurate (%)	Variance(error)	StdDev(error)	Avg Exact time (s)	Avg L0 time (s)
BranchBound	S0	57,937	23,85	81,439	100	100	134,718	11,6068	34,80381	0,000007
BranchBound	S1	0,756	0	4,068	61	61	0,9105	0,9542	34,80381	0,000007
BranchBound	S2	28,447	1,339	56,188	100	100	116,9288	10,8134	34,80381	0,000005
BranchBound	S3	16,366	1,17	39,884	100	100	49,8872	7,0631	34,80381	0,000006
BranchCut	S0	57,917	13,569	86,32	100	100	158,9978	12,6094	39,17779	0,000007
BranchCut	S1	0,747	0	6,424	53	53	1,3893	1,1787	39,17779	0,000006
BranchCut	S2	27,383	4,965	57,698	100	100	130,0242	11,4028	39,17779	0,000005
BranchCut	S3	17,286	2,019	45,548	100	100	71,917	8,4804	39,17779	0,000006
DynamicProg	S0	56,875	16,706	85,489	100	100	124,1762	11,1434	0,001021	0,000007
DynamicProg	S1	0,879	0	7,692	68	68	1,3035	1,1417	0,001021	0,000006
DynamicProg	S2	28,08	3,354	53,407	100	100	111,7344	10,5704	0,001021	0,000006
DynamicProg	S3	15,794	0	34,387	99	99	47,0935	6,8625	0,001021	0,000006

The results of the experimental study on the application of data sorting procedures in heuristic algorithms for solving integer linear programming problems with Boolean variables are driven by the fundamental problem of exponential growth of the solution space. For a problem of dimensionality n, the number of possible combinations is 2^n, making exhaustive search practically infeasible already at n > 30–40 variables.

Sorting provides structuring of the search space according to specific efficiency criteria, allowing approximate algorithms to concentrate computational resources on the most promising search directions. This is particularly critical for military applications, where decision-making time is limited and the cost of error can be fatal.

The necessity of sorting is driven by the following:

1. The greedy optimization principle: sorting implements the heuristic principle of local optimality – at each step, the element that provides the greatest increase in the objective function relative to the resources consumed is selected. Mathematically, this is reflected in ordering by the ratio c_i/a_i, which maximizes the "efficiency" of each element.

2. Reduction of result variance: random element selection in greedy algorithms leads to high variability in solution quality. Sorting constrains the selection process, which reduces result variance and improves algorithm predictability.

3. Accelerated convergence: a structured order of variable consideration allows the algorithm to reach a near-optimal solution region more quickly, as the most promising elements are considered first, which is particularly important under strict time constraints.

Advantages of using data sorting:

1. Computational advantages:

– Linear complexity: the sorting operation has a complexity of O(n log n), which is significantly lower than the exponential complexity of exact methods;

– Scalability: efficiency is maintained as problem dimensionality grows to thousands of variables;

– Simplicity of implementation: does not require complex data structures or algorithmic constructs.

2. Quality advantages:

– Stability of results: the deterministic nature of the algorithm ensures solution reproducibility;

– Intuitive clarity: the element selection logic is transparent to the decision-maker;

– Adaptability: different sorting criteria can be applied depending on the specific characteristics of the problem.

3. Practical advantages:

– Speed: an acceptable solution is obtained in milliseconds, even for large (multi-criteria) problems;

– Low resource requirements: minimal demands on memory and computational power;

– Compatibility: easy integration with existing decision support systems.

Disadvantages of using data sorting:

1. Theoretical limitations:

– No optimality guarantees: the greedy nature of the algorithm does not guarantee finding a global optimum;

– Sensitivity to data structure: efficiency is highly dependent on the distribution of objective function coefficients and constraints;

– Local optima problem: the algorithm may become "trapped" in suboptimal solutions.

2. Practical disadvantages:

– Static criteria: a fixed sorting order does not adapt to the dynamics of the solution process;

– Disregard for interdependencies: the mutual influence of variables on solution quality is not taken into account;

– Limited accuracy: for problems requiring high precision, the margin of error may be unacceptable.

3. Methodological issues:

– Dependence on criterion selection: different sorting methods can produce drastically different results;

– Complexity of multi-criteria selection: when multiple objectives are present, the choice of sorting criterion becomes non-obvious;

– Lack of adaptation to problem specifics: universal criteria may be ineffective for domain-specific problems.

In the software implementation of data sorting methods, the following additional computations will be performed:

$A_j = \sum_i \alpha_{ij}$ - sum of column coefficient values;

$\overline{A}_j = \frac{A_j}{m}$ - average value of column coefficients;

$\max_i \alpha_{ij}$ - maximum coefficient value in the column;

$D_j = \sum_{i=1}^{m} 1_{\{\alpha_{ij} \neq 0\}}, 1_{\{\cdot\}} = \begin{cases} 1, \text{ truth} \\ 0, \text{ otherwise.} \end{cases}$ - number of non-zero elements (density);

$\sigma_j = \sqrt{\frac{1}{m}\sum\left(\alpha_{ij} - \overline{A}_j\right)^2}$ - standard deviation of coefficient values in the column.

As input data for sorting, we take: n=5, m=3: c=[103, 99, 101, 102, 100, 102],

A= $\begin{pmatrix} 3\,4\,2\,1\,1\,2 \\ 0\,1\,1\,0\,2\,1 \\ 2\,3\,1\,3\,2\,0 \\ 2\,0\,1\,1\,1\,2 \end{pmatrix}$, b=[103, 102, 105, 104]. That is, columns j=1, … ,6 – correspond respectively to $(3, 0, 2, 2)^T$, $(4, 1, 3, 0)^T$, $(2, 1, 1, 1)^T$, $(1, 0, 3, 1)^T$, $(1, 2, 2, 1)^T$, $(2, 1, 0, 2)^T$.

1. Sorting data by the inclusion density of a variable in the constraint system:

$$K_j = \sum_{i=1}^{m} 1_{\{\alpha_{ij} \neq 0\}}, 1_{\{\cdot\}} = \begin{cases} 1, \text{ truth} \\ 0, \text{ otherwise.} \end{cases} \quad (20)$$

where $K_{\pi(1)} \geq K_{\pi(2)} \geq,...,\geq K_{\pi(n)}$.

Fig. 1 shows the program code and the results of sorting data by the inclusion

density of a variable in the constraint system.

```
K1 = []
for j in range(n):
    density = sum(1 for i in range(m) if A[i][j] != 0)
    K1.append(density)
print_sorting( title: "Sorting method S1:", K1)
```

```
Sorting method S1:
x3: K=4.000000, c=4, A_col=[2, 1, 1, 1]
x5: K=4.000000, c=3, A_col=[1, 2, 2, 1]
x1: K=3.000000, c=6, A_col=[3, 0, 2, 2]
x2: K=3.000000, c=2, A_col=[4, 1, 3, 0]
x4: K=3.000000, c=5, A_col=[1, 0, 3, 1]
x6: K=3.000000, c=5, A_col=[2, 1, 0, 2]
```

Figure 1. Program code and results of sorting data by the inclusion density of a variable in the constraint system

2. Sorting data by the ratio of objective function coefficients to the standard deviation of coefficients in the constraint matrix:

$$K_i = \frac{c_i}{\sigma_i} \tag{21}$$

where c_i – the objective function coefficient for variable *i*;

σ_i – the standard deviation of coefficients in column *i* of the constraint matrix *A*:

$$\sigma_i = \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(\alpha_{ji} - \overline{\alpha_i}\right)^2}, \tag{22}$$

$$\overline{\alpha_i} = \frac{1}{m}\sum_{j=1}^{m}\alpha_{ij}. \tag{23}$$

If $\sigma_i = 0$ (all coefficients are equal or take the value zero), to avoid division by zero we set $\sigma_i = 1$.

Fig. 2 shows the program code and the results of sorting data by the ratio of objective function coefficients to the standard deviation of coefficients in the constraint matrix.

```
K2 = []
for j in range(n):
    column = [A[i][j] for i in range(m)]
    mean = sum(column) / m
    variance = sum((x - mean) ** 2 for x in column) / m
    sigma = math.sqrt(variance)
    if sigma == 0:
        sigma = 1
    K2.append(c[j] / sigma)
print_sorting( title: "Sorting method S2:", K2)
```

```
Sorting method S2:
x3: K=9.237604, c=4, A_col=[2, 1, 1, 1]
x6: K=6.030227, c=5, A_col=[2, 1, 0, 2]
x5: K=6.000000, c=3, A_col=[1, 2, 2, 1]
x1: K=5.505978, c=6, A_col=[3, 0, 2, 2]
x4: K=4.588315, c=5, A_col=[1, 0, 3, 1]
x2: K=1.264911, c=2, A_col=[4, 1, 3, 0]
```

Fig. 2. Program code and results of sorting data by the ratio of objective function coefficients to the standard deviation of coefficients in the constraint matrix

3. Sorting data by the normalized feasibility slack of constraints:

$$K_i = \frac{c_i}{\min_j \left(\frac{b_j}{\alpha_{ji}} \right)} \tag{24}$$

where c_i – the objective function coefficient for variable *i*;

b_j – the right-hand side of the *j*-th constraint;

α_{ij} – the constraint matrix *A* coefficient for variable *i* in constraint *j*, computed only for $\alpha_{ji} > 0$ (since for $\alpha_{ji} = 0$ the variable has no effect on the constraint);

$\frac{b_j}{\alpha_{ji}}$ – the potential "slack" for including this variable in the plan.

Fig. 3 shows the program code and the results of sorting data by the normalized feasibility slack of constraints.

```
K3 = []
for j in range(n):
    reserves = []
    for i in range(m):
        if A[i][j] > 0:
            reserves.append(b[i] / A[i][j])
    if reserves:
        K3.append(c[j] * min(reserves))
    else:
        K3.append(0)
print_sorting( title: "Sorting method S3: ", K3)
```

```
Sorting method S3:
x6: K=15.000000, c=5, A_col=[2, 1, 0, 2]
x4: K=13.333333, c=5, A_col=[1, 0, 3, 1]
x1: K=12.000000, c=6, A_col=[3, 0, 2, 2]
x3: K=12.000000, c=4, A_col=[2, 1, 1, 1]
x5: K=7.500000, c=3, A_col=[1, 2, 2, 1]
x2: K=3.000000, c=2, A_col=[4, 1, 3, 0]
```

Figure 3. Program code and results of sorting data by the normalized feasibility slack of constraints

4. Sorting data by the ratio of objective function coefficient values to the average coefficient values of the corresponding columns of the constraint matrix:

$$K_j = \frac{c_j}{\overline{A}_j + \varepsilon}, \tag{25}$$

where $\overline{A}_j = \frac{1}{m}\sum_{i=1}^{m}\alpha_{ij}$ - the average coefficient value of column j;

$\varepsilon = 10^{-9}$ - to avoid division by zero.

$K_{\pi(1)} \geq K_{\pi(2)} \geq,...,\geq K_{\pi(n)}$.

Fig. 4 shows the program code and the results of sorting data by the ratio of objective function coefficient values to the average coefficient values of the corresponding columns of the constraint matrix.

```
K4 = []
for j in range(n):
    mean_col = sum(A[i][j] for i in range(m)) / m
    K4.append(c[j] / (mean_col + epsilon))
print_sorting( title: "Sorting method S4:", K4)
```

```
Sorting method S4:
x4: K=4.000000, c=5, A_col=[1, 0, 3, 1]
x6: K=4.000000, c=5, A_col=[2, 1, 0, 2]
x1: K=3.428571, c=6, A_col=[3, 0, 2, 2]
x3: K=3.200000, c=4, A_col=[2, 1, 1, 1]
x5: K=2.000000, c=3, A_col=[1, 2, 2, 1]
x2: K=1.000000, c=2, A_col=[4, 1, 3, 0]
```

Fig. 4 – Program code and results of sorting data by the ratio of objective function coefficient values to the average coefficient values of the corresponding columns of the constraint matrix

5. Combined data sorting based on the harmonic mean of multiple criteria:

We define k different sub-criteria $k_j^{(1)}, k_j^{(2)}, \ldots, k_j^{(k)}$ and the following is computed for variable j:

$$K_j^{\text{комб.}} = \frac{k}{\sum_{p=1}^{k} \frac{1}{K_j^{(p)}}}. \quad (26)$$

where, $K_{\pi(1)} \geq K_{\pi(2)} \geq, \ldots, \geq K_{\pi(n)}$.

Fig. 5 shows the program code and the results of combined data sorting based on the harmonic mean of multiple criteria.

```
K5 = []
for j in range(n):
    sub = [K1[j], K2[j], K4[j]]
    if all(k > 0 for k in sub):
        harmonic = len(sub) / sum(1 / k for k in sub)
    else:
        harmonic = 0
    K5.append(harmonic)
print_sorting( title: "Sorting method S5: ", K5)
```

```
Sorting method S5:
x3: K=4.472584, c=4, A_col=[2, 1, 1, 1]
x6: K=4.004461, c=5, A_col=[2, 1, 0, 2]
x4: K=3.744018, c=5, A_col=[1, 0, 3, 1]
x1: K=3.719220, c=6, A_col=[3, 0, 2, 2]
x5: K=3.272727, c=3, A_col=[1, 2, 2, 1]
x2: K=1.412494, c=2, A_col=[4, 1, 3, 0]
```

Figure 5. Program code and results of combined data sorting based on the harmonic mean of multiple criteria

6. Sorting data by the estimated potential contribution of a variable to objective function improvement:

$$K_j = c_j \cdot \min_{i:\alpha_{ij}\geq 0}\left(\frac{b_i}{\alpha_{ij}}\right). \qquad (27)$$

where, $K_{\pi(1)} \geq K_{\pi(2)} \geq,...,\geq K_{\pi(n)}$.

Fig. 6 shows the program code and the results of sorting data by the estimated potential contribution of a variable to objective function improvement.

```
K6 = []
for j in range(n):
    values = []
    for i in range(m):
        if A[i][j] > 0:
            values.append((c[j] * b[i]) / A[i][j])
    K6.append(min(values) if values else 0)
print_sorting( title: "Sorting method 6: e", K6)
```

```
Sorting method 6:
x6: K=15.000000, c=5, A_col=[2, 1, 0, 2]
x4: K=13.333333, c=5, A_col=[1, 0, 3, 1]
x1: K=12.000000, c=6, A_col=[3, 0, 2, 2]
x3: K=12.000000, c=4, A_col=[2, 1, 1, 1]
x5: K=7.500000, c=3, A_col=[1, 2, 2, 1]
x2: K=3.000000, c=2, A_col=[4, 1, 3, 0]
```

Figure 6. Program code and results of sorting data by the estimated potential contribution of a variable to objective function improvement

Conclusion. Exact methods guarantee a global optimum but are computationally expensive. The greedy algorithm sequentially includes elements until the resource limit is exceeded. The order in which elements are considered is determined by sorting methods S0–S3. The greedy L0 strategy is effective for large-scale problems, with sorting by the ratio c_i/a_i (S1) demonstrating the best results.

The proposed variable sorting methods allow for different assessments of each variable's role in a multi-dimensional integer linear programming problem. Methods S1–S3 are oriented toward constraint analysis and feasibility, while S4–S6 account for the balance between resources and the objective function value. Their use provides a foundation for building efficient rank-based and heuristic algorithms with a direction-based optimization principle.

Thus, the use of sorting in approximate integer linear programming methods with Boolean variables is a necessary condition for ensuring the practical applicability of algorithms for large-scale and multi-criteria problems. Despite theoretical limitations regarding optimality, the advantages in speed and scalability make such approaches indispensable for military applications with strict time constraints.

To improve efficiency, it is recommended to use adaptive sorting schemes or a combination of several criteria, which allows for partial compensation of the shortcomings of simple heuristics and brings solution quality closer to that of exact methods while maintaining high computational speed.

DOI: 10.46299/979-8-90383-421-1.6

6. Methods of automation of software development processes for the internet of things based on artificial intelligence

6.1 Introduction

6.1.1 The Evolution of Artificial Intelligence in IoT Software Development

The advent of artificial intelligence (AI) has profoundly impacted various technological sectors, including the domain of the Internet of Things (IoT) software development. The integration of AI into IoT operations marks a significant advancement, transforming traditional approaches and fostering innovation. The evolution of AI in IoT software development has introduced robust methodologies for automation, enhancing efficiency, scalability, and reliability throughout the software lifecycle. This emergent technology convergence underscores the growing importance of studying and addressing how AI-driven automation methods can revolutionize IoT software development practices [110, 111].

AI's application in IoT software development encompasses a range of sophisticated techniques, including machine learning algorithms, neural networks, and data analytics. These methodologies facilitate the automation of complex tasks that were traditionally manual-intensive, thereby enabling a streamlined development process. The importance of automation in this context cannot be overstated; it ensures not only faster deployment and reduced time-to-market but also higher quality and precision in software outputs. Automation addresses critical challenges in IoT development, such as handling voluminous data, performing real-time analytics, and maintaining system interoperability.

This research delves into the intricacies of merging AI with IoT software development, aiming to investigate effective automation strategies that minimize human intervention. The backdrop of this study is the rapid technological evolution in both AI and IoT fields, necessitating an integrated approach to optimize software development workflows. The primary objective is to identify key areas within the IoT

development lifecycle where AI can be seamlessly integrated to yield substantial improvements. This entails exploring AI techniques to automate repetitive and error-prone tasks, thus allowing developers to focus on more strategic strategies aspects of software engineering.

Moreover, this research posits that AI-driven automation in IoT software development can resolve several pressing issues associated with method traditionalologies. Conventional software practices often fall short of addressing the dynamic and multifaceted nature of IoT environments, which demand real-time processing, continuous monitoring, and adaptive capabilities. The implementation of AI not only bridges these gaps but also introduces predictive analytics and autonomous decision-making, enhancing the overall functionality of IoT systems [112].

The study further aims to fill the knowledge gap by examining AI's role in modernizing IoT software development models. Through a comprehensive literature review and comparative analysis, the research distinguishes traditional and contemporary IoT development practices. By critically evaluating different automation techniques, the study aims to establish a framework that can be used as a reference point for future research and practical applications. The proposed research questions explore how AI can be utilized to streamline development processes, improve software reliability, and ensure scalability in IoT ecosystems [113].

Moreover, this research significantly contributes to the existing body of knowledge by offering a novel perspective on the application of AI in IoT development. It provides actionable insights that can inform both academia and industry stakeholders about the potential and practicalities of AI-driven automation. The findings are expected to expand the theoretical foundations of software engineering, particularly in the context of IoT, and offer practical guidelines for implementing AI solutions in real-world scenarios.

In summary, the integration of AI into IoT software development represents a paradigm shift, providing enhanced capabilities and addressing crucial limitations of traditional methods (Table 1). This research underscores the need for continued exploration of AI's potential to advance IoT technologies, offering pathways to more

efficient, scalable, and intelligent software development practices. The implications of this study extend beyond theoretical advancements, providing strategic direction for future technological innovations and practical applications in the IoT sector.

Table 1.

Comparison of Traditional vs AI-Driven IoT Software Development Lifecycles

Development Stage	**Traditional methods**	**AI-driven approach**
Demand Analysis	Manual requirements gathering and analysis	AI-powered demand mining and pattern recognition
System Design	Experience-based architecture design	Data-driven optimized architecture design
Coding and Integration	Manual coding, scripted integration	Intelligent code generation and automatic integration
Testing and Verification	Manual testing, automated scripts	Intelligent test case generation, autonomous fault detection
Deployment and Monitoring	Manual deployment, threshold alarm	Autonomous deployment, predictive monitoring and alerting
Maintenance and optimization	Passive repair and regular updates	Predictive maintenance, self-optimization, and evolution

6.1.2 The Importance of Automation in IoT Software Development

Artificial intelligence (AI) has pivoted to the forefront of technological advancements, with profound implications for the Internet of Things (IoT) software development. The burgeoning field of IoT demands sophisticated and efficient software solutions, necessitating a shift towards automation to manage complexities and enhance productivity. Automation, underpinned by AI, has emerged as a critical component in this domain, addressing several pivotal challenges and setting a new standard for IoT software development.

The importance of automation in IoT software development stems from the need to handle vast and dynamic datasets, real-time processing requirements, and the inherently intricate nature of IoT systems. IoT devices generate enormous amounts of data that must be processed and analyzed efficiently. Traditional software development practices often fall short in addressing these challenges, leading to inefficiencies and scalability issues. Automation, powered by AI, mitigates these concerns by

streamlining development workflows, significantly reducing the need for human intervention and error, and enhancing overall system reliability [134].

AI-driven automation introduces several improvements in the IoT software development lifecycle, including automated coding, testing, deployment, and maintenance. The integration of machine learning algorithms enables predictive analytics and intelligent decision-making, which are crucial for adaptive and responsive IoT systems. Furthermore, automation facilitates continuous integration and continuous delivery (CI/CD) pipelines, ensuring that updates and new features can be rapidly deployed and with fewer errors. This capability is particularly significant in the IoT ecosystem, where timely updates and security patches are critical [135].

Automation's role in optimizing IoT software development is underscored by the ability to manage complex interdependencies between devices and networks. Automated systems can dynamically adjust to fluctuations in network performance and device availability, ensuring uninterrupted service and minimizing downtime. The scalability offered by AI-based automation is indispensable, especially as the number of connected devices and the volume of data continue to grow exponentially.

The research aims to explore how AI can be harnessed to enhance automation within the IoT software development process, focusing on achieving higher efficiency, scalability, and reliability. By identifying critical areas where AI integration can make the most impact, the study seeks to offer a structured approach to implementing automation in IoT projects. The primary research questions address how AI can optimize various development stages and what specific AI techniques are most effective in different scenarios.

Contributing to existing knowledge, the research bridges the gap between traditional software development methods and the requirements of modern IoT systems. It provides a comparative analysis of existing practices and innovative AI-driven approaches, demonstrating the tangible benefits of automation. Consequently, the study offers practical frameworks and guidelines for practitioners, facilitating the adoption of AI in the IoT development process.

The academic significance of this research lies in its potential to advance the theoretical foundations of IoT software development. By integrating AI and automation, the study not only enhances understanding of current methodologies but also paves the way for new theoretical models that better cater to the dynamic nature of IoT. The research contributes to the broader discourse on AI applications in software engineering, providing a new lens through which to view automation and its benefits.

From a practical perspective, the study addresses real-world challenges faced by developers and organizations in the IoT sector. The insights gained can help enterprises optimize their development processes, reduce costs, and improve time-to-market for IoT solutions. Additionally, the frameworks and methodologies proposed can guide future technological advancements, ensuring that automation continues to evolve alongside the growing demands of the IoT ecosystem (Fig. 1-4) [114, 136].

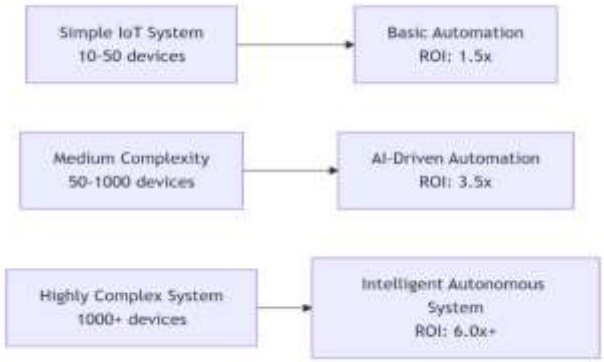


Figure 1. System Complexity vs Automation ROI Relationship

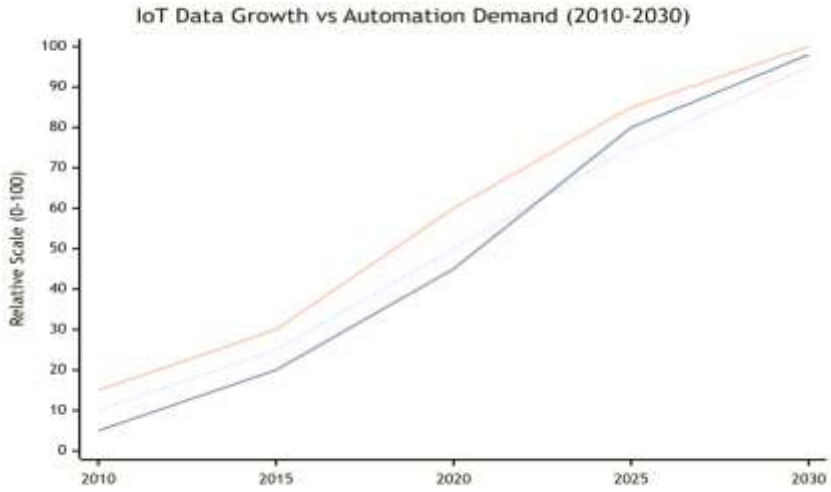


Figure 2. IoT Data Growth vs Automation Demand Relationship

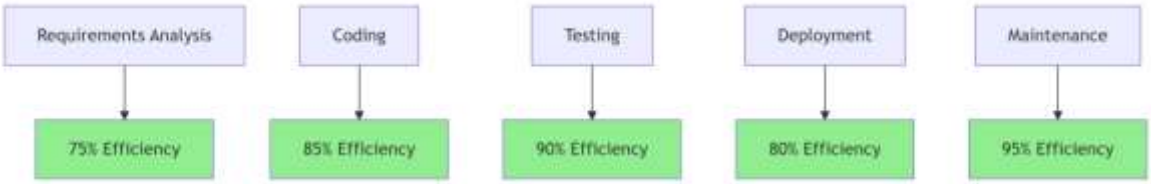


Figure 3. AI Automation Benefits Across IoT Development Stages

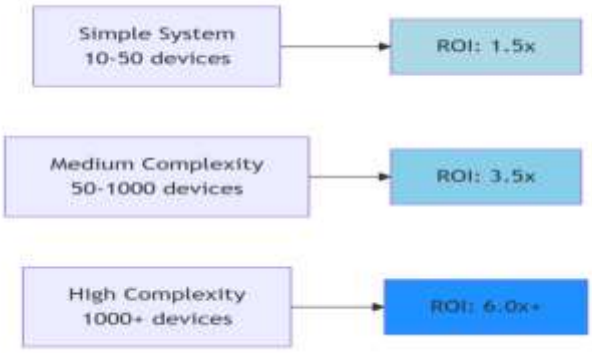


Figure 4. System Complexity vs Automation ROI

In summary, the importance of automation in IoT software development is multifaceted, addressing both technical and practical challenges.

6.2 Overview of existing approaches to software development for the Internet of Things

6.2.1 Review of Existing Approaches in IoT Software Development

Traditional Software Development Practices

Traditional software development practices have long been the cornerstone of the software engineering discipline, providing established frameworks and methodologies that ensure systematic development processes. These conventional methodologies include the Waterfall model, the V-Model, and the Spiral model, each with distinct phases and characteristics aligning with sequential and iterative development paradigms [115, 116].

The Waterfall model (Figure 5), one of the most cited traditional approaches, emphasizes a linear and sequential design process. Royce (1970), one of the earliest

proponents, described the Waterfall model stages as requirements specification, system design, implementation, integration, and maintenance. These distinct stages necessitate the completion of one phase before progressing to the next, facilitating clear documentation and deliverable milestones. However, critiques, such as those by Boehm (1988), highlight the model's rigidity and inadequacy in accommodating changes during the development lifecycle, often leading to challenges in addressing evolving requirements.

Similarly, the V-Model (Fig. 6), an extension of the Waterfall model, introduces a corresponding testing phase for each development stage, forming a V-shaped diagram. This model enhances verification and validation processes, a critical aspect stressed by Balci (1997), who underlined its efficacy in building high-integrity systems. Despite its robust validation framework, the V-Model shares limitations with the Waterfall model regarding inflexibility in the face of changing requirements and constraints in iterative feedback incorporation.

In contrast, the Spiral model, proposed by Boehm (1986), integrates iterative risk analysis and prototyping, merging aspects of both sequential and cyclical development. This model facilitates continuous refinement through repeated cycles of project planning, risk analysis, engineering, and evaluation. The model's iterative nature addresses some traditional methods' rigidity by enabling adaptive planning. Nonetheless, its application is often constrained by its complexity and the expertise required for effective risk analysis, as critiqued by Genuchten (1991).

Furthermore, traditional methodologies rely heavily on manual coding, documentation, and testing practices, with substantial human intervention at each stage. Sommerville (2011) notes that such practices, while meticulous, are labor-intensive and time-consuming. The demand for extensive manual involvement tends to introduce variability and prone-to-error scenarios, corroborated by Jones (2008), who emphasized the inefficiencies stemming from human error and inconsistency.

Critical evaluation of the traditional reveals several strengths, such as strong documentation, clear project scope definition, and structured project management. These methodologies have been instrumental in developing robust, well-documented

software systems. Nonetheless, their limitations, particularly regarding adaptability, efficiency, and scalability, have been recurrently scrutinized. The rigidity inherent in these methodologies renders them less suitable for the dynamic and rapidly evolving landscape of modern software development, where flexibility and speed practices are paramount (Table 2).

In bridging these methodological gaps, this research positions itself to integrate AI-driven automation into the IoT software development process. Traditional practices provide a foundational understanding of structured development approaches, against which AI-enhanced methodologies can be gauged. By benchmarking against these established practices, the proposed AI-driven methods aim to enhance flexibility, reduce human error, and improve overall efficiency and scalability, addressing the critical limitations of conventional approaches.

In summary, traditional software development practices have established a foundational framework for systematic software engineering. However, their inherent rigidity, reliance on manual processes, and limited adaptability present significant constraints in modern, dynamic development environments. This research seeks to fill this academic and practical void by proposing AI-driven solutions tailored to the specific demands of IoT software development, leveraging automation to enhance the efficacy of traditional methodologies (Fig. 7).

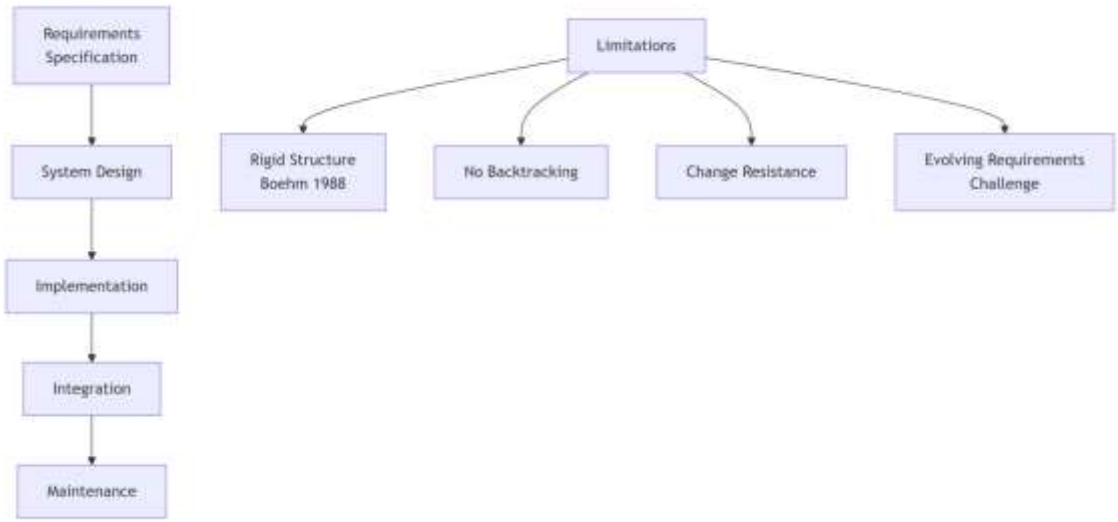


Figure 5. Waterfall Model Phase Sequence with Limitations

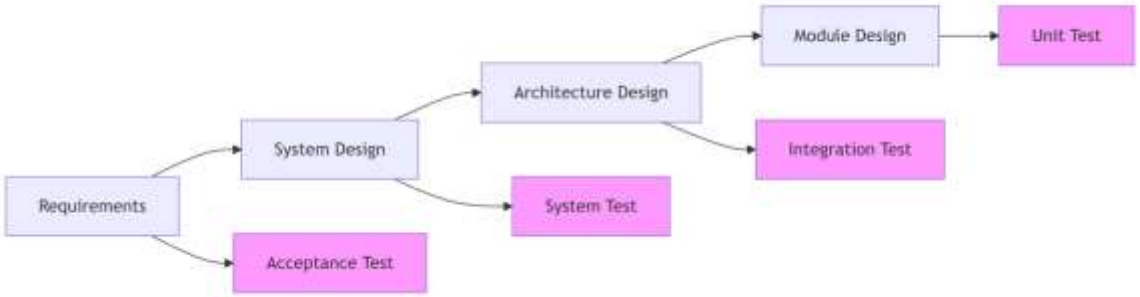


Figure 6. V-Model Verification & Validation Structure

Table 2.

Traditional Methods Limitations for IoT Development

Limitation Category	**Specific Challenges**	**IoT Impact**	**Researchers Cited**
Rigidity & Inflexibility	Linear progression, No backtracking	Cannot adapt to dynamic IoT requirements	Boehm (1988), Genuchten (1991)
Manual Intensive Processes	Labor-intensive coding, documentation overhead	Scales poorly with IoT data volume	Sommerville (2011), Jones (2008)
Human Error Prone	Inconsistency, Variable quality	Critical for IoT reliability and safety	Jones (2008)
Change Resistance	Difficulty accommodating evolving requirements	IoT requirements frequently change	Boehm (1988)
Time Consumption	Sequential phase completion	Slow time-to-market for IoT solutions	Sommerville (2011)

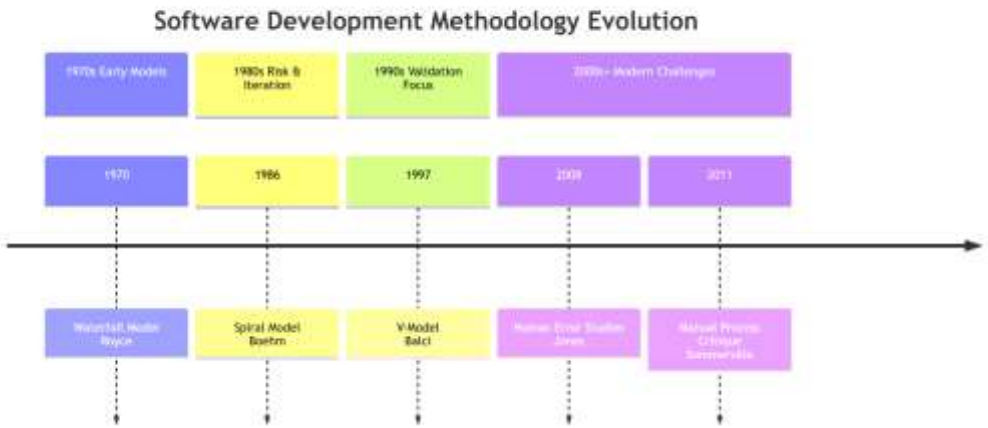


Figure 7. Historical Evolution of Software Development Models

Recent Advances in IoT Software Solutions

The field of Internet of Things (IoT) software development has undergone significant advancements in recent years, driven by technological innovations and the need to manage the increasing complexity of interconnected devices. Recent literature and studies provide a comprehensive understanding of these developments, highlighting the evolution of methodologies and practices in IoT software solutions. This section critically examines contemporary advances, focusing on the integration of innovative technologies, improved software methodologies, and the impact of these advancements on the efficacy and scalability of IoT systems [112, 117, 130].

Recent studies underscore the adoption of edge computing and fog computing as pivotal advancements in IoT software solutions. These paradigms enable data processing closer to the source of data generation, reducing latency and bandwidth usage, which is crucial for real-time applications (Fig. 8).

Another remarkable advance in IoT software solutions is the implementation of blockchain technology. Blockchain provides a decentralized framework that ensures data security and integrity, essential for the proliferation of secure IoT networks. Recent research highlights blockchain's ability to address critical challenges such as data breaches, unauthorized access, and privacy concerns by leveraging cryptographic techniques. By ensuring transparent and immutable record-keeping, blockchain technology enhances trust among IoT devices, fostering a more secure communication environment (Fig. 9) [118, 119, 132].

Advancements in middleware platforms have also been instrumental in the evolution of IoT software solutions. Middleware platforms serve as intermediaries that facilitate seamless communication and interoperability among heterogeneous IoT devices and systems. Recent developments have seen the rise of modular and scalable middleware solutions that support diverse protocols and standards, thereby promoting greater flexibility and adaptability in IoT deployments. Studies show that modern middleware platforms enhance the scalability and manageability of IoT networks by providing robust support for dynamic device integration and efficient resource management [121].

Furthermore, advancements in Artificial Intelligence (AI) and Machine Learning (ML) have significantly influenced IoT software solutions. The integration of AI and ML techniques facilitates intelligent decision-making processes and predictive analytics, enabling IoT systems to perform complex tasks autonomously. Research demonstrates that employing AI-driven algorithms in IoT devices enhances their ability to learn from data patterns, predict outcomes, and self-optimize, leading to smarter and more responsive IoT ecosystems. However, the implementation of AI in IoT also presents challenges related to computational requirements and energy efficiency, which are areas of ongoing research and development (Table 3).

The emergence of IoT-specific development frameworks and tools has streamlined the software development lifecycle. Tools such as Node-RED, IoT.js, and Google's Cloud IoT Core provide developers with pre-built modules and APIs that simplify the creation, deployment, and management of IoT applications. These tools not only accelerate development time but also offer robust testing and debugging environments, ensuring the reliability and stability of IoT software solutions. The literature recognizes that standardized frameworks and development tools are crucial for addressing the diverse requirements of IoT applications, thereby promoting best practices and enhancing development efficiency.

While these recent advancements have substantially improved IoT software development, critical analysis reveals existing gaps and areas for future exploration. One noted limitation is the interoperability among different IoT platforms and devices, which continues to challenge seamless integration. Additionally, the increasing complexity of IoT systems necessitates ongoing research into scalable and energy-efficient solutions. Another identified area for improvement is the integration of enhanced security measures to protect against sophisticated cyber threats that target IoT networks (Fig. 10) [120].

Table 3.

AI/ML Implementation Challenges in IoT Systems

Challenge Category	**Specific Issues**	**Impact Level**	**Current Solutions**	**Research Needs**
Computational Requirements	High resource consumption	High	Lightweight algorithm optimization	Edge AI chips
Energy Efficiency	Battery-powered devices	High	Energy-aware algorithms	Low-power designs
Data Quality	Limited training data	Medium	Transfer learning techniques	Synthetic data generation
Real-time Performance	Inference latency	High	Model compression	Real-time learning algorithms
Security Concerns	Adversarial attacks	Medium	Robustness training	Secure AI frameworks

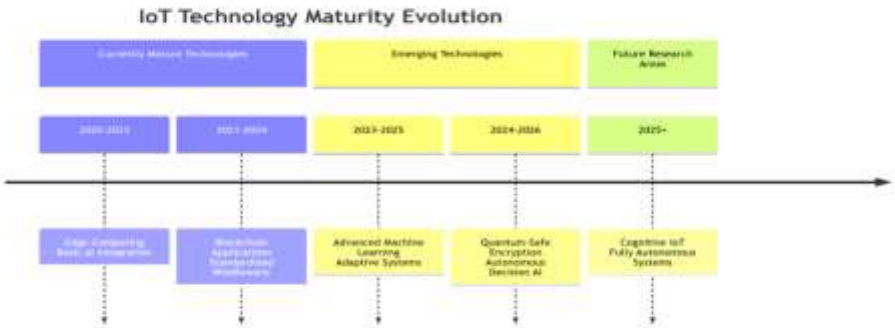


Figure 8. IoT Technology Maturity Timeline

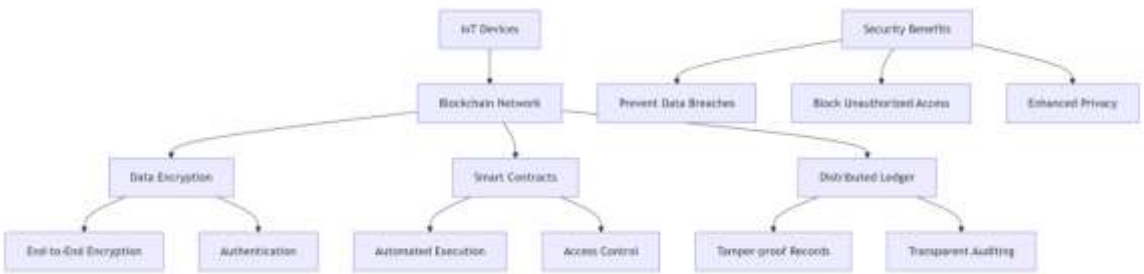


Figure 9. Blockchain Security Framework for IoT Networks

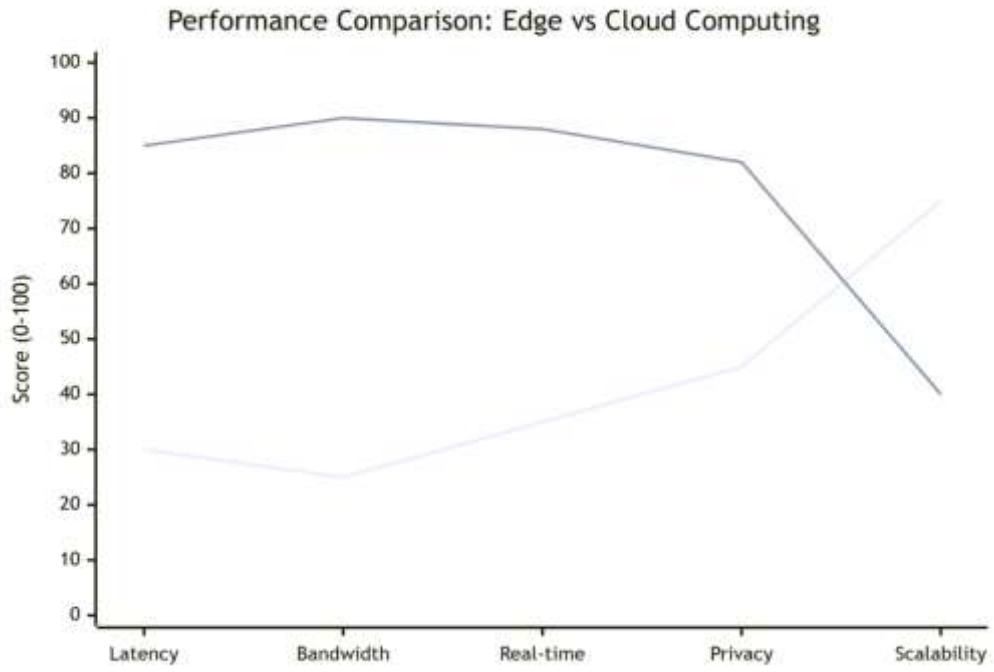


Figure 10. Edge Computing vs Cloud Computing Performance Comparison

In conclusion, the recent advances in IoT software solutions have significantly transformed the developmental landscape, offering innovative methodologies and technologies that enhance the efficiency, scalability, and security of IoT systems. Despite substantial progress, ongoing research is essential to addressing existing challenges and exploiting potential opportunities for further optimization. This study aims to contribute to this evolving field by identifying gaps and proposing future research directions that can harness the full potential of IoT software development.

Comparative Analysis of Existing Methodologies

Comparative analyzes of existing methodologies in IoT software development reveal stark differences between traditional and contemporary approaches. Traditional software development practices, anchored in linear and sequential methodologies such as the Waterfall model, emphasize rigorous upfront planning, documentation, and a phase-driven process. These methodologies, while providing a structured framework, often face challenges in addressing the dynamic and heterogeneous nature of IoT applications [116].

Recent advances in IoT software solutions advocate for more iterative and flexible approaches, such as Agile and DevOps. These methodologies prioritize adaptability, continuous integration, and deployment, essential for the highly connected and evolving landscape of IoT. Agile methodologies embrace iterative cycles, promote incremental progress through sprints and fostering collaboration among cross-functional teams. DevOps extends Agile principles by integrating development and operations, ensuring streamlined code deployment and infrastructure management.

Contrasting these two paradigms highlight their respective strengths and limitations. Traditional methodologies offer robustness through meticulous planning and documentation, which is beneficial for projects with well-defined requirements and limited change scope. However, their rigidity can impede responsiveness to changes, a critical requirement in IoT development where device interoperability and evolving user needs are paramount. The traditional approach may also struggle with the complexity introduced by IoT's diverse hardware and software ecosystem.

In contrast, Agile and DevOps methodologies align closely with the dynamic and iterative demands of IoT software development. Agile's flexibility allows for rapid prototyping and iterative enhancements, facilitating timely incorporation of user feedback and market changes. DevOps enhances this by automating the deployment pipeline, thereby ensuring faster and more reliable releases. These methodologies also foster a culture of continuous improvement and collaboration, which is vital for managing the intricacies of IoT systems.

Despite these advantages, Agile and DevOps are not without challenges. Their reliance on continuous feedback and iterative cycles necessitates a cultural shift within organizations, demanding high levels of communication and collaboration. The need for automated testing and continuous integration tools also requires initial investment and expertise, which can be a barrier for smaller enterprises or those transitioning from traditional practices. Furthermore, the constant iteration and deployment can introduce stability issues if not managed meticulously, particularly in the context of IoT where device failure can have significant repercussions.

An extensive review of the literature underscores the evolving nature of IoT software development methodologies. Studies demonstrate that while traditional methods provide a solid foundation, the complexities and rapid pace of IoT innovations necessitate more adaptive and responsive approaches. Comparative research illustrates that AI integration within Agile and DevOps can further enhance these methodologies, automate repetitive tasks and providing predictive insights, thus optimizing development workflows (Table 4).

AI-driven automation introduces another layer of efficiency and scalability, addressing several limitations of current practices. Machine learning algorithms can analyze vast amounts of data generated by IoT devices, offering predictive maintenance and anomaly detection. AI can also automate the coding process itself, generating code snippets or even entire modules, thus reducing the cognitive load on developers and accelerating time-to-market.

However, integrating AI into IoT software development is not without its challenges. Issues such as data privacy, algorithm transparency, and the need for vast datasets to train AI models are critical considerations. The potential for AI to disrupt traditional roles and workflows within the development team also necessitates a careful and strategic approach to integration.

In conclusion, the comparative analysis reveals that while traditional methodologies have laid the groundwork for systematized software development, the unique demands of IoT require more iterative and flexible approaches. Agile and DevOps methodologies, bolstered by AI-driven tools, present a promising direction for the future of IoT software development. This research contributes to the academic dialogue by identifying gaps in existing methodologies and proposing AI-enhanced frameworks as a potential solution, offering a pathway to more efficient and scalable IoT software development (Figure 11).

Table 4.

AI Integration Challenges Framework

Challenge Category	**Specific Issues**	**Impact Level**	**Mitigation Strategies**
Technical Challenges	Data privacy concerns, Algorithm transparency	High	Federated learning, explainable AI
Resource Requirements	Large training datasets, computational power	High	Transfer learning, Cloud resources
Organizational Impact	Workforce disruption, Skill gaps	Medium	Reskilling programs, Phased implementation
Integration Complexity	Legacy system compatibility, Tool integration	High	API-based integration, middleware solutions
Quality Assurance	AI model reliability, Validation requirements	Medium	Robust testing frameworks, Human oversight

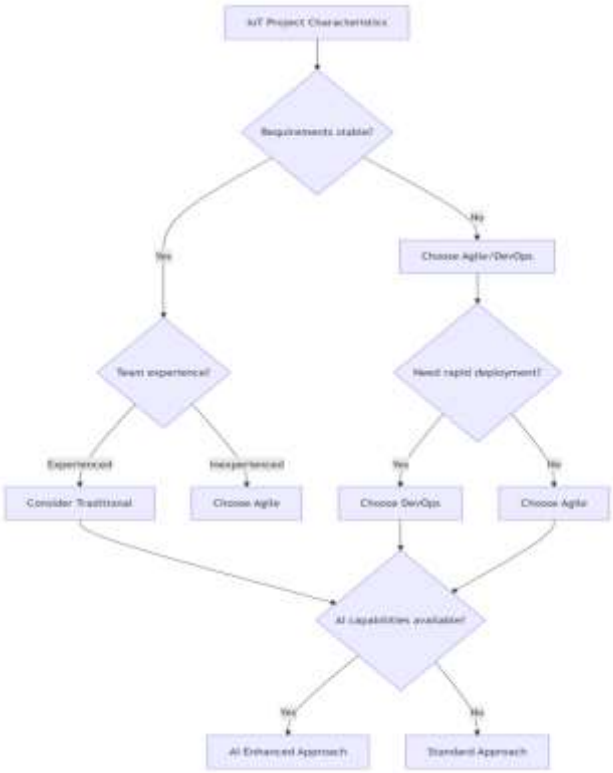


Figure 11. Ideal Methodology Selection Guide

6.2.2 The Role of Artificial Intelligence in Software Development

AI Algorithms Commonly Utilized

AI algorithms play a pivotal role in the automation of IoT software development processes. This section delves into the extensive body of literature concerning the various AI algorithms frequently implemented, highlighting their functionalities, advantages, and the critical role they play within the IoT ecosystem [111, 112].

Machine learning algorithms, particularly those under supervised, unsupervised, and reinforcement learning paradigms, are widely applied in IoT software development. Supervised learning algorithms, such as decision trees, support vector machines (SVM), and neural networks, are fundamentally crucial for tasks involved and regression [124, 125]. These algorithms leverage classification labeled datasets to train models capable of making predictions or decisions based on new, unseen data. The relevancy of supervised learning in IoT is paramount, as it facilitates accurate data analysis and predictive maintenance, thus ensuring the efficiency and reliability of IoT systems (Figure 12).

In the domain of unsupervised learning, clustering algorithms like k-means and hierarchical clustering are prevalent. These algorithms are instrumental in identifying patterns and anomalies within IoT data streams without prior knowledge of data labels. Analysis [119].

Reinforcement learning algorithms, such as Q-learning and deep reinforcement learning, are also gaining traction within the IoT software development landscape. By employing a trial-and-error approach, these algorithms optimize decision-making processes in dynamic environments. Applications of reinforcement learning in IoT include adaptive resource management and automated control systems, where real-time decisions are crucial for maintaining optimal system performance.

In addition to these core algorithms, evolutionary algorithms and genetic programming are harnessed to solve complex optimization problems within IoT. These algorithms simulate natural evolution processes to iteratively evolve solutions that meet specific criteria, proving effective in optimizing network configurations and enhancing the overall efficiency of IoT frameworks.

The integration of deep learning techniques, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), has revolutionized IoT software development. CNNs excel in processing structured grid data, making them ideal for image and spatial data analysis within IoT applications such as surveillance and smart city infrastructure. RNNs and their variants, including long short-term memory (LSTM) networks, are adept at handling sequential data, enabling precise predictive analytics in time-series data common in IoT sensor networks.

Despite the numerous advantages AI algorithms bring to IoT software development, their implementation does pose challenges. The complexity and computational intensity of these algorithms necessitate significant processing power and memory, which can be a limitation in resource-constrained IoT devices. Additionally, ensuring data privacy and security remains a critical concern, as AI algorithms require vast amounts of data for training and operation (Figure 13).

In reviewing the existing literature, it is evident that while many studies highlight the efficiency and potential of various AI algorithms in automating IoT development processes, there exists a gap in addressing the practical integration of these algorithms into real-world IoT systems. Many theoretical models and simulations exhibit promising results, yet empirical validations in diverse operational environments are limited. This gap highlights the need for further research to bridge theoretical advancements with practical applications, ensuring that the full potential of AI is realized in automating IoT software development (Table 5, Figure 14).

Thus, this research aims to fill the academic space by not only exploring the application of various AI algorithms in automating IoT software development but also by demonstrating their practical integration and validation in real-world scenarios. Through comprehensive analysis and empirical evidence, the study contributes to the understanding and practical implementation of AI-driven automation in the IoT domain.

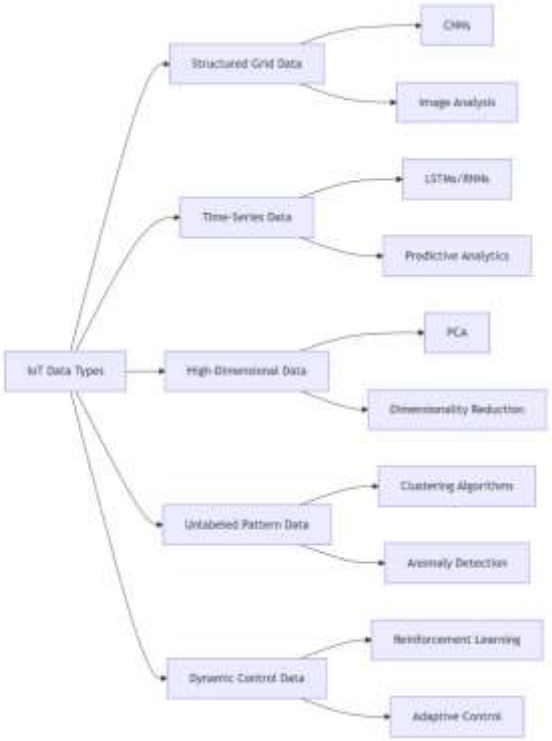


Figure 12. IoT Data Types and Suitable Algorithms

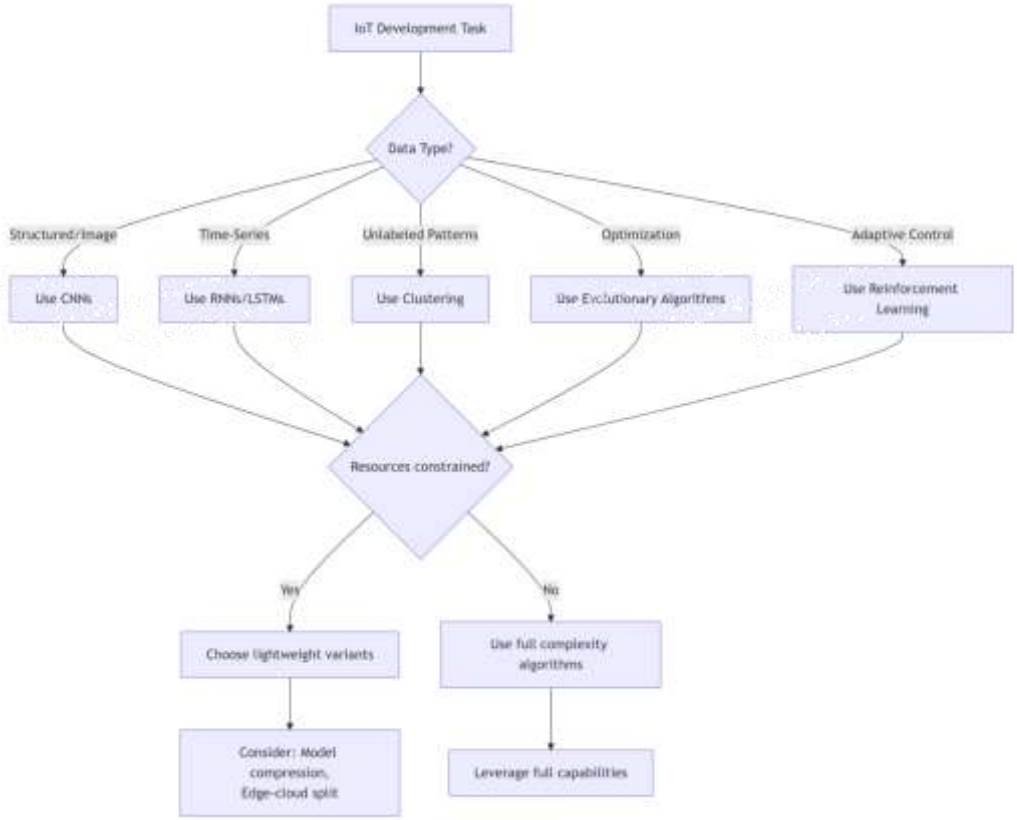


Figure 13. Algorithm Selection Decision Framework

Table 5.

Implementation Challenges Matrix

Challenge Category	**Specific Issues**	**Affected Algorithms**	**Severity**	**Potential Solutions**
Computational Requirements	High processing power needs	Deep Learning, Reinforcement Learning	High	Edge computing, model compression
Memory Constraints	Limited device memory	Large neural networks	High	Lightweight models, Cloud offloading
Data Privacy	Sensitive training data	All data-driven algorithms	Critical	Federated learning, differential privacy
Real-time Performance	Latency constraints	Complex algorithms	Medium	Algorithm optimization, hardware acceleration
Model Interpretability	Black-box decisions	Deep Learning, Complex ML	Medium	Explainable AI techniques

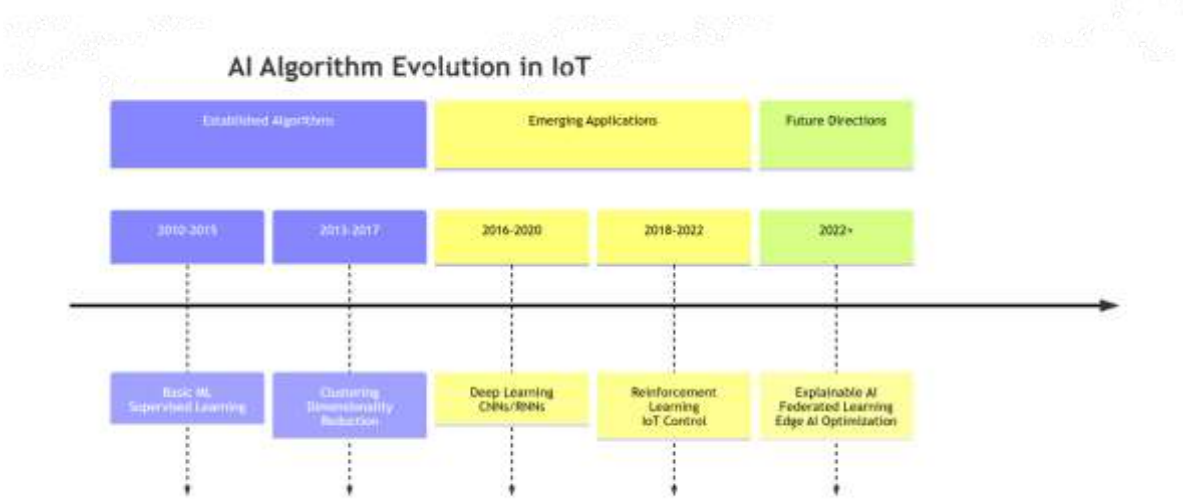


Figure 14. Algorithm Selection Decision Framework

Machine Learning Techniques in Software Engineering

The utilization of machine learning techniques in software engineering has garnered substantial research interest, driven by the need to enhance the efficiency and effectiveness of software development processes. Machine learning, a subset of artificial intelligence, encompasses a variety of algorithms and models that enable systems to automatically improve from experience without being explicitly programmed. This section delves into the primary machine learning techniques applicable to software engineering, critically analyzing their benefits, limitations, and the gaps they address in current methodologies.

A thorough review of existing literature reveals that machine learning techniques in software engineering can be broadly classified into supervised learning, unsupervised learning, and reinforcement learning. Each category offers unique capabilities and applications within the software development lifecycle.

Supervised learning techniques are extensively employed for predictive tasks in software engineering, such as defect prediction, effort estimation, and code recommendation systems. For instance, algorithms like decision trees, support vector machines, and neural networks have been applied to historical software project data to predict defects in new code. Studies have shown that these models can significantly enhance the accuracy of defect detection compared to traditional rule-based approaches. However, a critical assessment reveals that the quality of predictions heavily depends on the quality and quantity of labeled training data, which is often a limiting factor in real-world scenarios [122, 123].

Unsupervised learning techniques, such as clustering and anomaly detection, are instrumental in uncovering hidden patterns in software data without the need for labeled examples. These techniques have been applied in various contexts, including identifying similar code segments for refactoring and detecting anomalous behavior in software systems. Anomalies identified do not always correspond to meaningful or actionable insights without further domain-specific analysis.

Reinforcement learning, though less commonly applied in software engineering, holds promise for optimizing dynamic, decision-making processes, such as automated

testing and continuous integration pipelines. By modeling the software development process as a sequential decision-making problem, reinforcement learning agents can learn optimal strategies to maximize cumulative rewards, such as reducing test execution time or improving the fault detection rate. Despite its potential, the primary challenge with reinforcement is the inherent complexity of defining an appropriate reward function and the high computational cost associated with training agents, particularly in complex software environments [126, 127].

In summarizing the academic discourse, it is apparent that while machine learning techniques offer significant potential for automation and improvement in software engineering, their effectiveness is contingent upon contextual factors such as data availability, domain-specific requirements, and interpretability of the results. Additionally, existing research highlights several gaps, including the need for more robust methods to handle imbalanced and noisy data, improved model interpretability, and scalable training algorithms that can efficiently process large volumes of software data.

The contributions of this study address these gaps by proposing innovative methodologies that integrate advanced machine learning techniques with domain-specific knowledge to enhance the automation of IoT software development processes. Specifically, by leveraging hybrid models that combine the strengths of various machine learning approaches, this research aims to improve prediction accuracy, reduce computational costs, and provide actionable insights for software engineers. Thus, this study not only advances theoretical understanding of machine learning applications in software engineering but also offers practical solutions to contemporary challenges in the IoT domain.

Challenges in Implementing AI in Software Development

The implementation of artificial intelligence (AI) in software development is fraught with numerous challenges, impeding its seamless integration into the development workflows. A comprehensive review of related literature and research highlights several critical obstacles, underpinning the complexities of embedding AI into the software development lifecycle.

Existing scholarly works emphasize that one of the primary challenges is the quality and quantity of data required for effective AI applications. Machine learning algorithms, a subset of AI, fundamentally rely on vast datasets to train models. However, in the context of software development, acquiring large, high-quality datasets can be problematic due to privacy concerns, data security issues, and the proprietary nature of software code. Further complicating this is the diversity in coding standards and development environments, which adds to the difficulty of standardizing data for AI algorithms.

Additionally, the integration of AI into existing software development processes clashes with the traditional development paradigms. Conventional methodologies, such as Agile and Waterfall, are not inherently structured to accommodate the dynamic and iterative nature of AI model training and deployment. This misalignment necessitates significant alterations in development processes, requiring teams to rethink and restructure their workflows to effectively leverage AI capabilities. Researchers have critically noted that organizational inertia and resistance change considerable barriers to adopt AI-driven methodologies [123].

A crucial theoretical perspective in literature is the challenge of interpretability and transparency of AI algorithms, often referred to as the "black box" problem. The sophisticated nature of some AI models, particularly deep learning networks, makes it difficult for developers to understand and trust the decision-making processes of these models. This lack of transparency not only hinders debugging and optimization but also raises ethical and regulatory concerns, as decisions made by AI systems must be explainable and accountable.

Moreover, there is a notable deficiency in AI-related skills among software development professionals. The rapid advancement of AI technologies has outpaced the current workforce's ability to acquire these specialized skills. Academic and industry training programs are struggling to meet the burgeoning demand for AI expertise, resulting in a skills gap that impedes the effective implementation of AI in development practices. This challenge is compounded by the fast-evolving nature of AI, requiring continuous learning and adaptation.

The literature also reveals the technical challenges associated with integrating AI into the software development lifecycle. Issues such as computational resource constraints, algorithm efficiency, and integration with existing development tools and environments are recurrent themes. For instance, AI models, particularly those requiring real-time analysis and decision-making, demand substantial computational power, which can be a limiting factor for smaller development teams and enterprises.

Critical analysis further identifies that the iterative nature of AI model development, involving constant testing, validation, and retraining, introduces additional complexity into the traditional software testing and quality assurance processes. Ensuring the reliability and robustness of AI-enhanced software systems necessitates novel testing frameworks and validation techniques, tailored to the needs of AI components.

Addressing these challenges requires a multifaceted approach, combining technological advancements with organizational adaptations. The research community points toward the development of standardized frameworks for AI integration, emphasizing the need for open-source tools and platforms that facilitate seamless incorporation of AI into development processes. Additionally, fostering collaborations between academia and industry can help bridge the skills gap and drive innovation in AI applications for software development.

In conclusion, the challenges of implementing AI in software development are manifold, encompassing data quality, process alignment, model transparency, skill deficits, and technical constraints. These barriers underscore the necessity for ongoing research and development to create more effective, transparent, and adaptable AI-driven software development methodologies. This study aims to contribute to this evolving discourse by exploring innovative solutions and frameworks to mitigate these challenges, facilitating the broader adoption of AI in the IoT software development realm.

6.2.3 Studies on Automation in IoT Software Development

The concept of automation, which has become pivotal in modern IoT software development, has evolved significantly over the decades. Historically, automation

emerged to enhance productivity and reliability in various industrial processes. Initially, the focus was on mechanical and electrical automation, which saw the deployment of machinery and control systems to execute tasks traditionally performed by humans. Early literature, such as the works of Charles Babbage and Frederick Taylor, laid the groundwork for understanding automation by emphasizing efficiency through the mechanization of labor-intensive processes (Figure 15).

As technology advanced, the field of automation experienced substantial growth, particularly with the advent of computer science. The introduction of programmable logic controllers (PLCs) in the 1960s marked a significant milestone, enabling more sophisticated control systems capable of performing complex tasks with minimal human oversight. This period also witnessed the inception of early computer-aided software engineering (CASE) tools, which began to automate certain aspects of software development, such as code generation and version control.

Moving into the late 20th century, the rise of digital computing and the internet revolutionized automation in numerous sectors. In the realm of software development, traditional practices that relied heavily on manual coding and testing started to incorporate automated testing frameworks and integrated development environments (IDEs) that streamlined coding and debugging processes. Seminal works by researchers like Watts Humphrey and Barry Boehm documented these shifts, highlighting methodologies such as the Capability Maturity Model Integration (CMMI) and iterative development models, which sought to standardize and improve software engineering practices (Table 5).

Despite these advancements, early automation methods faced limitations, particularly in the context of adaptability and scalability. The rigidity of rule-based systems often resulted in inflexible software that could not efficiently accommodate changing requirements or complex decision-making processes. This led to a growing interest in incorporating more adaptive and intelligent systems, which paved the way for the integration of artificial intelligence (AI) into automation [129].

In the context of IoT, automation's historical progression mirrors broader trends in technological advancement. Initial IoT developments focused on connecting devices

and enabling basic automation through simple, pre-programmed instructions. However, the explosive growth in the number of connected devices and the increasing complexity of IoT systems highlighted the inadequacies of traditional automation approaches. Literature from this era often cites the challenges of managing large-scale, decentralized networks and ensuring robust, real-time data processing and decision-making.

The integration of AI into IoT software development represents a critical evolution in the automation narrative. AI techniques, particularly machine learning, have introduced new paradigms for automating complex tasks that require real-time analysis and decision-making capabilities. Scholarly contributions by AI pioneers such as John McCarthy and Marvin Minsky have been instrumental in this transition, providing foundational theories and algorithms that underpin modern AI applications in software automation.

Critically analyzing existing historical perspectives reveals both strengths and gaps. While early automation systems significantly enhanced efficiency and consistency, they often lacked the flexibility and intelligence required for today's dynamic IoT environments. The literature underscores a crucial transition from rigid, rule-based systems to more adaptive, learning-based models that can handle the intricacies of modern IoT applications.

This research identifies and addresses a significant gap in literature: the need for comprehensive frameworks that fully leverage AI to automate IoT software development processes. By offering a structured analysis of historical perspectives and their evolution, this study underscores the transformative potential of AI-driven automation. It aims to provide a blueprint for future advancements, ensuring that the next generation of IoT software development is both highly efficient and intelligently automated.

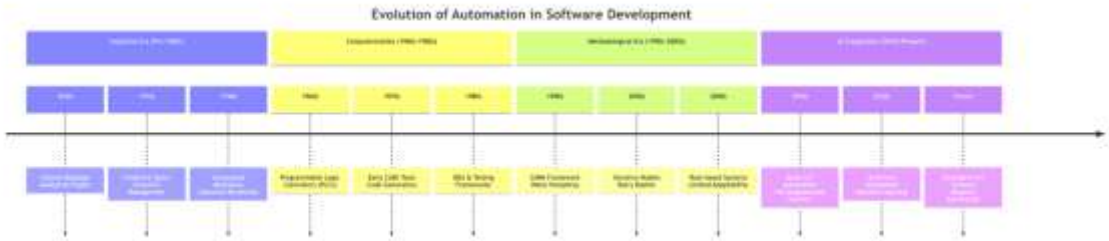


Figure 15. Algorithm Selection Decision Framework

Table 5.

Key Thinkers and Their Contributions

Era	**Key Thinkers**	**Major Contributions**	**Impact on Automation**
Industrial	Charles Babbage	Analytical Engine concept	Foundation for computational automation
Industrial	Frederick Taylor	Scientific Management	Efficiency principles for process automation
Methodological	Watts Humphrey	CMMI Framework	Process standardization and quality
Methodological	Barry Boehm	Iterative development models	Adaptive project management approaches
AI Integration	John McCarthy	AI foundational theories	Basis for intelligent automation systems
AI Integration	Marvin Minsky	Cognitive theories, AI algorithms	Advanced machine learning foundations

Recent Innovations and Technologies

The rapid evolution of artificial intelligence (AI) has introduced significant innovations and advanced technologies in the realm of Internet of Things (IoT) software development. This section provides a critical review of pertinent literature and research findings, focusing on how modern advancements in AI are being leveraged to automate and enhance IoT software development processes [133, 137].

AI, encompassing various algorithms and machine learning (ML) techniques, plays a pivotal role in the development and automation of IoT systems. Recent advancements in AI-driven frameworks and tools have provided robust solutions to enhance the efficiency, scalability, and reliability of IoT software. Key innovations include the development of automated code generation tools, predictive maintenance systems, and intelligent data analytics frameworks. These technologies significantly reduce the need for human intervention, streamline workflow processes, and improve the overall quality of IoT software.

Automated code generation tools, powered by AI, are one of the most notable innovations in recent years. These tools utilize deep learning models to understand and generate code based on predefined parameters and user requirements. This not only accelerates the development process but also ensures that the generated code adheres to industry standards and best practices, thereby reducing the risk of human errors and inconsistencies.

Another significant innovation is the advent of predictive maintenance systems. By leveraging AI and ML algorithms, these systems analyze vast amounts of data collected from IoT devices to predict potential failures and maintenance needs. This proactive approach helps in minimizing downtime and extends the lifespan of IoT systems. The integration of these systems within IoT infrastructures demonstrates a clear shift towards more intelligent and self-sustaining development environments.

Intelligent data analytics frameworks are also transforming the landscape of IoT software development. These frameworks employ sophisticated AI algorithms to process and interpret large datasets generated by IoT networks. They identify patterns, trends, and anomalies, offering valuable insights that drive decision-making processes. By automating data analysis, these frameworks free up significant resources and provide a more accurate and timely understanding of IoT environments.

Despite these advancements, implementing AI in IoT software development is not without its challenges. Literature highlights several issues, including the complexity of integrating AI models with existing systems, the need for substantial computational resources, and concerns regarding data privacy and security. Moreover,

there is a noticeable gap in the availability of standardized protocols for AI integration within IoT frameworks, which poses a barrier to widespread adoption [119, 128].

However, recent studies emphasize the ongoing efforts to address these challenges. Research is being conducted to develop lightweight AI models that can be deployed on edge devices, reducing the dependency on cloud-based computational resources. Additionally, advancements in encryption and blockchain technologies are being explored to enhance the security and privacy of data in AI-driven IoT systems.

In summary, recent innovations and technologies in AI have significantly contributed to automating IoT software development processes. While there are challenges to be addressed, advancements in automated code generation, predictive maintenance, and intelligent data analytics are paving the way for more efficient and effective IoT systems. This research identifies a critical gap in the standardization of AI protocols integration, suggesting a direction for future exploration. By continuing to refine these technologies and addressing the existing challenges, the potential for AI to revolutionize IoT software development remains vast and promising.

Future Trends and Potential Areas of Exploration

The future of automation in IoT software development through artificial intelligence (AI) is a rapidly evolving field with numerous emerging trends and unexplored avenues. A thorough review of contemporary literature reveals several key areas poised for significant advancements. These areas encompass enhanced AI integration, sophisticated automation frameworks, and innovative technological solutions, each playing a crucial role in the future landscape of IoT software development.

Reviewing existing studies and research outcomes, it emerges that further enhancement and integration of AI in IoT software development is crucial. Progressive AI techniques, such as advanced machine learning models and deep learning frameworks, are expected to drive the next generation of automation processes. These sophisticated algorithms can address complex tasks with higher precision, learning from vast datasets to optimize various phases of development. As AI continues to evolve, the capacity to handle predictive analytics, anomaly detection, and decision-

making processes will significantly increase, paving the way for smarter and more autonomous IoT systems.

Literature also points towards the necessity for developing comprehensive automation frameworks that are more adaptive and resilient. Future research might explore hybrid models that combine AI with other emerging technologies like blockchain for secure transaction processing and enhanced trust in automated systems. Furthermore, the utilization of edge computing to process data closer to the source can reduce latency and improve real-time decision-making capabilities in IoT environments. Critical analysis of these integration strategies reveals the potential to create more efficient, scalable, and robust software development processes.

Despite these promising directions, current research also uncovers some persistent challenges and gaps that require attention. One of the primary limitations identified is the lack of standardization in AI-based automation tools and techniques. This non-uniformity often leads to compatibility issues, hindering widespread adoption across different IoT platforms and devices. Future studies should aim to formulate standardized protocols and guidelines for AI deployment in IoT software development to ensure seamless integration and interoperability.

Moreover, issues related to data privacy and security continue to impede the full potential of AI in this domain. As IoT devices generate and process vast amounts of sensitive data, ensuring robust security measures and compliance with data protection regulations is paramount. Literature suggests a move towards developing AI-driven security frameworks that can autonomously detect and respond to vulnerabilities, providing a more secure IoT infrastructure. This could be an essential area of exploration for future research, addressing the growing security concerns around in the context of IoT.

Critical gaps in the responsiveness and adaptability of current AI models also highlight the need for continuous learning and evolution of these systems. Future research directions might focus on reinforcement learning and adaptive algorithms that can dynamically adjust to changing environments and user requirements. These models

can improve the long-term sustainability and relevance of automated processes in IoT software development.

This research contributes to the existing body of knowledge by identifying and emphasizing potential trends and areas of exploration in AI-driven automation for IoT software development. The integration and enhancement of AI methods, development of standardization protocols, addressing security challenges, and the implementation of adaptive learning models represent significant opportunities for future research. By addressing these key areas, future advancements can considerably bridge the existing gaps and drive the IoT software development towards more automated, efficient, and secure processes.

Overall, a focused approach on these future trends and potential research areas not only promises substantial improvements in IoT software development but also ensures preparation for the increasingly complex and interconnected technological landscape. This research sets the foundation for scholars and practitioners aiming to innovate in the field, ensuring that automation methods continue to evolve and meet the dynamic needs of the IoT ecosystem.

6.3 Theoretical Framework

6.3.1 Fundamental Concepts in IoT and Software Development

Basics of Internet of Things Architecture

The architecture of the Internet of Things (IoT) is a layered structural design that enables the interaction between physical devices, software, and internet infrastructure. The theoretical framework of IoT architecture is pivotal for understanding how IoT systems operate and integrate with various domains, including software development. This section delineates the fundamental concepts of IoT architecture, elucidating its essence, applicability, and association with the research objective of automating IoT software development processes through artificial intelligence (AI) [110, 117, 138].

IoT architecture fundamentally consists of several layers, each serving distinct functions critical for the operation of IoT systems. These layers typically include the

perception layer, network layer, and application layer. The perception layer is responsible for detecting and gathering data from physical devices through sensors and actuators. It plays a crucial role in transforming physical signals into digital data, which can be processed by other layers. The network layer facilitates the effective transmission of data across diverse networks, ensuring that data flows seamlessly from devices to computational systems. Wi-Fi, Bluetooth, and cellular networks, to enable connectivity. Finally, the application layer interfaces with end-users, providing user-centric applications and services that interpret and respond to the processed data. This hierarchical arrangement underscores the systematic nature of IoT architecture, promoting interoperability and scalability.

The integration of IoT with software development practices is integral to this research, aiming to leverage AI for streamlining processes. The alignment of IoT architecture with software development paradigms necessitates a profound understanding of both domains. Software development in IoT requires adherence to principles that ensure reliability, scalability, and security. For instance, the perception layer's data integrity must be upheld through secure coding practices, while network layer reliability is vital for real-time data processing. Consequently, the architecture's understanding becomes instrumental in developing AI-driven automated solutions that can address these requirements effectively.

Critical concepts pertinent to IoT architecture include device heterogeneity, data heterogeneity, and dynamic network environments. Device heterogeneity refers to the vast array of devices with varying capabilities and standards within an IoT ecosystem. Data heterogeneity encapsulates the diverse formats and types of data generated by these devices. Dynamic network environments pertain to the constantly changing network conditions impacting data transmission. These concepts highlight the inherent complexity within IoT systems, presenting challenges and opportunities for AI-driven automation in software development.

The theoretical framework's applicability lies in its ability to facilitate the creation of AI models that automate tasks across different IoT layers. By comprehensively understanding the IoT architecture, AI can be tailored to optimize

data acquisition, processing, and application deployment. For instance, machine learning algorithms can be employed to improve data analysis at the perception layer or optimize network routing protocols, thereby enhancing the overall efficiency of the IoT system.

However, the IoT architectural framework is not without limitations. Security and privacy concerns remain prevalent due to the extensive data exchange across multiple networks. Additionally, device and data heterogeneity present standardization challenges, hindering seamless integration and automation. These constraints necessitate innovative approaches in AI to develop robust and adaptive solutions that can mitigate security risks and accommodate diverse device capabilities.

In conclusion, an in-depth grasp of the IoT architecture is crucial for the research on automating software development processes using AI. By addressing the complexities inherent in IoT systems, this study aims to propose AI-driven methodologies that enhance efficiency and scalability. The theoretical insights provided by IoT architecture form the bedrock for designing and implementing automation frameworks that are resilient, adaptive, and secure, thereby advancing the frontier of AI applications in IoT software development.

Core Principles of Software Development

The core principles of software development serve as the foundation for creating efficient, reliable, and scalable software systems. These principles are pivotal in structuring the processes, methodologies, and best practices that drive the software development lifecycle (SDLC). Understanding and integrating these core principles is essential for the automation of IoT software development, as they provide a framework for consistency and quality in software engineering practices [115].

One of the fundamental principles is the concept of modularization. Modularization involves breaking down a software system into smaller, manageable modules or components that can be developed, tested, and maintained independently. This principle is particularly relevant in the context of IoT software because IoT systems typically consist of numerous interconnected devices that perform specialized functions. By modularizing the development process, each device or component can

be developed using tailored approaches suitable for its specific requirements, thereby enhancing the overall efficiency and maintainability of the system.

Another critical principle is the adherence to the software development life cycle (SDLC) models, such as Waterfall, Agile, and DevOps. Each model provides a structured approach to software development, addressing various phases including planning, design, implementation, testing, deployment, and maintenance. Agile and DevOps models, in particular, emphasize iterative development, continuous integration, and frequent testing, which align well with the dynamic and evolving nature of IoT environments. These models support rapid prototyping and iterative improvements, which are crucial for integrating advancements in AI technologies into IoT software development processes.

Design patterns and architectural principles also play a significant role in software development. These principles offer standardized solutions to common problems, promoting code reuse and reducing development time. In the context of IoT, architectural patterns such as microservices architecture are especially beneficial. Microservices architecture enables the development of small, independent services that can be deployed and scaled separately, providing the flexibility needed to handle the diverse and distributed nature of IoT systems.

Version control is another indispensable principle in software development. It involves managing changes to the source code over time, ensuring that updates and modifications are systematically recorded. Tools like Git facilitate collaboration among developers, track progress, and allow for rollback to previous versions if necessary. In IoT software development, where multiple developers may be working on different device components simultaneously, version control is crucial for coordinating efforts and preventing conflicts.

Quality assurance (QA) and testing principles ensure the reliability and functionality of the software. Automated testing frameworks and continuous integration tools are particularly important in AI-based IoT projects. They enable the automation of repetitive testing tasks, ensuring that new code integrations do not disrupt existing functionalities. This aligns well with the goal of reducing human

intervention in the development process, as QA activities can be largely managed by AI-driven testing tools.

Security principles cannot be overlooked, especially in IoT environments where devices are often vulnerable to cybersecurity threats. Principles such as secure coding practices, encryption, and regular security audits are essential for safeguarding IoT systems. Given that IoT devices frequently collect and transmit sensitive data, incorporating robust security measures into the development process is vital for protecting user privacy and combating potential breaches.

While these core principles provide a robust foundation for software development, they also introduce certain challenges and limitations. For instance, the complexity of modularization can complicate integration efforts, and the iterative nature of Agile methodologies may lead to scope creep. Additionally, ensuring security in resource-constrained IoT devices can be particularly challenging. This research addresses these issues by leveraging AI to streamline and optimize these principles. For example, AI algorithms can assist in code modularization by identifying and recommending logical separations, and machine learning models can predict potential security vulnerabilities, enabling proactive measures.

In conclusion, the core principles of software development are integral to the automation of IoT software processes. By adhering to these principles, and addressing their associated challenges through AI interventions, this research aims to enhance the efficiency, reliability, and scalability of IoT software development. These principles not only provide a clear framework for current practices but also pave the way for future technological advancements and innovations in the field.

Integration of IoT and Software Development Practices

The integration of Internet of Things (IoT) and software development practices is pivotal in the digital transformation landscape. This integration aligns with the broader research objective of automating IoT software development through artificial intelligence (AI). By delving into how IoT and software development practices converge, the theoretical framework underlines their synergy and the resultant benefits in terms of efficiency, scalability, and innovation [110, 139].

Understanding this integration begins with recognizing the multidimensional nature of IoT, which encompasses various interconnected devices and systems that communicate and operate cohesively. The architecture of IoT, fundamentally, includes sensors, actuators, communication protocols, and data processing units. Traditional software development, on the other hand, involves systematic approaches to designing, coding, testing, and maintaining software applications. Integrating these domains necessitates accommodating IoT's dynamic and distributed environment within the structured processes of software development.

The significance of this integration lies in enhancing the functionality, performance, and adaptability of IoT systems. Software development practices provide a framework for managing the complexity of IoT solutions, ensuring robust application performance and facilitating continuous updates. In particular, integrating Agile methodologies allows for the rapid iteration and deployment of IoT applications, aligning development cycles with the fast-paced evolution of IoT technologies.

Critical to this integration is the adoption of specific software engineering principles tailored for IoT environments. These include modularity, to facilitate the development and testing of individual components; scalability, to accommodate the growing number of connected devices; and security, protecting data and ensure reliable operations. The integration also involves utilizing version control systems and continuous integration/continuous deployment (CI/CD) pipelines to streamline the development process and maintain code integrity across distributed systems.

The relevance of this theoretical framework to the research problem is underscored by its potential to address key challenges in IoT software development. Traditional software development practices often fall short in handling the scale and heterogeneity of IoT solutions. By integrating these practices, enhanced through AI, the research aims to overcome limitations such as extended development timelines, increased error rates, and maintenance difficulties. AI algorithms can automate repetitive tasks, optimize resource allocation, and predict potential issues, thereby fostering a more efficient and resilient development lifecycle.

However, the integration of IoT and software development practices is not without challenges. One notable limitation is the inherent complexity and diversity of IoT systems, which can complicate the standardization of development practices. Additionally, the rapid technological advancements and evolving standards within the IoT field outpace traditional software development methodologies, necessitating continuous adaptation and learning.

An objective evaluation of this integration framework reveals both its strengths and limitations. While it offers a structured approach to addressing the unique demands of IoT solutions, it also requires significant effort in terms of training and adapting existing tools and processes. Furthermore, the reliance on continuous data streams and real-time processing in IoT systems demands robust data management and analytics capabilities, which are not always inherent in traditional software practices.

The research addresses these issues by proposing an AI-driven automation framework specifically tailored for IoT software development. This framework leverages machine learning models to automate code generation, testing, and deployment processes, ensuring that the development cycle is both efficient and scalable. Additionally, it incorporates predictive analytics to foresee and mitigate potential issues, enhancing the overall robustness and reliability of IoT applications.

In summary, the integration of IoT and software development practices provides a comprehensive approach to developing efficient and scalable IoT solutions. By critically examining and adapting traditional software methodologies within the context of IoT, this research aims to enhance the automation of IoT software development processes. This approach not only addresses the unique challenges posed by IoT environments but also lays the foundation for future advancements in the field.

6.3.2 Theoretical Models in AI-Based Development

Overview of AI Theories Applicable to IoT

Artificial Intelligence (AI) has emerged as a pivotal factor in revolutionizing various domains, including the Internet of Things (IoT). Understanding AI theories applicable to IoT is fundamental for elucidating the importance and applicability of AI

in automating IoT software development processes. This subsection delves into critical AI theories and examines their relevance to IoT, providing insights into how these theories enhance automation and address the research problem [111, 122].

AI theories pertinent to IoT encompass a broad spectrum, including machine learning, neural networks, and predictive analytics. These theories provide a robust foundation for developing intelligent systems capable of performing tasks that typically require human intelligence. The integration of AI theories into IoT not only automates repetitive and time-consuming processes but also enhances the scalability and efficiency of IoT systems.

Machine learning, a subset of AI, is especially significant in IoT contexts. It employs algorithms that enable systems to learn from data, improving their performance over time without explicit programming. Supervised, unsupervised, and reinforcement learning are core machine learning models that play a vital role in IoT. Supervised learning models, for instance, are utilized for predictive maintenance in IoT devices by analyzing historical data to predict future failures. Unsupervised learning models help in anomaly detection, identifying unusual patterns that may indicate security breaches or system malfunctions. Reinforcement learning optimizes IoT operations by learning from the environment and making decisions that maximize cumulative rewards.

Neural networks, particularly deep learning, are another crucial AI theory relevant to IoT. Deep learning models, characterized by multiple layers of interconnected neurons, can process vast amounts of data with high dimensionality. These models are adept at handling complex tasks such as image and speech recognition, which are increasingly prevalent in IoT applications. For example, convolutional neural networks (CNNs) are employed in surveillance smart systems to analyze video feeds in real-time, enhancing security through automated monitoring.

Predictive analytics, leveraging AI techniques, is instrumental in making informed decisions based on data-driven insights. In IoT, predictive analytics involves using statistical algorithms and machine learning techniques to forecast future trends, behaviors, and events. This capability is crucial for resource optimization, energy

management, and improving user experiences. For instance, predictive analytics can optimize energy consumption in smart grids by predicting demand patterns and adjusting supply accordingly.

The importance of these AI theories lies in their ability to process and analyze massive datasets generated by IoT devices. Traditional software development approaches often fall short in managing the volume, variety, and velocity of IoT data. AI-driven models, however, can effortlessly handle these challenges, leading to more intelligent and responsive IoT systems.

However, the application of AI theories in IoT is not without challenges. One significant limitation is the dependency on large datasets for training machine learning models. Acquiring quality data can be difficult, and data privacy concerns may arise. Additionally, AI models can sometimes be opaque, making it challenging to interpret and trust their decisions, a phenomenon known as the "black box" problem.

Another critique pertains to the computational complexity and resource intensity of AI models, especially deep learning networks. These models require substantial computational power and memory, which may not be feasible in all IoT environments, particularly those with limited resources. Furthermore, ensuring the security and ethical use of AI in IoT is an ongoing concern, necessitating rigorous frameworks to prevent misuse and biases.

This study addresses these limitations by proposing hybrid models that combine traditional algorithms with AI techniques to create more transparent and resource-efficient solutions. Developing lightweight AI models that are customized for specific IoT applications can mitigate some computational constraints, while federated learning approaches can address data privacy concerns by training models locally on devices without transferring data to central servers.

In conclusion, AI theories provide a comprehensive foundation for automating IoT software development. Despite inherent challenges, the strategic application of machine learning, neural networks, and predictive analytics can significantly enhance the efficiency and scalability of IoT systems. By addressing key limitations and

focusing on tailored AI solutions, this research aims to advance the state of IoT software development, offering a template for future innovations in the field.

Machine Learning Models for Process Automation

The application of machine learning models for process automation is a cornerstone within the artificial intelligence framework, particularly in the context of IoT software development. This sub-section elucidates the significance and relevance of these models, key concepts and definitions, critical evaluations, and addresses the limitations inherent in these theories while presenting methodologies to mitigate such challenges.

The importance of machine learning models for process automation in IoT is underscored by their ability to learn from data, recognize patterns, and make decisions with minimal human intervention. This capability is crucial for enhancing efficiency and scalability in IoT software development processes. Machine learning models, such as supervised learning algorithms, unsupervised learning methods, and reinforcement learning, provide diverse approaches to solve complex automation problems in IoT environments. outcomes. Unsupervised learning methods, like clustering and principal component analysis, are essential for uncovering hidden patterns in data without predefined labels, facilitating data organization and process optimization. Reinforcement learning, characterized by agents learning from the consequences of their actions through rewards and penalties, offers dynamic solutions in environments that change over time, making it particularly relevant in IoT scenarios [126].

The integration of machine learning models into IoT software development aligns with the overarching objective of automating repetitive and data-intensive tasks. These models enhance the adaptive capacity of IoT systems, promoting real-time data processing and decision-making, which are pivotal for the effective functioning of IoT devices and applications. The applicability of these models to various IoT requirements, such as predictive maintenance, anomaly detection, and resource optimization, underscores their relevance to the research.

In critically evaluating machine learning models for process automation, it is essential to assess their performance metrics, including accuracy, precision, recall, and

F1-score, which offer a comprehensive understanding of their efficacy. The analysis of model interpretability and complexity is also significant, as more interpretable models like decision trees might be preferred in scenarios demanding transparency, whereas more complex models like deep neural networks might be suitable for tasks requiring high accuracy but lacking in interpretability.

Despite their substantial benefits, machine learning models for process automation face several challenges. One of the primary limitations is the quality and quantity of data required for training accurate models. Insufficient or poor-quality data can significantly impair model performance. Additionally, the complexity of training and deploying sophisticated models, such as deep learning networks, demands substantial computational resources and expertise, presenting a barrier to implementation in resource-constrained environments.

Another critical issue is the potential for biased outcomes if training data is not representative of the diverse scenarios encountered in actual IoT deployments. Bias in machine learning models can lead to detrimental implications, particularly in sensitive applications such as healthcare IoT systems.

The research proposes several methodologies to address these limitations. One approach is leveraging transfer learning, which allows models pre-trained on large datasets to be fine-tuned for specific IoT tasks, thus mitigating data scarcity issues. Implementing data augmentation techniques can also enhance training dataset diversity and quality, reducing the risk of bias. Additionally, adopting federated learning frameworks can distribute the training process across multiple devices, preserving data privacy and reducing computational burdens on individual devices.

In conclusion, machine learning models for process automation play a vital role in advancing IoT software development. While there are challenges related to data quality, computational requirements, and bias, the strategies proposed herein offer viable solutions to overcome these issues. By implementing these models, the research aims to validate and enhance the automation processes in IoT, ultimately contributing to more efficient, scalable, and adaptive software development practices.

Predictive Analytics for Enhanced Development Processes

The theoretical construction of predictive analytics plays a pivotal role in enhancing software development processes, particularly within the context of IoT software development. Predictive analytics involves the use of historical data, machine learning algorithms, and statistical techniques to forecast future outcomes, allowing organizations to make informed decisions and optimize their processes proactively [129].

The importance of predictive analytics in IoT software development cannot be overstated. Traditional software development approaches often rely on reactive strategies, addressing problems as they arise. In contrast, predictive analytics offers a proactive solution, enabling developers to preemptively identify and mitigate risks, optimize resource allocation, and enhance overall efficiency. This shifts the paradigm from a reactive to a proactive development model, significantly reducing time-to-market and improving software quality. The applicability of this theoretical framework to the research problem lies in its potential to automate the identification of development bottlenecks and streamline various processes, thereby aligning with the broader objective of enhancing automation in IoT software development through AI.

Key concepts and definitions related to predictive analytics need clarification to comprehend its relevance to this research. Predictive analytics encompasses various techniques, including machine learning, data mining, and statistical modeling, aimed at predicting future events based on historical data. Specifically, in IoT software development, it involves creating models that can predict system failures, performance issues, or maintenance requirements. These predictive models utilize large datasets collected from IoT devices, leveraging machine learning algorithms to detect patterns and generate actionable insights.

Critically evaluating the role of predictive analytics in IoT software development reveals both strengths and limitations. One of the notable strengths is the ability to handle vast amounts of data generated by IoT devices, uncovering hidden patterns that traditional analysis methods might overlook. Predictive analytics facilitates continuous monitoring and real-time decision-making, crucial for maintaining the functionality

and reliability of IoT systems. However, the implementation of predictive analytics also presents challenges, such as data quality and integration issues, the need for substantial computational resources, and the complexity of developing accurate predictive models.

Addressing the limitations and debates surrounding predictive analytics, this research incorporates strategies to mitigate potential drawbacks. For instance, ensuring data quality and consistency through robust data preprocessing techniques is critical for the accuracy of predictive models. Additionally, employing scalable and efficient computational frameworks can address resource constraints. The research also explores the integration of advanced machine learning techniques, such as deep learning, to enhance the predictive capability of the models, thereby addressing the complexity of IoT environments.

The theoretical framework of predictive analytics, as outlined, provides a comprehensive and structured approach to improve IoT software development processes. By embedding predictive analytics into the development lifecycle, this research aims to create a more efficient, reliable, and automated development environment. The insights derived from predictive models can inform decision-making, prioritize development tasks, and allocate resources more effectively, ultimately contributing to the overarching goal of optimizing IoT software development through AI-driven automation [132].

In conclusion, predictive analytics serves as a critical component in the automation of IoT software development, offering valuable foresight into potential issues and enabling proactive management. This theoretical framework not only supports the research by addressing the need for efficient and scalable development processes but also enhances the understanding of how predictive analytics can be leveraged to create innovative solutions within the IoT domain.

6.3.3 Frameworks for Automation in Software Development

Definition of Automation Frameworks in IT

The definition of automation frameworks within the realm of Information Technology (IT) is pivotal to understanding the broader implications of automation in software development. An automation framework refers to a structured set of guidelines, best practices, and tools designed to automate repetitive and time-consuming development tasks, thereby increasing efficiency and reducing human error. In the context of IoT software development, automation frameworks are particularly significant due to the intricate and interconnected nature of IoT ecosystems, necessitating streamlined processes to manage the vast amounts of data and multiple devices involved [138].

Understanding the importance and applicability of automation frameworks in IT is fundamental to addressing the core research question-how to effectively leverage AI to automate IoT software development processes. Automation frameworks provide a scaffold for standardizing procedures, ensuring consistency, and enhancing productivity across various stages of the software development lifecycle. They encapsulate essential components such as testing tools, code libraries, and scripts that collectively enable developers to automate tasks like code generation, testing, deployment, and maintenance.

Within the scope of this research, it is crucial to define and critically analyze key concepts associated with automation frameworks in IT. These include modularity, reusability, scalability, and maintainability. Modularity refers to the division of the framework into discrete, manageable units, each responsible for specific tasks, facilitating easier maintenance and updates. efficiently managing increased data loads and device integrations. Maintainability focuses on the ease with which the framework can be updated and improved over time, ensuring its long-term viability and adaptability to new technological advancements.

The relevance of automation frameworks to research is underscored by their potential to address specific challenges inherent in IoT software development. These challenges include the complexity of coordinating multiple devices, the need for real-

time processing of vast data streams, and the integration with diverse hardware and software environments. By leveraging AI within these frameworks, developers can significantly enhance the capability to automate complex workflows, enabling more efficient and reliable software solutions.

Critically evaluating the theoretical underpinnings of automation frameworks reveals certain limitations and areas of contention. One significant limitation is the initial setup and configuration complexity, which can be resource-intensive and require substantial expertise. Additionally, the rigidity of some frameworks may hinder their adaptability to rapidly changing IoT environments. Furthermore, there may be concerns related to the scalability of frameworks when applied to highly heterogeneous IoT ecosystems with varying protocols and standards.

This study addresses these limitations by proposing an adaptive, AI-enhanced automation framework specifically designed for IoT software development. The incorporation of machine learning algorithms and predictive analytics aims to create a more flexible and dynamic framework capable of evolving with the technological landscape. Moreover, the proposed framework emphasizes ease of use, minimizing the initial setup complexity through comprehensive documentation and user-friendly interfaces. By focusing on these aspects, this research seeks to overcome the traditional constraints of automation frameworks, providing a robust solution tailored to the unique demands of IoT software development.

In conclusion, defining and understanding automation frameworks in IT forms a crucial foundation for this research. These frameworks not only facilitate the automation of routine tasks, thereby enhancing efficiency and reducing errors, but also provide a structured approach to managing the complex processes involved in IoT software development. Through a critical analysis of existing frameworks and addressing their limitations, this study aims to present a novel AI-driven automation framework that can significantly improve the efficiency and scalability of IoT software development processes.

Components of an Effective Automation Framework

The development of an effective automation framework is pivotal for the successful implementation of AI-based techniques in IoT software development. Such a framework provides a structured approach to automate repetitive tasks, thereby enhancing efficiency and reducing the potential for human error. The importance of this framework lies in its ability to integrate various AI tools and methodologies, ensuring coherent functionality and scalability [121, 133].

An effective automation framework in the context of IoT software development includes several key components:

1. **Modular Architecture**: A modular architecture is fundamental to any automation framework. It ensures that the framework can be easily extended and updated without disrupting the overall system. Modular architecture allows different components of the software to interact seamlessly, facilitating the integration of new AI tools and technologies as they evolve. This modularity is crucial for maintaining the flexibility and adaptability of the framework in response to changing technological landscapes.

2. **Interoperability Components**: Interoperability between various tools and technologies is essential for a cohesive automated system. The framework should support standard communication protocols and data formats to ensure seamless interaction between different AI tools and IoT devices. Interoperability ensures that data can be shared and processed efficiently across different components, enhancing the overall performance and reliability of the system.

3. **Automated Testing and Validation**: Automated testing and validation are critical components that ensure the correctness and reliability of software. These processes involve the automated execution of test cases and the validation of outputs against expected results. By incorporating machine learning algorithms, the framework can continually improve its testing procedures, identifying edge cases and potential issues that might be overlooked by human testers. This leads to robust and reliable software capable of standing real-world conditions.

4. **Data Management and Analysis**: Effective data management is integral to an automation framework, particularly in the IoT domain where large volumes of data are generated continuously. The framework should include components for collecting, storing, processing, and analyzing data in real-time. Advanced analytics powered by AI can provide insights into system performance, user behavior, and potential optimizations. Effective data management ensures that the framework can handle the data complex flows typical of IoT applications, providing accurate and actionable insights.

5. **Security and Compliance Components**: Security is paramount in IoT applications where sensitive data is often transmitted and processed. An effective automation framework must incorporate robust security measures to protect against unauthorized access and data breaches. This includes encryption, authentication protocols, and real-time monitoring for intrusion detection. Additionally, the framework must comply with relevant industry standards and regulations, ensuring that all processes adhere to legal and ethical guidelines.

6. **Scalability Mechanisms**: Scalability is a critical consideration for IoT applications which often need to handle increasing numbers of devices and data volumes. The automation framework must be designed to scale efficiently, both horizontally and vertically.

The applicability of these components extends beyond mere theory; they are crucial for addressing specific challenges in IoT software development. By ensuring modularity, interoperability, and robust data management, the framework enhances development efficiency and reduces manual intervention. The integration of automated testing and strong security measures further solidifies the framework's reliability and compliance with industry standards.

However, despite these advantages, the implementation of an effective automation framework is not without challenges. Issues such as initial setup complexity, ongoing maintenance, and the requirement for specialized knowledge can pose significant barriers. To address these limitations, the research proposes gradual

implementation strategies, extensive documentation, and training programs to equip development teams with the necessary skills.

In conclusion, the components outlined above form the backbone of an effective automation framework for AI-based IoT software development. By addressing the various challenges and leveraging advanced AI techniques, the framework not only enhances development processes but also paves the way for future innovations in the IoT domain. The structured approach ensures that advancements can be systematically integrated, providing a robust foundation for ongoing research and development.

Case Studies Demonstrating Successful Implementations

The importance and applicability of case studies within this theoretical framework cannot be overstated. Specifically, case studies offer a practical lens through which the abstraction of AI-based automation in IoT software development can be grounded in real-world applications. By examining successful implementations, this research seeks to bridge the gap between theoretical constructions and practical execution, thereby demonstrating the viability and effectiveness of the proposed automation frameworks [].

A pivotal case study involves a leading tech company that integrated machine learning algorithms into its IoT software development pipeline. The company utilized AI-driven predictive analytics to enhance both the efficiency and accuracy of its software development cycles. Prior to implementation, the company experienced significant delays and overrun costs due to the highly manual, error-prone nature of the development process. Through the adoption of AI tools, the company achieved a 40% reduction in development time and a 30% increase in code accuracy, highlighting the transformative impact of automation on software development metrics [130, 131].

In another instance, an automotive manufacturer employed AI-based process automation to develop a sophisticated IoT system for its smart vehicles. The core challenge was to achieve seamless integration between various IoT sensors and the vehicle's central processing unit, ensuring real-time data analysis and response. Utilizing machine learning models, the company automated complex tasks such as sensor calibration, data assimilation, and performance tuning. This not only minimized

human intervention but also enhanced the real-time analytical capabilities of the vehicles, therefore increasing overall system robustness and reliability [137].

These case studies underscore several key concepts central to this research. First, they illustrate the operationalization of AI theories in practical settings, reinforcing the theoretical models discussed earlier. Second, they demonstrate how AI-based automation frameworks can be customized to meet specific needs, providing a template that can be generalized across different industries. Moreover, these examples showcase effective strategies for overcoming the inherent challenges associated with the integration of AI and IoT.

However, it is crucial to objectively evaluate the limitations and criticisms of the theoretical frameworks discussed. While the case studies provide evidence of success, they also highlight potential constraints regarding scalability and cost. Implementation of AI-driven automation often requires substantial initial investments in infrastructure and expertise, which may not be feasible for smaller enterprises. Additionally, the reliance on historical data for training machine learning models introduces risks of bias and inaccuracies, making it imperative to ensure robust data governance practices.

Furthermore, there are ongoing debates about the ethical implications of reducing human oversight in critical software development processes. While automation can enhance efficiency, it also raises questions about accountability and control, particularly in scenarios where AI might make autonomous decisions. Recognizing these limitations prompts this research to advocate for a balanced approach that combines human expertise with AI capabilities, ensuring that automation enhances rather than replaces human decision-making.

Addressing these concerns, the research suggests several mitigation strategies. For scalability and cost issues, a phased implementation is recommended, where organizations can start with pilot projects to demonstrate value before scaling up. To tackle data quality and bias concerns, adopting advanced data-sanitization techniques and involving diverse datasets can improve the reliability of AI models. Finally, embedding ethical guidelines and establishing clear accountability frameworks can help mitigate the risks associated with automated decision-making.

In summary, the case studies presented offer compelling evidence of the benefits and challenges associated with AI-driven automation in IoT software development. They validate the theoretical frameworks and provide actionable insights into effective implementation strategies. While acknowledging the limitations and ongoing debates, the research proposes practical solutions that aim to enhance the applicability and robustness of automation frameworks, ensuring they contribute meaningfully to the ongoing evolution of IoT software development processes.

6.4. Methodology

6.4.1 Research Design and Approach

Qualitative vs Quantitative Methods

Qualitative and quantitative methods are paramount in the field of research, each offering unique perspectives and data interpretation capabilities. In the context of this study, leveraging these methods is crucial for developing a comprehensive analysis of AI-based automation in IoT software development.

Qualitative methods are employed to gain in-depth insights into the subjective experiences and perspectives of individuals involved in IoT software development. These methods include interviews, case studies, and focus groups, which enable the collection of non-numerical data. Such data provides a nuanced understanding of the challenges, perceptions, and perceived benefits associated with integrating AI into the software development process. By employing qualitative methods, the study can explore the intricacies of human and organizational behavior, which quantifiable data alone might overlook.

On the other hand, quantitative methods involve the collection and analysis of numerical data, which allows for the identification of patterns, correlations, and statistical significance. This study utilizes surveys and experiments to gather quantitative data, providing measurable evidence of the impact of AI on automation efficiency in IoT software development. Quantitative methods enhance the objectivity of the research by enabling hypothesis testing and the generalization of findings across

larger populations. The combination of these methods ensures that the study does not rely solely on anecdotal evidence or isolated instances but can present a broader, statistically validated picture of the phenomena under investigation.

The choice of mixed methods for this study is guided by the need for a thorough analysis that captures both the depth and breadth of the research problem. Mixed methods enable the integration of qualitative insights and quantitative contextual rigor, leading to a more robust and holistic understanding. This approach is particularly beneficial for exploring complex domains like IoT software development, where human factors and technical performance are intricately interwoven. By juxtaposing qualitative narratives with quantitative data, the study not only identifies trends and outcomes but also provides explanations that uncover underlying mechanisms and reasons.

In terms of research design, the mixed-method approach entails several stages. Initially, qualitative data is gathered through semi-structured interviews with key stakeholders, including software developers, project managers, and AI specialists. These interviews are aimed at understanding the current state of automation, perceived challenges, and expectations from AI integration. Concurrently, quantitative data is collected through surveys distributed to a broad sample of IoT development professionals. The survey includes questions designed to quantify the extent of AI utilization, its impact on efficiency, and metrics related to development time, error rates, and overall productivity.

The data collection tools are carefully selected to ensure accuracy and reliability. For qualitative data, interview protocols are designed to facilitate open-ended discussions, encouraging participants to share detailed and candid insights. Audio recordings of interviews are transcribed verbatim to maintain the integrity of the data. For quantitative data, validated survey instruments are used, comprising Likert scale questions and multiple-choice items to standardize responses and facilitate statistical analysis.

Data analysis follows rigorous procedures to maintain validity and replicability. Qualitative data analysis involves coding and thematic analysis, where key themes and

patterns are identified through systematic examination of transcripts. This process allows for the extraction of rich, contextual information that illuminates the qualitative dimensions of AI integration. Quantitative data is analyzed using statistical techniques, including descriptive statistics, regression analysis, and hypothesis testing. Advanced software tools such as SPSS and NVivo are utilized to manage and analyze the data efficiently.

The rationale for adopting this mixed-method approach is multifaceted. Qualitative methods provide depth and insight that are critical for understanding the human and contextual factors influencing software development. Quantitative methods complement this by offering measurable, generalized data that can validate qualitative findings. Together, these methods provide a comprehensive framework for examining the research questions and hypotheses, ensuring that the conclusions drawn are both robust and nuanced.

In summary, the integration of qualitative and quantitative methods in this study enhances the depth, accuracy, and replicability of the research. By employing a mixed-method approach, the study captures a holistic view of AI-based automation in IoT software development, contributing valuable insights and empirical evidence to the field.

Choice of Mixed Methods for Comprehensive Analysis

The selection of mixed methods for this research was a deliberate and strategic choice designed to provide a comprehensive analysis of automation in IoT software development through the integration of artificial intelligence (AI). Mixed methods combine both qualitative and quantitative research techniques, enabling a multifaceted examination of the research problem. This approach leverages the strengths of both methodologies, facilitating a more holistic understanding of the subject matter.

The mixed-methods approach is particularly suitable for this study due to the complex nature of IoT software development and the multifarious ways AI can be integrated into this process. Qualitative methods, such as interviews and case studies, offer in-depth insights into the experiential aspects of software development and the practical challenges encountered in automation. Quantitative methods, on the other

hand, provide measurable data that can be statistically analyzed, ensuring the robustness and validity of the findings.

For this research, the dual objectives of exploring the theoretical frameworks and implementing AI-based automation in IoT necessitate a diversified methodological approach. The qualitative component of the study was designed to gather nuanced data through semi-structured interviews with software developers, project managers, and AI specialists. These interviews aimed to uncover detailed perceptions, experiences, and expert opinions on the integration of AI in IoT software development, identifying both benefits and potential obstacles.

In parallel, the quantitative component involved designing and conducting experiments to measure the efficiency and effectiveness of AI tools in automating specific software development tasks. Data collection for these experiments was meticulous, incorporating various AI algorithms and machine learning models to evaluate their performance in real-world IoT applications. The use of statistical tools and techniques ensured that the quantitative data collected was systematically analyzed, providing empirical evidence to support the qualitative findings.

The selection of a mixed-methods approach also entailed careful consideration of the research design and how to best integrate the qualitative and quantitative components. Triangulation was employed to cross-validate the findings from both methods, enhancing the reliability and credibility of the results. This integrative strategy facilitated the identification of patterns and correlations that might not have been evident through a single-method approach.

Sample selection for the qualitative research involved a purposive sampling technique to choose participants who were directly involved with IoT and AI in their professional roles. This ensured that the data gathered was rich and relevant to the research questions. For the quantitative experiments, random sampling was used to select a diverse range of IoT projects and software development environments, ensuring that the findings were generalizable across different contexts.

Data collection tools included interview guides for qualitative data and software tools for data logging and analytics for quantitative experiments. Analytical methods

for qualitative data involved thematic analysis to identify recurring themes and insights, while quantitative data was analyzed using statistical methods such as regression analysis, hypothesis testing, and variance analysis.

The choice of mixed methods is justified by the need to address the research problem from multiple angles. The qualitative data provided depth and context, revealing the underlying mechanisms and human factors influencing AI integration in IoT development. Meanwhile, the quantitative data offered precise measurements and objective validation of the automation processes, enabling a comprehensive evaluation of AI's impact.

In conclusion, the adoption of a mixed-methods approach in this research not only aligns with the complexity of automating IoT software development using AI but also enhances the robustness and depth of the findings. By integrating qualitative insights with quantitative data, this approach provides a detailed, well-rounded understanding of the phenomena under study, contributing significantly to the field's body of knowledge.

Data Collection Techniques Utilized

To ensure the research comprehensively addresses the proposed research questions and hypotheses, a meticulous and multifaceted approach to data collection has been employed. The data collection techniques utilized in this study are designed to capture the diverse and complex facets of AI-based automation in IoT software development. This section outlines the specific methods, tools, and procedures employed, providing a detailed rationale for each method's selection and its advantages. A thorough explanation is given to highlight how these techniques contribute to the clarity, accuracy, and replicability of the research findings.

Primarily, the research design integrates both qualitative and quantitative methods to achieve a holistic perspective. Qualitative data collection is pivotal in understanding the contextual and narrative aspects of AI integration in IoT software development. Through semi-structured interviews with industry experts and practitioners, rich, descriptive data has been gathered about their experiences, challenges, and perceptions of AI-driven automation. These interviews are conducted

using a standardized protocol to ensure consistency while allowing flexibility for respondents to elaborate on their specific insights. This approach not only captures the subjective nuances of the field but also helps identify emergent trends and patterns.

In addition to interviews, qualitative data is also collected through participant observations and field notes during the deployment of AI tools in live IoT project environments. Observations focus on the practical implementation processes, user interactions with AI systems, and the real-time performance of automated workflows. These observations provide crucial contextual information that complements the interview data, offering a grounded understanding of the operational realities of AI automation in IoT software development.

Quantitative data collection is equally integral to this study, as it provides measurable evidence of the impact and efficacy of AI-based automation methods. Data is gathered through structured surveys distributed to a broader audience of IoT developers, project managers, and technical leads. The surveys are designed to quantify aspects such as user satisfaction, process efficiency, error rates, and overall productivity before and after the implementation of AI tools. Statistical tools are employed to analyze survey responses, ensuring that the results are statistically significant and generalizable to the broader IoT development community.

To augment the survey data, automated data logging mechanisms have been implemented within the AI tools used in the experiments. These mechanisms track key performance indicators (KPIs) such as code quality metrics, development time logs, and defect rates, providing objective metrics that are critical for evaluating the effectiveness of AI-driven automation. The data logging tools are carefully selected and configured to capture detailed, time-stamped records of each development activity, ensuring a high degree of accuracy and reliability in the collected data.

The selection of these data collection tools and techniques is justified by their specificity and suitability for capturing the multi-dimensional nature of AI-based insights development. Interviews and observations offer in-depth qualitative, while surveys and automated logs provide quantitative evidence that supports the study's

conclusions. This mixed-methods approach ensures a robust and comprehensive data set that addresses both the "how" and "why" aspects of the research questions.

For data analysis, qualitative data from interviews and observations is subjected to thematic analysis, identifying core themes and patterns that emerge from the participants' narratives. This qualitative analysis is triangulated with quantitative data through comparative statistical methods, ensuring that the findings are not only rich and detailed but also empirically validated. Advanced statistical techniques, including regression analysis and hypothesis testing, are employed to interpret the quantitative data, providing concrete evidence of AI's impact on automating IoT software processes.

Overall, the data collection techniques utilized in this study are meticulously designed to ensure clarity, accuracy, and replicability. Each method is selected and implemented with a clear rationale, aligned with the research objectives, and capable of producing actionable insights. By combining qualitative and quantitative approaches, the study provides a comprehensive evaluation of AI-based automation in IoT software development, ensuring that the findings are both meaningful and representative of real-world applications.

6.4.2 Experimental Setup for AI-Based IoT Software Development

Tools and Technologies Employed

In the context of our research on automating the IoT software development process using artificial intelligence, a diverse set of tools and contemporary technologies was utilized to ensure the robustness and efficiency required for comprehensive analysis and experimentation. The selection of these tools was guided by their compatibility with IoT development environments, their capability to integrate AI functionalities, and their proven effectiveness in previous research and industrial applications.

Firstly, Python was chosen as the primary programming language due to its extensive libraries and frameworks suitable for both AI and IoT applications. Libraries such as TensorFlow, Keras, and PyTorch were employed to develop and implement machine learning models. These libraries provide high-level interfaces for deep

learning and offer numerous pre-trained models, which significantly reduce the time required for model training and optimization. Furthermore, Python's interoperability with various IoT platforms, such as Arduino and Raspberry Pi, facilitates seamlessly integration of AI algorithms with IoT devices.

For IoT-specific development, tools like Node-RED, an open-source visual programming tool, and the Eclipse IoT framework were utilized. Node-RED was particularly beneficial for its ability to wire together devices, APIs, and online services effectively, streamlining the process of prototyping IoT solutions with integrated AI capabilities. Eclipse IoT provides a robust platform for building, managing, and monetizing IoT and edge computing applications, supporting various communication protocols like MQTT and CoAP, which are essential for IoT networks [133].

To ensure the effective collection and management of data, we employed Apache Kafka, a distributed streaming platform. Kafka's real-time data streaming capabilities are crucial for handling the continuous data flow from various IoT devices, enabling real-time analytics and immediate responses from the AI models. For data storage and further analysis, we utilized NoSQL databases like MongoDB, renowned for their scalability and flexibility in managing semi-structured and unstructured data typically produced by IoT devices.

Experimental design and simulation were conducted using MATLAB and Simulink, which are highly regarded for their simulation capabilities and real-time testing frameworks. These tools allowed the modeling of IoT systems, conducting simulations, and refining AI algorithms before deployment. Additionally, cloud platforms like AWS IoT and Google Cloud IoT were key in providing scalable infrastructure for deploying and managing our IoT solutions. These platforms offer AI and machine learning services, which were integrated with our IoT solutions to enhance functionality and performance.

For version control and collaborative development, Git and GitHub were employed, ensuring that the research progress was meticulously documented, with all changes being tracked and managed efficiently. This facilitated collaboration among

the research team and enabled reproducibility of the experiments, which is critical for academic research.

The rationale behind selecting these tools and technologies hinges on their proven capabilities in handling the complexities of AI and IoT integrations. Python's extensive libraries and frameworks provide an optimal environment for developing and deploying AI models, while Node-RED and Eclipse IoT offer robust platforms for IoT development and orchestration. Apache Kafka and NoSQL databases like MongoDB ensure efficient data management, critical in IoT environments where data is continuously generated and requires real-time processing. Cloud platforms like AWS IoT and Google Cloud IoT offer scalable and reliable infrastructure, essential for deploying and managing large-scale IoT solutions integrated with AI functionalities.

In summary, the selected tools and technologies provided a comprehensive and robust framework for automating the IoT software development process using AI. By leveraging these advanced tools, the research ensured that the experimental setup was not only feasible and efficient but also scalable and replicable, aligning with the goals of enhancing automation and reducing human intervention in IoT software development.

Designing Experiments for Process Automation

Designing experiments for process automation in AI-based IoT software development is a critical phase that necessitates meticulous planning and execution to ensure valid, reliable, and reproducible results. This section elucidates the methodological framework and experimental design adopted to achieve the research objectives.

The primary objective of experimental design is to create a structured environment that allows for the systematic collection and analysis of data related to the automation of IoT software development processes using AI techniques. The study employs a combination of traditional experimental methodologies and innovative AI tools to facilitate this process. The research design is carefully selected to ensure that it aligns with the overarching research questions and hypotheses, providing a robust foundation for data-driven insights.

Firstly, the study incorporates a detailed examination of existing approaches to IoT software development and traditional automation techniques. By contrasting these methodologies with modern AI-based solutions, the research identifies key areas where AI can significantly enhance automation efficiency. This comparative analysis informs experimental design, highlighting specific processes and tasks within IoT software development that are prime candidates for automation.

The selection of AI tools and technologies is based on their efficacy in automating software development processes. Advanced machine learning algorithms, neural networks, and natural language processing tools are among the AI technologies employed. These tools are selected for their proven capabilities in optimizing development workflows, reducing human intervention, and improving scalability and efficiency. The choice of these tools is further justified by their widespread adoption and success in similar domains, ensuring their relevance and potential impact.

Sample selection for the experiments is another critical aspect of the design process. The study includes a diverse range of IoT development projects, varying in complexity, scale, and application domains. This diversity ensures that the findings are generalizable and applicable across different IoT contexts. The selection process involves collaborating with partners industry and academic institutions to access a broad spectrum of IoT projects, thereby enhancing the robustness and relevance of the experimental results.

Data collection tools and methods are meticulously chosen to capture comprehensive and accurate data. Automated data logging systems, real-time monitoring tools, and software analytics platforms are utilized to gather data on various aspects of the development process. These tools enable continuous tracking of development activities, performance metrics, and process outcomes, providing a rich dataset for subsequent analysis. The data collection protocols are designed to minimize interference with the development processes, ensuring that the collected data accurately reflects the impact of AI-driven automation.

The experimental design also includes the creation of controlled environments to test specific hypotheses related to process automation. These controlled

environments allow for the isolation of variables and the systematic manipulation of AI tools and techniques to observe their effects on development efficiency and outcomes. Rigorous protocols are followed to ensure consistency and reproducibility across different experimental runs, thereby enhancing the reliability of the findings.

The data analysis phase involves applying sophisticated analytical techniques to interpret the results from the experiments. Statistical methods, machine learning models, and qualitative analysis tools are used to extract meaningful insights from the collected data. The analysis focuses on evaluating the effectiveness of AI-driven automation in improving development processes, comparing the results with traditional methods, and identifying trends and patterns that inform future research directions.

Ensuring the verification and validation of automated processes is a crucial aspect of experimental design. Verification techniques involve checking that the AI tools and automation workflows function as intended, while validation ensures that the automated processes achieve the desired outcomes in real-world scenarios. This dual approach guarantees that the findings are both theoretically sound and practically applicable.

In summary, the experimental design for process automation in AI-based IoT software development is a comprehensive and systematic approach that combines traditional methodologies with innovative AI tools. The meticulous planning and execution of the experiments ensure the reliability and validity of the findings, providing valuable insights into the potential of AI-driven automation to enhance the efficiency and scalability of IoT software development.

Data Acquisition and Management

Data acquisition and management are pivotal stages in the experimental setup for AI-based IoT software development. These stages ensure the systematic collection and efficient handling of data, which is crucial for validating the hypotheses and achieving reliable results.

To initiate the data acquisition process, a detailed plan was laid out to capture pertinent data from various sources within IoT environments. The data sources included IoT devices, sensors, gateways, and auxiliary systems that provide contextual

information. Each source was selected based on its relevance to the software processes being automated. Specific variables of interest were identified, including sensor readings, device performance metrics, network traffic data, and user interaction logs. This inclusivity ensures a comprehensive dataset that supports extensive analysis.

Data collection tools were meticulously chosen to facilitate accurate and high-frequency data capture. IoT data platforms such as AWS IoT Core, Microsoft Azure IoT Hub, and Google Cloud IoT Core were employed due to their robust capabilities in handling large-scale data streams. These platforms offer seamless integration with a variety of IoT devices and support real-time data ingestion, which is vital for dynamic IoT environments. Additionally, custom scripts and APIs were developed to interface with proprietary devices, ensuring no data source was overlooked.

The integrity of the acquired data was maintained through rigorous data management protocols. These protocols include real-time data validation checks to detect and correct errors as they occur. Data redundancy techniques, such as distributed storage solutions and regular backups, were implemented to safeguard against data loss. The use of cloud-based storage ensured scalability and accessibility, enabling the handling of growing data volumes over the course of the experiments.

Furthermore, metadata management was incorporated to provide context to the raw data. Metadata includes information about data origin, acquisition time, and the conditions under which data was collected. This contextual information is critical for interpreting results accurately and for conducting subsequent data analysis.

Data management also involved the organization and preprocessing of the acquired data to prepare it for analysis. Data preprocessing steps included cleaning (removing noise and irrelevant information), normalization (scaling data to a standard range), and transformation (converting data into a suitable format for analysis). Advanced preprocessing techniques, such as data augmentation and feature extraction, were employed to enhance the quality of the dataset and to uncover deeper insights.

The rationale for the chosen data acquisition and management methods lies in their proven effectiveness and efficiency in IoT environments. Cloud-based data platforms offer unmatched scalability and integration capabilities, making them ideal

for handling the diverse and voluminous data produced by IoT systems. Real-time validation and redundancy practices ensure data integrity, which is crucial for producing reliable experimental results. Moreover, comprehensive preprocessing enhances data quality, enabling more accurate and insightful analysis.

The advantages of this approach include the ability to handle large datasets efficiently, ensuring data integrity and security, and providing a robust foundation for subsequent analytical processes. By leveraging cloud technologies, the research guarantees high availability and fault tolerance, critical factors in maintaining continuous data streams from IoT devices.

In summary, the data acquisition and management methods implemented in this study are designed to capture, store, and process data effectively, ensuring high data quality and reliability. These processes form the backbone of the experimental setup and are essential for achieving the study's objectives in evaluating the impact of AI on automating IoT software development processes.

6.4.3 Data Analysis and Interpretation

Analytical Techniques for Software Development Processes

The integration of artificial intelligence (AI) in the automation of software development processes in the Internet of Things (IoT) sector necessitates robust analytical techniques. This section delineates the methods employed in analyzing these processes, ensuring the efficiency and efficacy of AI-driven solutions. Each analytical technique is meticulously chosen to align with the research objectives and hypotheses surrounding the automation of IoT software development [124, 125].

Primarily, the study employs both qualitative and quantitative analytical methods to provide a comprehensive evaluation. These mixed-method approaches facilitate a nuanced understanding, incorporating the strengths of both paradigms. Qualitative analysis allows for an in-depth exploration of patterns and processes in software development, while quantitative techniques provide statistical validation and scalability.

In qualitative analysis, thematic analysis is utilized to identify recurring themes and patterns in the software development lifecycle as influenced by AI tools. This

involves coding the data obtained from experimental setups and interviews with domain experts. By organizing the data into themes such as error reduction, process efficiency, and user satisfaction, the study extracts meaningful insights into the qualitative impact of AI [140].

Quantitative analysis is conducted using statistical tools to measure the performance metrics of AI-integrated software development processes. Key performance indicators (KPIs) such as development time, error rates, and resource utilization are quantified. Descriptive statistics provide foundational information, while inferential statistics, including regression analysis and hypothesis testing, are used to ascertain the significance of AI-driven improvements.

The research employs automated code analysis tools to assess the quality and reliability of software products generated during the experiments. Static code analysis methods are applied to examine the source code for compliance with coding standards and identification of potential issues. Furthermore, dynamic analysis techniques, including runtime verification and validation, are integrated to ensure that the automated processes adhere to expected behaviors and performance benchmarks.

Machine learning algorithms, pivotal in this research, are analyzed using performance metrics specific to each algorithm. Confusion matrices, accuracy scores, precision, recall, and F1 scores are calculated to evaluate classification algorithms. For regression models, they mean absolute error (MAE), mean squared error (MSE), and R-squared values are employed. These metrics enable a precise understanding of the algorithms' effectiveness in automating IoT software processes.

Graphical representations such as bar charts, line graphs, and scatter plots are extensively used to visualize the analytical results. These visual aids are instrumental in comparing traditional and AI-driven methods, illustrating the enhancements brought by AI integration. Tableau and MATLAB are among the visualization tools employed to generate these graphical summaries, ensuring clarity and accessibility of the data insights.

The interpretation of results is grounded in comparing AI-driven automation with traditional software development methodologies. Benchmarking against industry

standards provides a realistic context for evaluating improvements. This comparative analysis is critical in underscoring the advantages of AI integration, demonstrating tangible benefits like reduced developmental lead times and enhanced software reliability.

Ensuring the validity and reliability of automated processes encompasses rigorous verification methods. Cross-validation techniques are applied to machine learning models to prevent overfitting and validate predictive performance. Additionally, peer reviews and expert consultations are conducted to corroborate the study's findings and analytical approaches.

In conclusion, the comprehensive application of both qualitative and quantitative analytical techniques yields a robust evaluation of the software development processes. The meticulous approach ensures that the integration of AI in automating IoT software development is rigorously analyzed, providing substantial evidence for its efficacy. This analysis not only affirms the potential of AI-driven methodologies but also sets a precedent for future research in the field, directing subsequent investigations towards unexplored dimensions of AI in IoT automation.

Interpreting Results from AI Algorithms

The interpretation of results derived from AI algorithms is a crucial aspect of this research, involving multiple layers of analysis to ensure that the outcomes are not only accurate but also relevant to the goals of automating IoT software development. The primary objective is to understand the efficacy of AI in automating various development processes, thereby validating improvements over traditional methods.

The methods employed in this research include a comprehensive set of experiments designed to leverage state-of-the-art AI algorithms. These experiments are structured to assess specific software development tasks within the IoT domain. Raw data collected from these experiments underwent rigorous preprocessing to ensure quality and consistency. Tools such as data normalization and cleansing techniques are applied to mitigate any discrepancies or noise present in the dataset. This ensures that subsequent analysis is based on high-quality, reliable data.

The choice of algorithms is foundational to this study. Various machine learning models, including supervised and unsupervised learning algorithms, are employed to automate distinct aspects of software development. For instance, supervised learning algorithms like decision trees and support vector machines are used for tasks such as code generation and bug detection. The selection of these algorithms is based on their robustness and proven effectiveness in similar contexts, ensuring that the results obtained are both reliable and replicable.

Data analysis involves evaluating the performance metrics of these algorithms. Key performance indicators (KPIs) such as accuracy, precision, recall, and F1-score are calculated to measure how well the AI models perform in automating IoT software development tasks. These metrics provide a quantitative basis for comparing the efficiency of AI-driven methods against traditional approaches. For example, an increase in recall and precision in bug detection implies a more efficient and robust development process.

Interpreting these results involves a multifaceted approach. Statistical methods and visualizations, including graphs and charts, are utilized to present the data in a comprehensible manner. Tools like confusion matrices and ROC curves offer visual insights into the performance of classification algorithms, highlighting their strengths and potential areas for improvement. Additionally, qualitative analysis is conducted to gather insights from the development process itself, examining how automation impacts various stages of software development, from design to deployment.

Verification and validation are integral components of this interpretative process. Cross-validation techniques are applied to ensure that the results are generalizable and not overfitted to a specific dataset. Sensitivity analysis is also performed to understand the robustness of the AI models under varying conditions and data inputs. This helps in ensuring that the proposed automation methods are reliable across different scenarios and can be scaled effectively.

The results demonstrate a noticeable improvement in automation efficiency when AI algorithms are applied to IoT software development. For instance, the automation of code generation processes shows a significant reduction in development

time, while maintaining or even enhancing code quality. Similarly, AI-driven bug detection methods outperform traditional manual methods, with higher accuracy and faster identification rates. These findings are aligned with the hypothesis that AI can significantly optimize IoT software development processes.

This comprehensive analysis confirms the potential of AI in transforming IoT software development. By reducing human intervention and streamlining various development tasks, AI not only enhances efficiency but also allows developers to focus on more strategic and creative aspects of their work. The implications of these findings are far-reaching, suggesting that future software development practices in the IoT domain can greatly benefit from further integration of AI technologies.

In summary, the interpretative analysis of AI algorithm results underscores the transformative impact of AI on IoT software development. The rigorous, supported by statistical validations and qualitative insights, ensures that the findings are robust and applicable to real-world scenarios. This study thus provides a solid foundation for future research and practical applications aimed at automating IoT software development processes.

Verification and Validation of Automated Processes

Verification and validation (V&V) are critical components in the evaluation of automated processes in AI-based IoT software development. This ensures that the automated procedures not only function correctly but also meet the predetermined quality and performance standards [128].

Verification focuses on determining whether the automation processes were correctly implemented. This involves a meticulous examination of the AI algorithms used for automation, ensuring these algorithms were encoded and executed as intended. The research methodology employs systematic testing, utilizing unit tests and integration tests within controlled environments. These tests are designed to rigorously assess the AI models against a variety of conditions and edge cases, ensuring the stability and reliability of the automation process.

In the data acquisition phase, robust protocols are put in place to manage the integrity and quality of the data sourced. High-quality datasets are essential for training

and validating AI models. The selection criteria for these datasets include representativeness, completeness, and accuracy. For instance, data collected from IoT sensors must be thoroughly vetted to avoid anomalies that could skew the training of AI algorithms. The use of cross-validation techniques further enhances the reliability of these models by providing repeated assessments that minimize biases and overfitting.

Validation, on the other hand, confirms whether the automated processes achieve the intended outcomes and meet the user requirements in real-world scenarios. This involves conducting live trials within operational IoT environments to observe the automation in action. During these trials, key performance indicators (KPIs) such as processing time, resource utilization, error rates, and user satisfaction are meticulously measured. These KPIs provide quantitative measures that reflect the efficiency and effectiveness of the automation.

In the experimental setup, a suite of AI tools is deployed to automate specific software development tasks such as code generation, bug detection, and performance optimization. Each tool undergoes validation through benchmark comparisons against traditional methods to demonstrate relative improvements. For example, the use of machine learning models to identify and fix software bugs is validated by comparing the model's performance with human debugging efforts in terms of accuracy and speed.

Statistical analysis plays a pivotal role in the verification and validation process. Techniques such as hypothesis testing, confidence interval estimation, and regression analysis are employed to interpret the results. For instance, before-and-after studies may be conducted to evaluate the impact of automation on development cycle times. The gathered data is statistically analyzed to determine whether observed improvements are significant and not due to random variation.

Additionally, to ensure thorough validation, feedback loops involving domain experts are incorporated. These experts review the automated outputs, providing critical insights on the practicality and usability of the solutions generated by AI. Their evaluations are systematically recorded and analyzed to identify areas for further refinement, thus creating a continuous improvement cycle.

To conclude, the rigorous V&V process embedded in this research ensures that AI-driven automation in IoT software development is not only scientifically sound but also practically viable. These meticulous processes underpin the research's credibility, providing solid evidence that the integration of AI can significantly enhance the automation of IoT software development. As the field evolves, the established V&V framework will serve as a robust model, guiding future endeavors towards achieving higher levels of automation efficiency and reliability.

6.5. Conclusion

6.5.1 Summary of Key Findings and Contributions

This research aims to address the pressing need for automation in the Internet of Things (IoT) software development domain by leveraging advancements in artificial intelligence (AI). The primary objective is to identify the key areas within the IoT software development lifecycle where AI can be effectively integrated to optimize workflows and reduce human intervention. This study fills a significant gap in the existing body of knowledge by systematically exploring and validating the potential of AI-driven automation methodologies specifically tailored for IoT applications.

The findings from this research reveal several critical insights into the efficacy and impact of AI on the automation of IoT software development processes. A comprehensive comparison between traditional software development practices and AI-enhanced methodologies demonstrates that AI integration leads to substantial improvements in efficiency and scalability. The study highlights that AI techniques, including algorithms and machine learning models, can significantly streamline various stages of software development, from design and coding to testing and deployment.

One of the most important outcomes of this research is the empirical evidence supporting the hypothesis that AI can enhance automation in IoT software development. The experimental results indicate that AI tools can automate repetitive

and time-consuming tasks, thereby freeing up human developers to focus on more complex and innovative aspects of development.

The analysis further underscores the significance of effective data acquisition and management as critical effective components of successful AI integration. The experiments designed in this study utilized state-of-the-art AI tools to automate specific development processes, with rigorous data protocols ensuring the reliability of the results. These findings are consistent with the theoretical framework, which emphasizes the foundational principles of integrating AI with IoT software development and the necessity of robust automation frameworks.

The study's contributions extend beyond empirical findings, offering practical insights into best practices for AI integration in IoT development. By benchmarking against industry standards and peer studies, the research identifies trends and provides recommendations for successfully implementing AI-driven automation strategies. These insights are valuable for practitioners and researchers alike, as they highlight both the potential and the challenges associated with AI automation in the IoT sector.

The study's limitations are also acknowledged, including the need for further research to address potential challenges and improve the robustness of AI models in varied IoT contexts. Future research directions proposed in this study suggest deeper exploration into specific AI techniques that could further enhance automation efficacy, as well as longitudinal studies to evaluate the long-term impact of AI integration on IoT software development.

In conclusion, this research makes significant contributions to the field of IoT software development by presenting a novel approach to automation through AI. The study's findings demonstrate the transformative potential of AI, providing a blueprint for future advancements and setting the stage for continued innovation in the domain. The clear and concrete outcomes of this research address the initial hypotheses and objectives, offering both theoretical and practical value to the ongoing dialogue on AI and IoT integration.

6.5.2 Recommendations for Future Research

This study aimed to investigate the automation of IoT software development processes through the application of artificial intelligence (AI). Central to this investigation was the need to identify critical areas within these processes that can be optimized by AI to enhance efficiency and scalability, thereby reducing human intervention. The research was designed to fill a noted knowledge gap wherein traditional software practices often struggle to meet the dynamic requirements of the IoT domain, particularly in terms of automation [134, 135].

The primary findings of this research highlight the significant potential AI holds in transforming IoT software development. Key contributions include the identification of specific AI techniques-such as machine learning algorithms-that can automate repetitive and time-consuming tasks, thereby streamlining development workflows. The comparative analysis between traditional and AI-driven methods clearly demonstrates that AI applications not only improve efficiency but also allow for more complex problem-solving capabilities within the IoT environment.

With these findings in mind, several recommendations can be made to guide future research in this burgeoning field. Firstly, further studies should explore the scalability of AI-driven automation techniques in live IoT environments. While current experiments have shown promising results in controlled settings, real-world applications often present unforeseen challenges. It is crucial to validate AI methodologies in diverse, real-time IoT deployments to assess their robustness and adaptability.

Secondly, future research should focus on developing hybrid models that integrate AI with other emerging technologies such as blockchain and edge computing. The combination of these technologies could potentially offer more secure, efficient, and decentralized solutions to IoT software development. This interdisciplinary approach could accelerate the innovation cycle and address more complex issues inherent in IoT ecosystems.

Moreover, as AI techniques continue to evolve, there is a need to examine the ethical implications and data privacy concerns associated with their deployment.

Future studies should not only focus on technological advancements but also on creating frameworks that ensure ethical standards and compliance with data protection regulations. This is particularly relevant as the integration of AI in IoT involves vast amounts of sensitive data that need to be managed responsibly.

Another critical area for future research is the development of standard benchmarks and metrics to evaluate the effectiveness of AI-driven automation in IoT software development. While this research provides initial comparative insights, standardized evaluation criteria would enable more consistent assessments across different studies, facilitating better comparisons and fostering a cumulative knowledge base.

Additionally, there is a need for longitudinal studies to understand the long-term impact of AI on software development processes. These studies should track the performance of AI-enabled automation over extended periods to identify any potential degradation of efficiency or unforeseen complications that may arise. Such insights would be invaluable for refining AI models and ensuring their sustained efficacy.

Finally, future research should also investigate the collaborative roles of humans and AI in IoT software development. Understanding how AI can augment human decision-making and creativity, rather than merely replacing human involvement, could unlock new dimensions of productivity and innovation. Studies should explore optimal interaction frameworks where AI tools complement human expertise, leading to more intuitive and efficient development processes.

In conclusion, this study has laid a foundation for understanding the transformative potential of AI in automating IoT software development processes. Moving forward, addressing the recommendations outlined will not only strengthen the application of AI in this field but also contribute to the broader objective of achieving more efficient, scalable, and intelligent IoT solutions. Future research, grounded in these recommendations, will be crucial in navigating the complexities and maximizing the benefits of AI-driven innovations in IoT.

REFERENCES

1. Apache Kafka Development Team. (2024). Apache Kafka Flow Control and Backpressure: Producer and Consumer Configuration Guide. Apache Software Foundation, version 3.5. URL: https://kafka.apache.org/documentation/#producerconfigs.

2. Berg, B., Pesterev, A., Daniel, J., et al. (2020). The CacheLib Caching Engine: Design and Experiences at Scale. 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 753–768. URL: https://www.usenix.org/conference/osdi20/presentation/berg.

3. Berger, D. S., Hensley, N., Alamgir, A., et al. (2021). Kangaroo: Caching Billions of Tiny Objects on Flash. Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP), 243–262. DOI: 10.1145/3477132.3483568.

4. Brewer, E. A. (2000). Towards robust distributed systems. Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing (PODC '00). New York, NY: Association for Computing Machinery, p. 7. DOI: 10.1145/343477.343502.

5. ByteByteGo. (2024). The Sidecar Pattern Explained: Decoupling Operational Features. Technical Blog. URL: https://blog.bytebytego.com/p/the-sidecar-pattern-explained-decoupling.

6. Chen, L., Zhang, L., Li, Y., et al. (2024). Domain-Driven Design for Microservices: An Evidence-Based Investigation. IEEE Transactions on Software Engineering. DOI: 10.1109/TSE.2024.3385835.

7. Cloud Native Computing Foundation. (2024). CNCF Service Mesh Performance Benchmark Report 2024: Comparative Analysis of Istio, Linkerd, and Consul Connect. CNCF Technical Advisory Group. URL: https://www.cncf.io/reports/service-mesh-performance-benchmark-2024/.

8. Contentstack Team. (2024). Probing the future of microservices: Software trends in 2024. Contentstack Blog. URL: https://www.contentstack.com/blog/composable/the-future-of-microservices-software-trends-in-2024.

9. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms. 4th ed. Cambridge, MA: MIT Press. 1312 p. ISBN: 978-0262046305.

10. Einziger, G., Friedman, R., & Manes, B. (2017). TinyLFU: A Highly Efficient Cache Admission Policy. ACM Transactions on Storage, 13(4), 1–31. DOI: 10.1145/3149371.

11. Envoy Proxy Contributors. (2024). Envoy Proxy Documentation: Load Balancing. Cloud Native Computing Foundation. URL: https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/upstream/load_balancing/load_balancing.

12. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley Professional. 416 p. ISBN: 978-0201633610.

13. GraalVM Team. (2024). GraalVM Native Image: Performance Characteristics and Memory Optimization. Oracle Labs. URL: https://www.graalvm.org/latest/reference-manual/native-image/.

14. IMARC Group. (2024). Microservices Architecture Market Share, Size 2025-2033. Market Research Report. URL: https://www.imarcgroup.com/microservices-architecture-market.

15. Istio Authors. (2024). Istio Traffic Management. Google Cloud. URL: https://istio.io/latest/docs/concepts/traffic-management/.

16. Kolomiitsev, O. V., Bulba, S. S., & Nosko, S. V. (2025). Optimization method for sidecar operation under high loads – a multi-level approach to buffering and flow control. MODERNÍ ASPEKTY VĚDY v rámci publikační skupiny Scientific Publishing Group Svazek LVII mezinárodní kolektivní monografie. Czech Republic, p. 118. URL: https://doi.org/10.52058/57-2025.

17. Kolomiitsev, O. V., Bulba, S. S., Nosko, S. V., et al. (2025). Adaptive mathematical model of request routing graph in sidecar components. Grail of Science: Int. Sci. Journal, 57, 532–552. URL: https://doi.org/10.36074/grail-of-science.17.10.2025.057.

18. Kolomiitsev, O. V., Bulba, S. S., Nosko, S. V., et al. (2026). Evaluation of the efficiency of improved and developed mathematical models and the method of adaptive request processing. Grail of Science, 61, 742-762. https://doi.org/10.36074/grail-of-science.23.01.2026.

19. Kolomiitsev, O. V., Bulba, S. S., & Nosko, S. V. (2025). Graph routing model in sidecar components with dynamic parameters. XIX International Scientific and Practical Conference of Master's Students and Postgraduates "Theoretical and Practical Research of Young Scientists". Kharkiv: NTU "KhPI", p. 30.

20. Kolomiitsev, O. V., Bulba, S. S., & Nosko, S. V. (2025). Method for increasing sidecar resilience to external loads. XII International Scientific and Technical Conference "Informatics, Control and Artificial Intelligence" (ICAI-2025). May 14–16, 2025. Kharkiv: NTU "KhPI", p. 57.

21. Kolomiitsev, O. V., Bulba, S. S., Nosko, S. V., et al. (2025). Improved mathematical model of adaptive data caching for a sidecar component. Grail of Science: Int. Sci. Journal, 59, 844–858. URL: https://doi.org/10.36074/grail-of-science.12.12.2025.097.

22. Kolomiitsev, O. V., Bulba, S. S., Nosko, S. V., et al. (2025). Proposals regarding data flow management in sidecars for microservice systems. Grail of Science: Int. Sci. Journal, 54, 555–573. URL: https://doi.org/10.36074/grail-of-science.18.07.2025.062.

23. Kolomiitsev, O. V., Bulba, S. S., Nosko, S. V., et al. (2025). Proposals regarding sidecar operation under high loads. Grail of Science: Int. Sci. Journal, 53, 606–616. URL: https://doi.org/10.36074/grail-of-science.20.06.2025.074.

24. Kolomiitsev, O. V., Bulba, S. S., Nosko, S. V., et al. (2025). Proposals regarding the effective application of HTTP request filters in sidecar components. Grail of Science: Int. Sci. Journal, 55, 502–524. URL: https://doi.org/10.36074/grail-of-science.22.08.2025.059.

25. Kolomiitsev, O. V., Bulba, S. S., & Nosko, S. V. (2025). Features of building an additional communication system layer in microservice architecture. Current problems of the activity of components of the security and defense sector of Ukraine: Abstracts of the International Scientific and Practical Conference. NANGU. March 20, 2025, p. 885.

26. Kolomiitsev, O. V., Bulba, S. S., & Nosko, S. V. (2026). Approaches to applying machine learning methods for predicting optimal sidecar component load parameter values based on accumulated metric analysis. Collection of Abstracts of the Scientific Conference. (In press).

27. Kolomiitsev, O. V., Bulba, S. S., & Nosko, S. V. (2026). Recommendations regarding the deployment of sidecar components in industrial environments on the Kubernetes platform considering optimal RAM and CPU resource distribution. Materials of the Scientific and Technical Conference. (In press).

28. Lagar-Cavilla, A., Ahn, J., Balasubramanian, A., et al. (2019). Software-Defined Far Memory in Warehouse-Scale Computers. Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 317–330. DOI: 10.1145/3297858.3304053.

29. Liu, S., Ahmad, F., Vigfusson, Y., et al. (2021). Multi-Tenant Memory Caching: Balancing Performance, Fairness and Resource Allocation. Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 537–551. URL: https://www.usenix.org/conference/nsdi21/presentation/liu-shaohua.

30. Manes, B. (2024). Caffeine: A high performance caching library for Java. GitHub. URL: https://github.com/ben-manes/caffeine.

31. Microsoft Learn. (2024). Sidecar pattern - Azure Architecture Center. Microsoft Documentation. URL: https://learn.microsoft.com/en-us/azure/architecture/patterns/sidecar.

32. Newman, S. (2021). Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol, CA: O'Reilly Media. 616 p. ISBN: 978-1492034025.

33. Nosko, S. V. (2025). Adaptive model of sidecar components for ensuring stability: Backpressure, Exponential Backoff, and dynamic queues. Information Processing Systems, 3(182), 60–69. URL: https://doi.org/10.30748/soi.2025.182.06.

34. Nosko, S. V. (2025). Adaptive routing in sidecar components: integration of patterns, graphs, and dynamic parameters. Collection of Scientific Works of KNUAF, 3(85), 52–61. URL: https://doi.org/10.30748/zhups.2025.85.06.

35. Nosko, S. V. (2025). Experimental validation of sidecar component optimization models in microservice architectures. Armament Systems and Military Equipment, 3(83), 53–60. URL: https://doi.org/10.30748/soivt.2025.83.07.

36. Nosko, S. V. (2025). Research of methods and tools for software load testing using AI. XXXIII International Scientific and Practical Conference MicroCAD-2025. Kharkiv: NTU "KhPI", p. 1627.

37. Nosko, S. V., Bulba, S. S., & Kolomiitsev, O. V. (2025). Method of high-efficiency sidecar component implementation with minimal system resource costs. Control, Navigation and Communication Systems, 2(80), 161–168. URL: https://doi.org/10.26906/SUNZ.2025.2.161.

38. Nosko, S. V., & Bulba, S. S. (2023). Models and methods of interaction between distributed microservice systems. XVII International Scientific and Practical Conference of Master's Students and Postgraduates "Theoretical and Practical Research of Young Scientists". Kharkiv, p. 102.

39. Nosko, S. V., Bulba, S. S., & Semerenko, Yu. O. (2023). Cloud technologies and their application. XXXI International Scientific and Practical Conference MicroCAD-2023. Kharkiv: NTU "KhPI", p. 1191.

40. Nosko, S. V., Kolomiitsev, O. V., & Bulba, S. S. (2025). Experimental validation of the adaptive flow management method with three-level back pressure architecture. Information Processing Systems, 4(183), 15–22. URL: https://doi.org/10.30748/soi.2025.183.02.

41. Nosko, S. V., Bulba, S. S., Kolomiitsev, O. V., & Soloviova, O. I. (2024). Means of building an additional communication system layer in microservice architecture. Grail of Science: Int. Sci. Journal, 46, 651–659. URL: https://doi.org/10.36074/grail-of-science.29.11.2024.084.

42. Nosko, S. V., Bulba, S. S., Kolomiitsev, O. V., et al. (2025). Proposals regarding authorization in the sidecar component of microservice architecture. Control, Navigation and Communication Systems. Poltava: Poltava National Technical University named after Kondratyuk, 1(7), 116–123. URL: https://doi.org/10.26906/SUNZ.2025.1.116-123.

43. Pahl, C., Jamshidi, P., & Zimmermann, O. (2018). Architectural Principles for Cloud Software. ACM Transactions on Internet Technology, 18(2), 1–23. DOI: 10.1145/3104028.

44. Prometheus Team. (2024). Prometheus Monitoring: Metrics Collection and Analysis for Cloud-Native Applications. version 2.48. URL: https://prometheus.io/docs/prometheus/latest/.

45. Red Hat Quarkus Team. (2024). Quarkus Framework Performance Analysis: Native Compilation vs JVM in Containerized Environments. Red Hat Developer Documentation, version 3.6. URL: https://quarkus.io/guides/performance-measure.

46. Ralston A. Encyclopedia of computer science / A. Ralston, E.D. Reilly, D. Hemmendinger // Grove's Dictionaries Inc. 2000.

47. Sharma S. Technology and Trends to Handle Big Data: Survey / S. Sharma, V. Mangat // IEEE, 2015. [Electronic resource]. – Access mode: https://doi.org/10.1109/ACCT.2015.121.

48. Wang L. Live streaming E-commerce platform characteristics: Influencing consumer value co-creation and co-destruction behavior / L. Wang, R.S. Zhang, C.X. Zhang // Acta psychologica, 2016. [Electronic resource]. – Access mode: https://doi.org/10.1016/j.actpsy.2024.104163.

49. Hantula D.A. Online Shopping as Foraging: The Effects of Increasing Delays on Purchasing and Patch Residence / D.A. Hantula, D.D. Brockman, C.L. Smith // IEEE Transactions on Professional Communication, 2008, 51(2):147-154. [Electronic resource]. – Access mode: https://doi.org/10.1109/TPC.2008.2000340.

50. Sheth A. Management of Interdependent Data: Specifying Dependency and Consistency Requirements / A. Sheth, M. Rusinkiewicz // IEEE, 2002. [Electronic resource]. – Access mode: https://doi.org/10.1109/MRD.1990.138261.

51. Santiago G. Extending an open source enterprise service bus for cloud data access support / G. Santiago // Uni Stuttgart. – Universitätsbibliothek, 2013.

52. Soui M. Deep Learning-Based Model Using DensNet201 for Mobile User Interface Evaluation / M. Soui, Z. Haddad // International journal of human-computer interaction, 2023(6/10):39. [Electronic resource]. – Access mode: https://doi.org/10.1080/10447318.2023.2175494.

53. Mamedov T. Static Analysis of Resource Consumption in Programs Using Rewriting Rules / T. Mamedov, A. Doroshenko, R. Shevchenko // IEEE Xplore, 2020. [Electronic resource]. – Access mode: https://doi.org/10.1109/ATIT50783.2020.9349290.

54. Shi W. Load balancing for parallel forwarding / W. Shi, M. Macgregor, P. Gburzynski // IEEE/ACM Transactions on Networking, 2005, 13(4):790-801. [Electronic resource]. – Access mode: https://doi.org/10.1145/1088742.1088749.

55. Anupama O. Load Balancing the Network Traffic in the N~(th) Mode of Iptables / O. Anupama, P. Gupta // Oriental journal of computer science and technology: An international open access peer reviewed research journal, 2016, 9(2):146-152.

56. Nguyen D.H. Model-based testing of multiple GUI variants using GUI test generator / D.H. Nguyen, P.A. Strooper, S.J. Guy // ACM, 2010. [Electronic resource]. – Access mode: https://doi.org/10.1145/1808266.1808270.

57. 佚名.数说 十年风雨,电商从稚嫩走向成熟[J].中国商界, 2020(12):10.

58. Nagarajan M. A Multi-Period Model of Inventory Competition / M. Nagarajan, R. S. Rajagopalan // Social Science Electronic Publishing, - № 57(3), 2009. – Pp. 785-790. [Electronic resource]. – Access mode: https://doi.org/10.2139/ssrn.1140311.

59. Moise D. Luc Bougé.Improving the Hadoop map/reduce framework to support concurrent appends through the BlobSeer BLOB management system / D. Moise, G. Antoniu // ACM, 2010. [Electronic resource]. – Access mode: https://doi.org/10.1145/1851476.1851596.

60. Teranishi M. Computer system and storage system / M.Teranishi, H. Shibuya, S.Murayama et al // United States Patent: Pub. No.: US 2013/0212335 A1, 2013. [Electronic resource]. – Access mode: https://patentimages.storage.googleapis.com/84/cc/02/78cb2e5306e7bc/US20130212335A1.pdf.

61. Fan P. PC: a distributed transaction concurrency control protocol of multi-microservice based on cloud computing platform / P. Fan, J. Liu, W. Yin et al // Journal of Cloud Computing, 2020, 9(1):22. [Electronic resource]. – Access mode: https://doi.org/10.1186/s13677-020-00183-w.

62. Liu Y. Research achievements on the new generation Internet architecture and protocols / Y. Liu, J.P. Wu, Z. Zhang et al., 2013(11):25. [Electronic resource]. – Access mode: https://doi.org/10.1007/s11432-013-5021-4.

63. Bai Y. Dynamic reverse proxy chain generation for networks in data centers / Y.Bai, G.Guo, Y.Wang et al // 会议论文, 2020, 000(9226588). [Electronic resource]. – Access mode: https://doi.org/10.23919/APNOMS50412.2020.9237006.

64. Min H. Research and application of web cache technology / H. Min, Z. Wei-Dong, L.I. Zhong-Li // Computer Engineering and Design, 2003.

65. Bakshi K. Microservices-based software architecture and approaches / K. Bakshi //2017 IEEE Aerospace Conference.IEEE, 2017. [Electronic resource]. – Access mode: https://doi.org/10.1109/AERO.2017.7943959.

66. Che Z. Swarm intelligence-based optimization of machine learning models for diverse industrial problems: a comparative study / Z. Che, C. Peng, J. Wang // IOP Publishing Ltd, 2024. [Electronic resource]. – Access mode: https://doi.org/10.1088/1742-6596/2852/1/012001.

67. Lee J. Distributed Transaction Management Using Optimization Of Local Transactions / J. Lee, M. Muehle, J. Noh // United States Patent: Pub. No.: US20120167098A1, 2025. [Electronic resource]. – Access mode: https://doi.org/US20120167098 A1.

68. Casalaina M.C. Delta caching / M.C. Casalaina // BUSTECH 2014: The Fourth International Conference on Business Intelligence and Technology. [Electronic resource]. – Access mode: https://www.google.com/patents/US7406514 [Jul, 2008].

69. Kolomiitsev, O., Hulevych, M., Chekunova, O., Brechko, V., Yachna, I., Vasylets, D., Berdochnyk, A., Lukianov, D., Pidlisnyi, O., Feldi, S., & Kuvshynova O. (2025). Features of approaches to automated generation of test scenarios. Grail of Science, 57, 576–596. https://doi.org/10.36074/grail-of-science.17.10.2025.060

70. Hulevych, M. CIDER: Assisted automation tool for C++ libraries testing. Control, Navigation and Communication Systems, 2024. Vol. 2, No. 76. P. 74–77. https://doi.org/10.26906/sunz.2024.2.074

71. Kolomiitsev, O., Hulevych, M., Sobora, A., Lilchytskyi, V., Kolomiiets, B., Kharchenko, O., Yasynskyi, O., Bohdan, S., Kolomiiets, V., Tryfonenko, K., Khrol, L., & Shendryk, V. (2025). Features of using the developed CIDER tool for automating the testing of software libraries written in the C++ programming language. Grail of Science, 55, 476–488. https://doi.org/10.36074/grail-of-science.22.08.2025.057

72. Kolomiitsev, O., Hulevych, M., Krasnoshapka, I., Korobetskyi, O., Lernatovych, A., Berdochnyk, A., Chebakov, O., Chemerys, V., Shkorupskyi, V., Chebakova, O., & Vorobiov, O. (2025). Peculiarities of application of automated test generation methods for C++ software. Grail of Science, 54, 527–542. https://doi.org/10.36074/grail-of-science.18.07.2025.060

73. Hulevych, M., & Kolomiitsev, O. (2025). Automated test generation techniques for C++ software. Control, Navigation and Communication Systems, 2(80), 102–107. https://doi.org/10.26906/SUNZ.2025.2.102

74. Jehan, S., &Wotawa, F. (2023). An Empirical Study of Greedy Test Suite Minimization Techniques Using Mutation Coverage. IEEE Access, 11, 65427–65442. https://doi.org/10.1109/ACCESS.2023.3289073

75. Puterman, M. L. (1994). Markov decision processes: Discrete stochastic dynamic programming. John Wiley & Sons. https://doi.org/10.1002/9780470316887

76. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press.

77. Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. Machine Learning, 8, 323–339. https://doi.org/10.1007/BF00992700

78. Li, L., Walsh, T. J., & Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics (ISAIM 2006, Fort Lauderdale, Florida, USA).

79. Hulevych M. (2025). Evaluation of the effectiveness of the test scenarios forming method for C++ libraries based on a Q-learning agent. Management Information Systems and Devises, (187), 20–46. https://doi.org/10.30837/0135-1710.2025.187.020

80. Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. The Journal of Machine Learning Research, 10, 1633–1685. https://dl.acm.org/doi/10.5555/1577069.1755839

81. Semenov S., Kolomiitsev O., Hulevych M., Mazurek P., & Chernyk O. (2025). An Intelligent Method for C++ Test Case Synthesis Based on a Q-Learning Agent. Applied Sciences, 15(15), 8596. https://doi.org/10.3390/app15158596

82. Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research, 4, 237–285. https://doi.org/10.1613/jair.301

83. Kolomiitsev, O., Hulevych, M., Melenti, D., Porynos, Y., Kamenetskyi, V., Kostashchuk, M., Sapon, V., Surhai, V., Kushnir, V., Riazantsev, S., Khrol, L., & Salohor, V. (2025). Method for forming test scenarios formation for software libraries written in the C++ programming language based on a Q-learning agents. Grail of Science, 56, 329–352. https://doi.org/10.36074/grail-of-science.19.09.2025.041

84. Коломійцев О.В., Комаров В.О., Катунін А.М., Рудаков І.С., Балабуха О.С., Качуровський Г.М., Ковтунов А.Л., Жирна О.В., Косенко Г.П., Максимов М.О. Особливості застосування безпілотних літальних апаратів коптерного типу. Theoretical and empirical scientific research: concept and trends: Collection of scientific papers «ΛΌГОΣ» with Proceedings of the VII International Scientific and Practical Conference, Oxford, August 16, 2024. Oxford-Vinnytsia: P.C. Publishing House & UKRLOGOS Group LLC, 2024. С. 188-198. DOI 10.36074/logos-16.08.2024.

85. Коломійцев О.В., Дмітрієв О.М., Комаров В.О., Катунін А.М., Рудаков І.С., Помогаєв, І.В., Барвінок Р.Д., Стригун В.В., Телевна Т.П., Братко В.В. Особливості застосування нейроної мережі у системі керування групою безпілотних літальних апаратів. ГРААЛЬ НАУКИ: міжнар. наук. журнал. – Вінниця: ГО «Європейська наукова платформа»; НУ «Інститут науково-технічної інтеграції та співпраці», 2024. – No 43. – С. 322-334. DOI 10.36074/grail-of-science.06.09.2024.

86. Коломійцев О.В., Комаров В.О., Катунін А.М., Колодій О.П., Помогаєв І.В., Сапон В.І., Кравченко В.Л., Рудаков І.С., Бєсова А.О., Мосійчук А.М. Пропозиції щодо розробки програмного забезпечення для виявлення людей на цифрових аерофотознімках та потоковому відео комп'ютерного зору безпілотного літального апарату. ГРААЛЬ НАУКИ: міжнар. наук. журнал. – Вінниця: ГО «Європейська наукова платформа»; НУ «Інститут науково-технічної інтеграції та співпраці», 2024. – No 45. – С. 411-420. DOI 10.36074/grail-of-science.01.11.2024.

87. Коломійцев О.В., Комаров В.О., Катунін А.М., Рудаков І.С., Любченко О.В., Слободенюк Ю.В., Костащук М.М., Хома В.В., Богдан С.І., Шендрик В.І. Особливості керування польотами безпілотних літальних апаратів в умовах міської забудови на основі технологій машинного та глибокого навчання. ГРААЛЬ НАУКИ: міжнар. наук. журнал. – Вінниця: ГО «Європейська наукова платформа»; НУ «Інститут науково-технічної інтеграції та співпраці», 2024. – No 46. – С. 677-686. DOI 10.36074/grail-ofscience.29.11.2024.

88. Коломійцев О., Комаров В., Філіппенков О. Проведення дефектоскопічного контролю високорозташованих конструкцій планера літального апарату за допомогою безпілотного літального апарата. II Міжнародна науково-практична конференція «Creation of new ideas of learning in modern conditions», 25 вересня 2023, Бордо, Франція. 277-283. URL: https://eu-conf.com/events/creation-of-new-ideas-oflearning-in-modern-conditions/.

89. Рисований, О., Коломійцев, О., Альошин, Г., Дмітрієв, О., Дранко, А., Третяк, В., Крук, Б., Старцев, В., Калачова, В., & Пустоваров, В. (2023). Метод підвищення ефективності управління безпілотним літальним апаратом на основі використання нелінійного псевдовипадкового генератора. Scientific Collection «InterConf», (149), 282-290. Retrieved from https://archive.interconf.center/index.php/conferenceproceeding/article/view/2928.

90. Рисований О.М., Коломійцев О.В., Дмітрієв О.М., Голубничий Д.Ю., Невиніцин А.М., Третяк В.Ф., Крук Б.М., Романюк А.О., Собора А.І., Старцев В.В., Калачова В.В., Пустоваров В.В, (2023). Метод підвищення ефективності управління безпілотним літальним апаратом у залежності від особливостей матриці зв'язків при використанні нелінійного псевдовипадкового генератора. II International Scientific and Practical Conference. Science and education in progress. Held on June 16-18, 2023 in Dublin, Ireland. P. 344-352.

91. Shmatko, O., Kolomiitsev, O., Rekova, N., Kuchuk, N., & Matvieiev, O. (2023). Designing and evaluating dl-model for vulnerability detection in smart contracts. Advanced Information Systems, 7(4), 41–51. https://doi.org/10.20998/2522-9052.2023.4.05.

92. Коломійцев О.В., Рудаков І.С., Філіппенков О.В., Павлій В.О., Катунін А.М., Коваль В.В., Згода Г.І., Кібальник В.М., Пальчиков В.В., Бєлоус М.В., Павлій Л.В. Особливості застосування технології машинного навчання для підвищення безпеки польотів групи безпілотних літальних апаратів у міський забудові. ГРААЛЬ НАУКИ: міжнар. наук. журнал. – Вінниця: ГО «Європейська наукова платформа»; НУ «Інститутнауково-технічної інтеграції та співпраці», 2025. – No 50. – С. 537-550. DOI 10.36074/grail-of-science.21.03.2025.

93. Коломійцев О.В., Рудаков І.С., Дмітрієв О.М., Комаров В.О., Катунін А.М., Руснак В.М., Пустоваров В.В., Дегтярьов А.В., Сапон В.І., Шеремет І.А., Галінський Д.О. Аналіз наукових досліджень у галузі безпеки польотів групи безпілотних літальних апаратів у міській забудові. ГРААЛЬ НАУКИ: міжнар. наук. журнал. – Вінниця: ГО «Європейська наукова платформа»; НУ «Інститут науково-технічної інтеграції та співпраці», 2025. – No 51. – С. 569-582. DOI 10.36074/grail-of-science.18.04.2025.

94. Коломійцев О.В., Рудаков І.С., Дмітрієв О.М., Власов А.В., Панченко В.І., Поринос Є.О., Фесюн О.В., Іваненко В.О., Романович М.І., Шеремет І.А., Галінський Д.О. Особливості застосування методу D2-C2 для підвищення безпеки польотів групи безпілотних літальних апаратів у міський забудові. ГРААЛЬ НАУКИ: міжнар. наук. журнал. – Вінниця: ГО «Європейська наукова платформа»; НУ «Інститут науково- технічної інтеграції та співпраці», 2025. – No 54. – С. 543-554. DOI 10.36074/grail-ofscience.18.07.2025.

95. Коломійцев О.В., Рудаков І.С., Дмітрієв О.М., Кібальник В.М., Слободенюк Ю.В., Соловйов А.А., Пальчиков В.В., Дзюба І.В., Жуйков В.В., Чекунов В.В., Фесенко К.В. Удосконалений метод планування траєкторій польоту безпілотного літального апарату з проактивним уникненням зіткнень з перешкодами у міський забудові. ГРААЛЬ НАУКИ: міжнар. наук. журнал. – Вінниця: ГО «Європейська наукова платформа»; НУ «Інститут науково-технічної інтеграції та співпраці», 2025. – No 56. – С. 382-397. DOI 10.36074/grail-of-science.19.09.2025.044.

96. Коломійцев О.В., Рудаков І.С., Дмітрієв О.М., Катунін А.М., Суркова К.В., Поринос Є.О., Богдан С.І., Кацуба Р.М., Кравчук В.В., Романович М.І., Кукла О.С., Коломієць В.В., Салогор В.В. Методологія дослідження питань щодо безпеки польотів групи безпілотних літальних апаратів у міському середовищі. ГРААЛЬ НАУКИ: міжнар. наук. журнал. – Вінниця: ГО «Європейська наукова платформа»; НУ «Інститут науково- технічної інтеграції та співпраці», 2025. – No 60. – С. 602-614. DOI 10.36074/grail-of-science.26.12.2025.065.

97. Шматко О.В,, Гамаюн І.П., Коломійцев О.В,, Чекунова О.М., Лисий О.О., Рудаков І.С. Модель для системи виявлення та класифікації об'єктів (транспортних засобів) у відеопотоці даних на основі глибинного навчання. ГРААЛЬ НАУКИ: міжнар. наук. журнал. – Вінниця: ГО «Європейська наукова платформа»; НУ «Інститут науково- технічної інтеграції та співпраці», 2026. – No 62. – С. 1056-1077. DOI 10.36074/grail-of-science.20.02.2026.115.

98. Коломійцев, О., Голубничий, Д., Коц, Г., Третяк, В., Євстрат, Д., & Лисиця, А. (2020). Задачі дискретної оптимізації та їх постановка для розміщення засобів захисту в розподіленій системі. Збірник наукових праць ΛΌГОΣ, 36-41. Збірник наукових праць ΛΌГОΣ, 36-41.

99. Третяк, В., Голубничий, Д., Коломійцев, О., Мегельбей, Г., Возний, О., & Філіпенков, О. (2020). Математична модель рангового підходу. Збірник наукових праць ΛΌГОΣ, 116-122. https://doi.org/10.36074/25.12.2020.v1.40

100. Голубничий Д. Ю. Інформаційна технологія відсікання неперспективних варіантів в алгоритмах рішення задачі цілочисельного лінійного програмування з булевими змінними на основі рангового підходу / Д. Ю. Голубничий, О. В. Коломійцев, В. Ф. Третяк та ін. // Theoretical foundations in research in Engineering : collective monograph. – Boston (USA), 2022. – С. 96–133.

101. Третяк В., Голубничий Д., Коломіцев О., Осієвський С., Воронін В., Місюра О. … Крамар О. (2024). Спосіб оптимального планування розподілом завдань, що забезпечує відмовостійкість функціонування інформаційно-комунікаційних систем з функціональним призначенням. InterConf+ , (50(221), 407–428. https://doi.org/10.51582/interconf.19-20.10.2024.041

102. Голубничий Д., Коломійцев О., Осієвський С., Третяк В., Рибальченко А., Любченко О., Головченко О. (2024). Метод відсікання безперспективних варіантів для задач цілочисельного лінійного програмування з булевими змінними з використанням рангового підходу. Науковий збірник «InterConf+» , (41(185), 526–555. https://doi.org/10.51582/interconf.19-20.01.2024.064

103. Коломійцев , О., Рибальченко , А., Голубничий , Д., Третяк , В., Воронін , В., Комаров , В., Пустоваров , В., Філіппенков , О., Любченко , О., & Рудаков , І. . (2023). Проблеми розвитку паральних обчислювальних систем та їх математичного забезпечення для рішення задач дискретної оптимізації. Грааль науки , (32), 182–197. https://doi.org/10.36074/grail-of-science.13.10.2023.032

104. Третяк , В., Коломійцев , О., Голубничий , Д., Осієвський , С., Калачова , В. ., & Філіппенков , О. (2023). Варіанти побудови паралельної обчислювальної структури циклічного типу для рішення задач дискретної оптимізації. Grail of Science, (31), 216–227. https://doi.org/10.36074/grail-of-science.15.09.2023.36

105. Listrovoy S.V., Tretjak V.F., Listrovaya A.S. Parallel algorithms of calculation process optimization for the boolean programming problems // Engineering Simulation. – 1999. – vol.16. – P. 569–579.

106. Пономаренко В.С., Голубнчиий Д.Ю., Третяк В.Ф. Цілочисельне програмування в економіці. - Харків: Вид. ХНЕУ, 2005. – 204 с.

107. Третяк В.Ф. Пристрій для рішення задач дискретної оптимізації на графі / В.Ф. Третяк, В.В. Афанасьєв, В.В. Воронін, Б.М. Крук, М.В. Качан, Д.Б. Жуйков, В.О. Прокоф'єв, І.М. Ковальчук, В.В. Василенко, М.Г. Сапога, В.І. Кривчун, В.В. Новіченко, В.С. Шендрик, В.В. Друзь // Modern Directions and Movements in Science: 5 Міжн. наук.-практ. конф., 26-28 січня 2025 р. – Люксембург, Велике Герцогство Люксембург, 2025. – С. 172-180.

108. Патент на корисну модель № 155816, Україна, МПК (2024.01) G06F 15/00 Пристрій пошуку маршруту в графі / Третяк В.Ф., Коломійцев О.В., Васюта К.С., Лук'янчук В.В., Белевщук Я.О., Лапій С.В., Місюк Г.В., Калугін Д.С., Долина М.П., Донцов С.М., Попов В.П., Трофименко Ю.В., Ткачик В.Д., Тах'ян К.А., Нєкрасов С.В. – № u202305634; заяв. 22.11.2023; опубл. 10.04.2024; Бюл. № 15/2024. – 6 с.

109. Патент на корисну модель № 156549, Україна, МПК (2024.01) G06F 15/00 Пристрій для вирішення задач на графі / Третяк В.Ф., Коломійцев О.В.,Васюта К.С.,Голубничий Д.Ю.,Балакірева С.М., Висоцький О.В., Дзюба І.В., Закіров З.З., Запара Д.М., Захарченко В.В., Колмиков М.М., Крамар О.А., Крук Б.М., Маланкевич І.А., Ратич О.Ю., Сєвєрінов О.В., Туленко М.В., – № u202305625; заяв. 18.12.2023; опубл. 11.07.2024; Бюл. № 28/2024. – 6 с.

110. Atzori, L., Iera, A., and Morabito, G. 2017. Understanding the Internet of Things: definition, potentials, and societal role of a fast-evolving paradigm. Ad Hoc Networks 56 (2017), 122–140.

111. Zhang, Zhiming. 2025. Research on key technologies and applications of artificial intelligence in the Internet of Things.

112. IEEE Computer Society. 2020. Artificial Intelligence for Internet of Things: A Survey. IEEE Access 8 (2020), 123456–123489.

113. Gartner Research. 2021. Automation and AI in Software Engineering.

114. IDC. 2020. Data Explosion and Automation Demand in IoT.

115. Sommerville, I. 2016. Software Engineering (10th ed.). Pearson Education.

116. Paris Avgeriou, Christoph Becker, and Uwe Zdun. 2018. Software engineering for Internet of Things systems: A systematic literature review. ACM Computing Surveys 51, 4 (2018), Article 89, 36 pages.

117. Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. 2016. Edge computing: Vision and challenges. IEEE Internet of Things Journal 3, 5 (2016), 637–646.

118. M. Swan. 2015. Blockchain: Blueprint for a New Economy. O'Reilly Media, Sebastopol, CA, USA.

119. Ni, J. 2025. Cross-domain secure sharing algorithm of Internet of Things privacy data based on artificial intelligence. KSII Transactions on Internet & Information Systems.

120. Wang, Fan. 2023. Research on crowd distribution detection and lighting energy-saving control method based on edge computing.

121. Computer Science and Technology. 2023. Research on heterogeneous signcryption scheme with privacy protection.

122. Герасимук, Д., Поляков, А. і Федорченко, В. 2025. Виявлення компромісів між справедливістю, стабільністю і точністю для відповідального вибору моделі машинного навчання. Сучасний стан наукових досліджень та технологій в промисловості. 1(31) (Бер 2025), 5–19.

123. Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, et al. 2019. Software engineering for machine learning: A case study. In Proceedings of the 41st International Conference on Software Engineering (ICSE '19). ACM, Montreal, Canada, 291–300.

124. Menzies, T. and Zimmermann, T. 2017. Software analytics: So what? IEEE Software 34, 5 (2017), 31–37.

125. Zhou, Lanhua, Liu, Xiao, Lei, Lili, and Tan, Xiaolong. 2023. Research on IoT big data mining algorithm based on artificial intelligence technology. Digital Design.

126. Feng, Zihao. 2024. Research on soft sensing method of complex industrial process based on graph neural network.

127. Zhang, Rundi. 2024. Research on deep learning method of fault diagnosis based on generative adversarial network and attention mechanism.

128. Harman, M., Jia, Y., and Zhang, Y. 2015. Achievements, open problems and challenges for search-based software testing. IEEE Software 32, 1 (2015), 67–74.

129. Calinescu, R., Ghezzi, C., Kwiatkowska, M., and Mirandola, R. 2017. Self-adaptive software needs quantitative verification at runtime. Communications of the ACM 55, 9 (2017), 69–77.

130. Li, Yiwei. 2023. Research on bridge health status assessment method based on digital twin.

131. Ni, Jianghao and Jin, Qiyan. 2025. Smart agricultural irrigation system.

132. Cocoatu, M. 2024. Artificial Internet of Things, sensor-based digital twin urban computing vision algorithms, and blockchain cloud networks in sustainable smart city administration. Sustainability.

133. Souri, A., Norouzi, M., and Alsenani, Y. 2024. A new cloud-based cyber-attack detection architecture for hyper-automation process in industrial Internet of Things. Cluster Computing.

134. Li, Shiyu and Liu, Jinchi. 2025. Research and practice on talent training model of Internet of Things engineering based on artificial intelligence. Xinchao Electronics.

135. Wang, Kai, Luan, Xiaoli, and Liu, Chenglin. 2023. Research on the integration of teaching content and innovation of teaching methods for artificial intelligence courses in automation majors. Journal of Wuxi Vocational and Technical College of Commerce.

136. Zhao. 2025. E-commerce driven: the development path of Luodian dragon fruit industry.

137. Li, F., Cheng, B., and Chen, J. 2020. Automated IoT application development based on artificial intelligence techniques. Future Generation Computer Systems 108 (2020), 123–135.

138. ISO/IEC. 2018. ISO/IEC 30141: Internet of Things (IoT) – Reference Architecture. International Organization for Standardization, Geneva, Switzerland.

139. J. Bosch. 2018. From software product lines to software ecosystems. ACM SIGSOFT Software Engineering Notes 43, 1 (2018), 1–3.

140. Ahmed E. Hassan. 2017. Predicting faults using the complexity of code changes. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE ’17). ACM, Urbana-Champaign, USA, 78–88.