

Хмельницький національний університет
Міністерство освіти і науки України

Кваліфікаційна наукова праця
на правах рукопису

КОЗЕЛЬСЬКИЙ ОЛЕКСАНДР ВОЛОДИМИРОВИЧ

УДК 004.03:004.45:004.75

ДИСЕРТАЦІЯ

МЕТОДИ ТА ЗАСОБИ ПЛАНУВАННЯ ЗАДАЧ І ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ
ОПЕРАЦІЙНИХ СИСТЕМ РЕАЛЬНОГО ЧАСУ

123 Комп'ютерна інженерія

12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Подані до захисту наукові положення є власним напрацюванням. Всі використані ідеї, наукові результати, цитати супроводжуються належними посиланнями на їх авторів та джерела опублікування. Всі частини тексту дисертації, під час написання яких використовувались технології штучного інтелекту, перевірені та відредаговані особисто.

О.В. Козельський

(підпис)

Наукові керівники:

Савенко Богдан Олегович, доктор філософії

Коробчинський Максим Володимирович, доктор технічних наук, професор

Хмельницький – 2026

АНОТАЦІЯ

Козельський О. В. Методи та засоби планування задач і підвищення ефективності операційних систем реального часу. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 123 – Комп'ютерна інженерія. – Хмельницький національний університет, Хмельницький, 2026.

У дисертації здійснено аналіз методів планування задач і підвищення ефективності операційних систем реального часу (ОСРЧ). У роботі удосконалено архітектуру планувальника задач шляхом залучення зовнішнього модуля машинного навчання, розроблено метод динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції, метод превентивного відновлення компонентів ОСРЧ, а також метод виявлення фальсифікацій та аномалій у комп'ютерних системах для сигналів кіберфізичних систем (КФС) на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів при збереженні детермінованості виконання задач реального часу.

Об'єктом дослідження є процеси функціонування операційних систем реального часу в кіберфізичних системах в умовах інтенсивних змін станів.

Предметом дослідження є методи та засоби адаптивного планування задач і розподілу ресурсів на основі багатовимірної аналізу навантаження, превентивного відновлення компонентів із прогнозуванням станів, а також виявлення аномалій і фальсифікацій сенсорних даних в ОСРЧ для КФС.

Метою дисертаційного дослідження є підвищення ефективності використання ресурсів, безперервності та надійності роботи ОСРЧ у КФС з інтенсивними змінами станів шляхом розроблення методів адаптивного планування, динамічного розподілу ресурсів, превентивного відновлення, виявлення аномалій та фальсифікацій.

Наукова новизна отриманих результатів полягає в наступному:

1) *розроблено удосконалену архітектуру* планувальника задач із залученням зовнішнього модуля машинного навчання, яка, на відміну від традиційних архітектур ОСРЧ, передбачає винесення аналітичних обчислень за межі мікроконтролера та формування зворотного каналу самоадаптації, що забезпечує збирання багатовимірної телеметрії, її перетворення у тензорні подання та адаптивне оновлення параметрів планувальника без втручання користувача і дає змогу враховувати багатовимірну динаміку навантаження під час керування ресурсами, усуваючи обмеження, зумовлені обчислювальними ресурсами мікроконтролерних платформ, при цьому зберігаючи детермінованість виконання задач у критичному контурі реального часу;

2) *вперше розроблено* метод динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції, який використовує багатовимірні моделі системного навантаження для прогнозування поведінки задач з урахуванням взаємозв'язків між параметрами з наступним формуванням рішення на основі їх комплексного аналізу, що на відміну від існуючих алгоритмів розподілу ресурсів, які базуються переважно на одномірних метриках або локальних характеристиках задач, дає змогу скоротити час виконання, зменшити навантаження на процесор та підвищити ефективність використання ресурсів системи;

3) *вперше розроблено* метод превентивного відновлення компонентів ОСРЧ, який поєднує низькорозмірну марковську модель прогнозування станів із багаторівневим сторожовим контролем, що на відміну від традиційних watchdog-схем, які реагують лише після зупинки або зависання, дозволяє завчасно оцінювати ризик відмови та локально програмно ініціювати апаратний перезапуск з формалізацією індексу стану, перехідних ймовірностей та політик ескалації, зберігаючи при цьому низьку обчислювальну складність, придатну для мікроконтролерів у реальному часі;

4) *набув подальшого розвитку* метод виявлення фальсифікацій та аномалій у комп'ютерних системах для сигналів КФС на основі модифікованого комбінованого

фільтра Калмана з подієвим перемиканням режимів, який, на відміну від поширених багатомодельних або адаптивних фільтрів, не потребує одночасного запуску набору моделей і забезпечує селективне пригнічення підозрілих вимірів на основі аналізу інновацій, що дає змогу відрізняти нормальні режимні зміни від зловмисних відхилень шляхом оперативної зміни режиму оцінювання, зберігати низьку обчислювальну складність, необхідну для мікроконтролерів у системах реального часу, забезпечувати швидке реагування, зменшувати кількість хибних тривог і робить метод придатним до впровадження у ресурсообмежених КФС.

Практичне значення отриманих результатів. За результатами виконаних досліджень здобувачем удосконалено архітектуру планувальника задач шляхом залучення зовнішнього модуля машинного навчання, з адаптивним керуванням розподілу задач і ресурсів без збільшення навантаження на мікроконтролер і без порушення детермінованості виконання задач реального часу. Це дало змогу створювати ОСРЧ з покращеними характеристиками використання обчислювальних ресурсів, що проявляється у зменшенні сумарного часу виконання задач, зниженні навантаження на процесор і скороченні використання оперативної пам'яті під час роботи системи.

Застосування методу динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції забезпечило покращення використання обчислювальних ресурсів за рахунок більш рівномірного розподілу обчислювальних фаз задач у часі. За результатами першої серії експериментів середнє значення завантаженості процесора зменшилося на 3,94 %, середній час виконання задач зменшився на 10,86 %, тоді як використання пам'яті збільшилося на 0,87 %. У другій серії експериментів зниження використання процесора становило 13,08 %, зменшення часу виконання задач становило 3,61 %, при збільшенні використання пам'яті на 0,51 %. Інтегральний показник ефективності, визначений як середнє відносне покращення показників використання процесора, пам'яті та часу виконання задач, за результатами двох серій експериментів становив 5,19 %, що підтверджує доцільність застосування

запропонованого методу для підвищення ефективності функціонування ОСРЧ в умовах обмежених обчислювальних ресурсів.

Застосування методу превентивного відновлення компонентів ОСРЧ на основі низькорозмірної марковської моделі та дворівневого програмно-апаратного сторожового механізму дозволило зменшити частку простою системи з 4,5 % до 1,37 %, що відповідає скороченню простою на 69,63 % та прискоренню відновлення у 3,29 раза порівняно з традиційним апаратним сторожовим таймером. Додаткове навантаження на процесор не перевищувало 1,4 %.

Застосування методу виявлення фальсифікацій та аномалій у сигналах КФС на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів дозволило підвищити ефективність виявлення аномалій та фальсифікацій при умовах ЗПЗ у цифрових сигналах, забезпечивши найвищу інтегральну ефективність ($I = 9,17$) серед порівнюваних методів, перевищивши класичний фільтр Калмана (5,71), адаптивний фільтр Калмана (7,52), H_∞ -фільтр (1,71) та подієво-керований фільтра Калмана (5,18). При цьому відносна ефективність запропонованого методу порівняно з адаптивним фільтром Калмана, який показав найкращу ефективність серед інших зросла приблизно на 21,94%.

Теоретичні та практичні результати дослідження впроваджені у прототипі ОСРЧ на базі STM32F407 з використанням FreeRTOS в ТОВ Ультра ІТ (Акт від 13.11.2025), ТОВ ДЕВІКС ДІДЖИТАЛ (Акт від 18.11.2025), ТОВ Nolt technologies (Акт від 30.12.2025), в освітньому процесі Хмельницького національного університету (Акт від 02.12.2025) на кафедрі комп'ютерної інженерії та інформаційних систем при викладанні дисциплін «Технічна діагностика і надійність комп'ютерних пристроїв», «Безпека та захист комп'ютерних систем», а також, в освітньому процесі у блоці військово-спеціальних дисциплін та використані при удосконаленні навчально-лабораторного комплексу другої кафедри Другого навчально-наукового інституту Військової академії імені Євгена Березняка (Акт від 18.12.2025).

У вступі обґрунтовано актуальність дослідження, пов'язану з потребою підвищення ефективності, надійності та захисту ОСРЧ в КФС з інтенсивними змінами станів. Визначено мету, завдання, об'єкт і предмет дослідження, спрямовані на створення самоадаптивних ОСРЧ, здатних прогнозувати збої, ефективно використовувати ресурси та протидіяти зловмисним впливам. Подано наукову новизну, висвітлено практичне значення отриманих результатів, а також представлено зв'язок тематики дослідження з науковими напрямками, у межах яких працюють відомі дослідники цієї проблеми.

У першому розділі подано аналітичний огляд сучасних ОСРЧ та КФС, розглянуто їх архітектурні особливості, класифікацію, підходи до планування задач і забезпечення відмовостійкості. Проаналізовано існуючі методи моніторингу, самовідновлення та захисту від зловмисних впливів, визначено їх обмеження в умовах інтенсивних змін станів. Узагальнено результати досліджень вітчизняних і зарубіжних учених у сфері підвищення надійності й ефективності ОСРЧ. На основі аналізу сформульовано проблему, яку не розв'язують відомі підходи, та обґрунтовано необхідність розроблення нових методів синтезу ОСРЧ, здатних до самоадаптації, інтелектуальної кластеризації задач і проактивного захисту системи.

У другому розділі викладено концептуальні засади удосконалення архітектури ОСРЧ з планувальником задач шляхом залучення зовнішнього аналітичного модуля, а також визначено вимоги до її ефективності, надійності та захищеності в умовах інтенсивних змін станів. Розроблено структурну модель взаємодії компонентів ОСРЧ, описано принципи інтеграції підсистем моніторингу, планування, самовідновлення та аналізу даних. Детально розглянуто метод динамічного розподілу задач і ресурсів на основі тензорної декомпозиції та архітектуру планувальника із зовнішнім модулем, що приймає дані з ОСРЧ, здійснює їх обробку засобами машинного навчання та передає оновлені параметри для подальшої самоадаптації системи. Представлено можливість як зовнішнього, так і внутрішнього розміщення аналітичного модуля без порушення детермінованості системи. У результаті здійснено розроблення інтеграції

механізмів тензорного аналізу, кластеризації та адаптивного керування у структуру сучасних ОСРЧ.

У третьому розділі подано методи прогнозування збоїв, превентивного перезапуску компонентів і виявлення аномалій та фальсифікацій у цифрових сигналах ОСРЧ. Запропоновано марковську модель прогнозування станів системи, яка дає змогу оцінювати ймовірності переходів між станами системи та реалізовувати локальне відновлення компонентів без повного перезапуску, що забезпечує відмовостійкість системи. Розроблено комбінований фільтр Калмана з подієвим режимом роботи, адаптивним перемиканням моделей і модулем виявлення аномалій. Метод формалізовано як гібридну систему з перемиканнями та подано алгоритм, що виконує прогнозування, аналіз інновацій, перевірку порогів і реакцію системи у разі виявлення аномалій або атак.

У четвертому розділі подано експериментальні дослідження запропонованих методів і рішень у складі операційної системи реального часу для КФС. Описано створення прототипу на базі FreeRTOS із реалізованими механізмами інтелектуальної кластеризації, дворівневої відмовостійкості та комбінованого фільтра Калмана. Наведено результати випробувань роботи мікроконтролера, які підтвердили підвищення ефективності розподілу ресурсів, зменшення часу простою та стійкість системи до збоїв і зловмисних впливів. Узагальнено результати експериментів і зроблено висновок про практичну придатність розроблених методів для застосування у вбудованих і КФС.

У висновку узагальнено результати дослідження, підтверджено досягнення поставленої мети, сформульовано основні наукові та практичні результати.

У додатках наведено перелік наукових праць, в яких представлено наукові результати дослідження, акти впровадження, лістинг програмного забезпечення, таблиці, що містять результати експериментів.

Ключові слова: операційна система реального часу, кіберфізична система, інтелектуальна кластеризація, тензорна декомпозиція, відмовостійкість, сторожовий

таймер, ланцюг Маркова, комбінований фільтр Калмана, самовідновлення, захист від зловмисного впливу, датчики, інтернет речей, моніторинг.

ANNOTATION

Kozelskyi O. V. Methods and Means for Task Scheduling and Efficiency Improvement in Real-Time Operating Systems. – Qualification research paper (manuscript).

Dissertation for obtaining the Doctor of Philosophy degree in specialty 123 – Computer Engineering. – Khmelnytskyi National University, Khmelnytskyi, 2026.

The dissertation analyzes methods of task scheduling and efficiency improvement in real-time operating systems (RTOS). The work improves the architecture of the task scheduler through the integration of an external machine learning module, develops a method for dynamic allocation of tasks and resources in RTOS based on tensor decomposition, a method for preventive recovery of RTOS components, as well as a method for detecting falsifications and anomalies in computer systems for signals of cyber-physical systems (CPS) based on a modified combined Kalman filter with event-driven mode switching while preserving the determinism of real-time task execution.

The object of research is the processes of functioning of real-time operating systems in cyber-physical systems under conditions of intensive state changes.

The subject of research is methods and tools for adaptive task scheduling and resource allocation based on multidimensional workload analysis, preventive recovery of components using state prediction, as well as detection of anomalies and falsification of sensor data in RTOS for CPS.

The purpose of the dissertation research is to improve the efficiency of resource utilization, as well as the continuity and reliability of RTOS operation in cyber-physical systems with intensive state changes by developing methods for adaptive scheduling, dynamic resource allocation, preventive recovery, and detection of anomalies and falsifications.

The scientific novelty of the obtained results is as follows:

1) *an improved architecture* of the task scheduler incorporating an external machine learning module has been developed. Unlike traditional RTOS architectures, it enables analytical computations to be offloaded outside the microcontroller and establishes a feedback channel for self-adaptation. This approach ensures the collection of multidimensional telemetry, its transformation into tensor representations, and the adaptive updating of scheduler parameters without user intervention. As a result, the system can account for multidimensional workload dynamics during resource management, overcoming limitations imposed by the computational resources of microcontroller platforms while preserving deterministic task execution within the critical real-time loop;

2) *for the first time*, a method for dynamic task and resource allocation in RTOS based on tensor decomposition has been developed, which uses multidimensional models of system workload to predict task behavior considering interrelations between parameters and subsequently form decisions based on their comprehensive analysis. Unlike existing resource allocation algorithms that rely mainly on one-dimensional metrics or local task characteristics, the proposed method makes it possible to reduce execution time, decrease processor load, and improve the efficiency of system resource utilization;

3) *for the first time*, a method for preventive recovery of RTOS components has been developed that combines a low-dimensional Markov state-prediction model with multi-level watchdog supervision. Unlike traditional watchdog schemes that react only after a halt or freeze has occurred, the proposed approach enables early assessment of failure risk and local software-initiated hardware reset, with formalization of a state index, transition probabilities, and escalation policies, while maintaining low computational complexity suitable for real-time microcontrollers;

4) the method for detecting falsifications and anomalies in computer systems for CPS signals has been further developed based on a modified combined Kalman filter with event-driven mode switching which, unlike widely used multimodel or adaptive filters, does not require the simultaneous execution of multiple models and provides selective suppression

of suspicious measurements through innovation analysis, enabling the distinction between normal mode changes and malicious deviations via instantaneous switching of the estimation regime while preserving the low computational complexity required for real-time microcontrollers, ensuring rapid response, reducing false alarms, and making the method suitable for deployment in resource-constrained CPS.

Practical significance of the obtained results. As a result of the conducted research, the applicant has improved the architecture of the task scheduler by incorporating an external machine learning module, enabling adaptive management of task and resource allocation without increasing the load on the microcontroller and without violating the determinism of real-time task execution. This made it possible to design real-time operating systems with improved efficiency of computational resource utilization, manifested in a reduction of the total task execution time, decreased processor load, and reduced RAM usage during system operation.

The application of the method for dynamic allocation of tasks and resources in an RTOS based on tensor decomposition ensured improved utilization of computational resources through a more uniform distribution of computational phases of tasks over time. According to the results of the first experimental series, the average CPU utilization decreased by 3.94%, the average task execution time decreased by 10.86%, while memory usage increased by 0.87%. In the second experimental series, CPU utilization decreased by 13.08%, task execution time decreased by 3.61%, while memory usage increased by 0.51%. The integral efficiency indicator, defined as the average relative improvement of CPU utilization, memory usage, and task execution time metrics, amounted to 5.19% based on the results of the two experimental series. This confirms the feasibility of applying the proposed method to improve the efficiency of RTOS operation under limited computational resources.

The application of the method for preventive recovery of RTOS components based on a low-dimensional Markov model and a two-level hardware–software watchdog mechanism reduced system downtime from 4.5% to 1.37%, corresponding to a 69.63%

reduction in downtime and a 3.29-fold decrease in recovery time compared to a conventional hardware watchdog timer. The additional CPU overhead did not exceed 1.4%.

The application of the method for detecting falsification and anomalies in CPS signals based on a modified hybrid Kalman filter with event-driven mode switching made it possible to improve the effectiveness of anomaly and falsification detection under intentional interference conditions in digital signals, achieving the highest integral efficiency ($I = 9.17$) among the compared methods, exceeding the classical Kalman filter (5.71), the adaptive Kalman filter (7.52), the H_∞ filter (1.71), and the event-triggered Kalman filter (5.18). At the same time, the relative efficiency of the proposed method compared to the adaptive Kalman filter, which demonstrated the best performance among the other baseline methods, increased by approximately 21.94%.

The theoretical and practical results of the research were implemented in a RTOS prototype based on the STM32F407 using FreeRTOS in Ultra IT LLC (Act of 13.11.2025), DEVIX DIGITAL LLC (Act of 18.11.2025), Nolt technologies LLC (Act of 30.12.2025), in the educational process of the Khmelnytskyi National University (Act of 02.12.2025) at the Department of Computer Engineering and Information Systems while teaching the disciplines "Technical diagnostics and reliability of computer devices", "Security and protection of computer systems", as well as in the educational process within the block of military-specialized disciplines, and were used to improve the educational and laboratory facilities of the second department of the Second Educational and Research Institute of the Yevhen Bereznyak Military Academy (Act of 18.12.2025).

In the introduction, the relevance of the research is substantiated in connection with the need to improve the efficiency, reliability, and security of RTOSs in CPS with intensive state changes. The goal, objectives, object, and subject of the research are defined, aimed at creating self-adaptive RTOSs capable of predicting failures, efficiently utilizing resources, and counteracting malicious influences. The scientific novelty is presented, the practical significance of the obtained results is highlighted, and the connection of the research topic

with scientific fields in which well-known researchers in this area work worldwide is also described.

The first chapter presents an analytical review of modern RTOS and CPS, considering their architectural features, classification, approaches to task scheduling, and fault-tolerance mechanisms. Existing methods of monitoring, self-recovery, and protection against malicious impacts are analyzed, and their limitations under conditions of intensive state changes are identified. The results of studies by domestic and foreign researchers in the field of improving the reliability and efficiency of RTOS are summarized. Based on this analysis, a problem not solved by existing approaches is formulated, and the necessity of developing new methods for RTOS synthesis capable of self-adaptation, intelligent task clustering, and proactive system protection is substantiated.

The second chapter presents the conceptual foundations for improving the RTOS architecture with a task scheduler through the integration of an external analytical module, and also defines the requirements for its efficiency, reliability, and security under conditions of intensive state changes. A structural model of interaction between RTOS components is developed, and the principles of integrating monitoring, scheduling, self-recovery, and data analysis subsystems are described. A method for dynamic allocation of tasks and resources based on tensor decomposition and the architecture of a scheduler with an external module are considered in detail; this module receives data from the RTOS, processes it using machine learning techniques, and transmits updated parameters for further self-adaptation. The possibility of both external and internal placement of the analytical module without violating system determinism is demonstrated. As a result, mechanisms of tensor analysis, clustering, and adaptive control are integrated into the structure of modern RTOSs.

The third chapter presents methods for failure prediction, preventive component restart, and detection of anomalies and falsifications in digital signals of RTOS. A Markov model for predicting system states is proposed, which makes it possible to estimate the probabilities of transitions between system states and to implement local recovery of components without a full system restart, thereby ensuring system fault tolerance. A

combined Kalman filter with an event-driven operating mode, adaptive model switching, and an anomaly detection module is developed. The method is formalized as a hybrid system with switching, and an algorithm is presented that performs prediction, innovation analysis, threshold checking, and system response in the event of detected anomalies or attacks.

The fourth chapter presents experimental studies of the proposed methods and solutions as part of a real-time operating system for CPS. The creation of a prototype based on FreeRTOS with implemented mechanisms of intelligent clustering, two-level fault tolerance, and a combined Kalman filter is described. The results of microcontroller performance tests are presented, confirming improved efficiency of resource allocation, reduced downtime, and system resilience to failures and malicious impacts. The experimental results are summarized, and a conclusion is drawn regarding the practical applicability of the developed methods for use in embedded systems and CPS.

In the conclusion, the results of the research are summarized, the achievement of the stated objective is confirmed, and the main scientific and practical results are formulated.

The appendices contain a list of scientific publications presenting the research results, acts of implementation, software listings, and tables containing the results of experiments.

Keywords: real-time operating system, cyber-physical system, intelligent clustering, tensor decomposition, fault tolerance, watchdog timer, Markov chain, combined Kalman filter, self-recovery, protection against malicious influence, sensors, IoT, monitoring.

Список публікацій здобувача за темою дисертації

Наукові праці, в яких опубліковані основні наукові результати дисертації

1. Козельський О., Савенко Б. Виявлення фальсифікацій та аномалій у цифрових сигналах від кіберфізичних систем на основі комбінованого фільтра Калмана. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 2025. №2. С. 23–35. DOI: <https://doi.org/10.32782/IT/2025-2-4>

2. Козельський О.В. Метод тензорної декомпозиції для адаптивного розподілу ресурсів у системах реального часу. *Measuring and computing devices in technological*

processes. 2025. №82(2). С. 426-433. DOI: <https://doi.org/10.31891/2219-9365-2025-82-61>

3. Козельський О., Савенко Б., Савенко О. Дворівнева стратегія підвищення відмовостійкості операційних систем реального часу з використанням ймовірнісного аналізу. *Herald of Khmelnytskyi National University. Technical Sciences*. 2025. №353(3.2). С. 438-446. DOI: <https://doi.org/10.31891/2307-5732-2025-353-60>

4. Козельський О., Савенко Б. Зовнішня адаптивна система кластеризації для підвищення гнучкості операційних систем реального часу на основі динамічного планування завдань. *Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки*. Том 5. 2025. Ч. 2. С. 124-132 DOI: <https://doi.org/10.32782/2663-5941/2025.5.2/18>

5. Козельський О., Савенко О. Проактивний механізм інформаційної безпеки в операційних системах реального часу з використанням гібридного сторожового таймера. *Measuring and computing devices in technological processes*. 2025. №83(3). С. 459 – 466. DOI: <https://doi.org/10.31891/2219-9365-2025-83-57>

6. Козельський О., Савенко Б. Виявлення зловмисних атак на сенсори та подробиці телеметрії в кіберфізичних системах на основі модифікованого фільтра Калмана. *Measuring and computing devices in technological processes*. 2025. №84(4). С. 228–235. DOI: <https://doi.org/10.31891/2219-9365-2025-84-24>

Праці, які засвідчують апробацію матеріалів дисертації

7. Kozelskiy O., Kashtalian A., Stetsyuk V., Martiniuk D., Sachenko A. A model of an intelligent clustering system with an external module for the architecture of RTOS with intensive changes of states regarding their flexibility and balancing. *1st International Workshop on Advanced Applied Information Technologies: (AdvAIT-2024)*, Khmelnytskyi, Ukraine and Zilina, Slovakia, December 5, 2024. Vol. 3899. P. 234-243. URL: <https://ceur-ws.org/Vol-3899/paper21.pdf> (Scopus)

8. Kozelskiy O., Drozd A., Savenko B., Gaj P. A model for probabilistic monitoring

and proactive restart of real-time operating systems under intensive state changes in cyber-physical systems. *2nd International Workshop on Intelligent & CyberPhysical Systems: (ICyberPhyS 2025)*, Khmelnytskyi, Ukraine, July 4, 2025. Vol. 4013. P. 198-210. URL: <https://ceur-ws.org/Vol-4013/paper16.pdf> (Scopus)

9. Gaj P., Sochor T., Korobchynskyi M., Savenko B., Kozelskyi O. Hybrid Method for Protecting RTOS from Failures and Cyberattacks Using Compact Markov Models. *The International Workshop on Applied Intelligent Security Systems in Law Enforcement (AISSLE-2025)*, Vinnytsia, Ukraine, October, 30 – November, 02, 2025. Vol. 4126. pp. 361-376. URL: <https://ceur-ws.org/Vol-4126/paper19.pdf>

Публікації, які додатково відображають наукові результати дисертації

10. Свідоцтво про реєстрацію авторського права на твір №141525. Комп'ютерна програма «Програмне забезпечення аналізатора імовірності відмов у реальному часі для операційних систем кіберфізичних систем з інтенсивними змінами станів (АІВРЧ)» / О.В. Козельський, Б.О. Савенко. Дата реєстрації 17.12.2025. URL: <https://sis.nipo.gov.ua/uk/search/detail/1897556>

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ.....	19
ВСТУП.....	20
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ, НАДІЙНОСТІ ТА ЗАХИСТУ ОПЕРАЦІЙНИХ СИСТЕМ РЕАЛЬНОГО ЧАСУ, ПРИЗНАЧЕНИХ ДЛЯ КФС З ІНТЕНСИВНИМИ ЗМІНАМИ СТАНІВ	31
1.1 Особливості функціонування та проблеми забезпечення ефективності ОСРЧ у КФС з інтенсивними змінами станів.....	31
1.2 Аналіз методів забезпечення відмовостійкості та прогнозування збоїв в ОСРЧ.....	43
1.3 Забезпечення виявлення аномалій і фальсифікацій у КФС та існуючі обмеження.....	48
1.4 Постановка задачі дослідження.....	53
1.5 Висновки до першого розділу.....	57
РОЗДІЛ 2 АРХІТЕКТУРА ПЛАНУВАЛЬНИКА ЗАДАЧ ІЗ ЗАЛУЧЕННЯМ ЗОВНІШНЬОГО МОДУЛЯ ТА МЕТОД ДИНАМІЧНОГО РОЗПОДІЛУ ЗАДАЧ І РЕСУРСІВ НА ОСНОВІ ТЕНЗОРНОЇ ДЕКОМПОЗИЦІЇ	58
2.1 Особливості архітектури та принципів функціонування модифікованої ОСРЧ для КФС.....	58
2.2 Архітектура планувальника задач із залученням зовнішнього модуля...	70
2.3 Метод динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції.....	78
2.4 Висновки до другого розділу.....	89
РОЗДІЛ 3 МЕТОД ПРЕВЕНТИВНОГО ВІДНОВЛЕННЯ КОМПОНЕНТІВ ОПЕРАЦІЙНИХ СИСТЕМ РЕАЛЬНОГО ЧАСУ ТА МЕТОД ВИЯВЛЕННЯ ФАЛЬСИФІКАЦІЙ ТА АНОМАЛІЙ У КОМП'ЮТЕРНИХ СИСТЕМАХ ДЛЯ СИГНАЛІВ КІБЕРФІЗИЧНИХ СИСТЕМ	91

3.1	Організація превентивного відновлення та алгоритм роботи задачі прогнозного сторожового контролю в ОСРЧ.....	91
3.2	Метод превентивного відновлення компонентів в ОСРЧ на основі низькорозмірної марковської моделі.....	102
3.3	Метод виявлення фальсифікацій та аномалій у комп'ютерних системах для сигналів КФС на основі модифікованого комбінованого фільтра Калмана.....	115
3.4	Висновки до третього розділу.....	127
РОЗДІЛ 4 ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА АРХІТЕКТУРИ ОПЕРАЦІЙНИХ СИСТЕМ РЕАЛЬНОГО ЧАСУ ТА РОЗРОБЛЕНИХ МЕТОДІВ В УМОВАХ ІНТЕНСИВНИХ ЗМІН СТАНІВ.....		129
4.1	Експериментальне дослідження ефективності динамічного розподілу задач та ресурсів на основі тензорної декомпозиції в ОСРЧ.....	129
4.2	Прототип системи автономного моніторингу та результати експериментальних досліджень.....	148
4.3	Експериментальна порівняльна оцінка ефективності комбінованого фільтра Калмана.....	156
4.4	Висновки до четвертого розділу.....	171
ВИСНОВКИ.....		173
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....		175
ДОДАТОК А Список публікацій здобувача.....		196
ДОДАТОК Б Акти впровадження.....		198
ДОДАТОК В Лістинг псевдокоду та програмного коду менеджера для роботи з бібліотекою TensorFlow.....		207
ДОДАТОК Г Результати збору даних з модифікованої FreeRTOS на основі тензорної декомпозиції.....		212
ДОДАТОК Д Порівняльні таблиці поведінки фільтрів.....		242

ДОДАТОК Е Текст програми аналізатора ймовірності відмов у реальному часі для операційних систем кіберфізичних систем з інтенсивними змінами станів.....	247
---	-----

ПЕРЕЛІК СКОРОЧЕНЬ

АСТ – апаратний сторожовий таймер (Hardware Watchdog Timer, HW WDT)

ГА – генетичні алгоритми

ЗПЗ – зловмисне програмне забезпечення

ККФ – комбінований фільтр Калмана

КФС – кіберфізичні системи

ОСРЧ – операційна система реального часу

ПЗСК – прогнозна задача сторожового контролю (Predictive Watchdog Task, PredictiveWDT)

ПКФК – подієво-керований фільтр Калмана (Event-Triggered Kalman Filter, ETKF)

ФСС – фільтр сильного стеження (Strong Tracking Filter, STF)

EDF – Earliest Deadline First (пріоритетне витісняюче планування)

IDS – Intrusion Detection Systems (система виявлення вторгнень)

IMM – метод взаємодіючих моделей (Interacting Multiple Model)

IoT – Інтернет речей

ML – Machine Learning (методи машинного навчання)

ММАЕ – метод адаптивного оцінювання на основі множини моделей (Multiple Model Adaptive Estimation)

PHM – Prognostics and Health Management (платформа прогнозування і технічного обслуговування за станом)

TDM – Time Division Multiplexing (часове розділення)

ВСТУП

Обґрунтування вибору теми дослідження. Стрімкий розвиток КФС, Інтернету речей та інтелектуальних вбудованих пристроїв супроводжується зростанням складності керуючих алгоритмів, обсягів телеметричних даних і вимог до надійності та безпеки функціонування. У таких умовах операційні системи реального часу повинні забезпечувати не лише детермінованість виконання задач, а й здатність ефективно працювати за наявності інтенсивних змін станів, нестаціонарних навантажень і зовнішніх збурень. Традиційні архітектури ОСРЧ, орієнтовані на використання локальних і переважно статичних критеріїв планування, таких як пріоритет, дедлайн і період виконання, мають обмежену адаптивність і не дозволяють повною мірою враховувати складну багатовимірну динаміку системного навантаження, структурні взаємозв'язки між задачами та довготривалі зміни режимів роботи. У реальних КФС перевантаження і деградація часто виникають не внаслідок перевищення окремого ресурсу, а через несприятливу комбінацію часових, ресурсних і структурних факторів, що ускладнює своєчасне виявлення критичних станів і прийняття коректних керуючих рішень.

Додатковим обмеженням є те, що складні аналітичні підходи, необхідні для глибокого аналізу станів системи, важко реалізувати безпосередньо на мікроконтролерних платформах через жорсткі часові вимоги та обмежені обчислювальні ресурси. Унаслідок цього в багатьох практичних реалізаціях застосовуються спрощені або реактивні механізми планування, моніторингу та відновлення, які реагують на відмови вже після їх виникнення і не забезпечують достатнього рівня прогнозованості та стійкості функціонування. Особливо гостро ці проблеми проявляються в автономних і напівавтономних системах керування, розподілених промислових кіберфізичних комплексах, а також у системах з підвищеними вимогами до безпеки та безперервності роботи, де запізніле виявлення деградацій або неефективне використання ресурсів може призводити до порушення керованості, простоїв чи переходу системи в критичні режими.

Розглянуті обмеження традиційних підходів свідчать про актуальність досліджень, спрямованих на забезпечення своєчасного виявлення деградаційних процесів, коректного прийняття керуючих рішень та раціонального використання обчислювальних ресурсів в операційних системах реального часу при інтенсивних змінах станів.

Значний внесок у розробку моделей, методів та засобів ОСРЧ і КФС здійснено та відповідно сформовано в працях провідних зарубіжних дослідників Д. Гуссенса (J. Goossens) [40], К. Амадери (K. Amadera) [23], Д. Донги (J. Donga) [24], Чжень Суна (Z. Sun) [25], Сяо Гу (X. Gu) [31], С. Парка (S. Park) [45], В. Манжунатха (V. Manjunath) [20], а також українського дослідника О. Летічевського (A. Letichevsky) [150]. У зазначених роботах розглянуто питання детермінованого планування, аналізу часової передбачуваності, гарантування дедлайнів та побудови операційних систем реального часу з прогнозованою поведінкою, що стали основою розвитку сучасних ОСРЧ.

Забезпеченню працездатності ОСРЧ, підвищенню їх надійності, а також безпеки обробки, збору даних присвячені роботи Р. Луни (R. Luna) [49], Ф. Паскуччі (F. Pascucci) [21], Ф. Реггенцані (F. Reghenzani) [57], С. Хама (S. Nahm) [121], а також українського дослідника О. Петришина (O. Petrishin) [149]. У цих працях розглядаються архітектурні та програмно-апаратні механізми підвищення надійності і живучості систем, методи резервування, моніторингу стану та підходи до побудови безпечних і надійних систем реального часу, призначених для збору та первинної обробки даних.

Окремий напрям досліджень присвячено аналізу вразливостей ОСРЧ і КФС, класифікації загроз та сценаріїв атак. Такі питання розглянуто у роботах В. Дуо (W. Duo) [94], А. Маркетто (A. Marchetto) [90], В. Паллеті (V. Palleti) [91], українських дослідників І. Журавської (I. Zhuravska) [109; 141], А. Волокити (A. Volokyta) [148], В. Харченка (V. Kharchenko) [151; 152]. У цих працях досліджено проблеми порушення доступності та цілісності, каскадні ефекти у взаємопов'язаних інфраструктурах, впливи на компоненти керування та мережеву взаємодію, що створює передумови для DDoS/DoS-атак, навмисного внесення відмов і використання

прихованих каналів взаємодії, а також методи виявлення атак і моніторингу стану кіберфізичних систем у реальному часі та їх гарантоздатності.

Незважаючи на значну кількість проведених наукових досліджень у сфері побудови спеціалізованих ОСРЧ для функціонування у КФС, не вирішено проблему забезпечення їх стабільної роботи в умовах інтенсивних змін станів і зовнішніх впливів. Існуючі методи переважно орієнтовані на реактивне усунення збоїв після їх виникнення, а не на їхнє попередження чи передбачення.

Таким чином, виявлено протиріччя між зростаючою потребою у забезпеченні стабільного функціонування КФС в умовах інтенсивних змін станів, нестаціонарних навантажень і можливих спотворень сенсорних даних та обмеженими можливостями традиційних типів архітектури операційних систем реального часу, які орієнтовані переважно на статичні критерії планування задач, реагують на відмови вже після їх виникнення і не забезпечують своєчасного виявлення критичних режимів роботи та коректної обробки вимірювань в умовах обмежених обчислювальних ресурсів.

Це зумовлює потребу у підвищенні ефективності використання обчислювальних ресурсів, своєчасному виявленні відхилень режимів роботи, запобіганні переходу системи в критичні стани, виявленні аномалій і фальсифікацій у сенсорних даних, що є актуальною науково-прикладною задачею.

Одним із шляхів розв'язання цієї задачі є розроблення методів організації планування задач, динамічного розподілу ресурсів, прогнозування станів, превентивного відновлення компонентів, кращого виявлення аномалій і фальсифікацій у сигналах операційних систем реального часу, а також формування відповідної архітектури програмно-апаратних засобів для КФС з інтенсивними змінами станів.

Необхідність усунення виявленого протиріччя обумовлює актуальність роботи і дає змогу сформулювати науково-прикладну задачу, яка полягає в розробленні методів і синтезі ОСРЧ для КФС в умовах інтенсивних змін станів, нестаціонарних навантажень і обмежених обчислювальних ресурсів.

Зазначена науково-прикладна задача відповідає предметній області Стандарту вищої освіти України зі спеціальності 123 – Комп'ютерна інженерія для третього (освітньо-наукового) рівня вищої освіти, зокрема, такому об'єкту вивчення та діяльності, як: кіберфізичні системи, Інтернет речей, системи та засоби оброблення даних і штучного інтелекту, методи та способи подання, отримання, зберігання, передавання та опрацювання інформації, математичні моделі обчислювальних процесів та технології виконання обчислень, архітектура та організація функціонування комп'ютерних систем, інформаційні процеси, технології, методи, способи та інструментальні засоби для дослідження, проектування і експлуатації комп'ютерних та кіберфізичних систем, а також розроблення, верифікації та розгортання програмного забезпечення і забезпечення надійності та безпеки їх функціонування.

Зв'язок роботи з науковими програмами, планами, темами.

Дисертаційне дослідження виконувалось у рамках науково-дослідних тематик Хмельницького національного університету: держбюджетної науково-дослідних теми № 2Б-2024 «Система виявлення ЗПЗ та комп'ютерних атак в корпоративних мережах з використанням хибних об'єктів атак та пасток» (номер державної реєстрації: 0124U000980), № 1Б-2026 «Система забезпечення стійкості до витоку конфіденційної інформації в корпоративних мережах в умовах впливів комп'ютерних атак» (номер державної реєстрації: 0126U002082), в яких автор дисертації був виконавцем.

Мета і завдання дослідження.

Об'єктом дослідження є процеси функціонування операційних систем реального часу в кіберфізичних системах в умовах інтенсивних змін станів.

Предметом дослідження є методи та засоби адаптивного планування задач і розподілу ресурсів на основі багатовимірного аналізу навантаження, превентивного відновлення компонентів із прогнозуванням станів, а також виявлення аномалій і фальсифікацій сенсорних даних в ОСРЧ для КФС.

Метою дисертаційного дослідження є підвищення ефективності використання ресурсів, безперервності та надійності роботи ОСРЧ у КФС з інтенсивними змінами

станів шляхом розроблення методів адаптивного планування, динамічного розподілу ресурсів, превентивного відновлення, виявлення аномалій та фальсифікацій.

Задачі дослідження формуються в роботі наступним чином:

1. Провести аналіз існуючих архітектур ОСРЧ для КФС з інтенсивними змінами станів та визначити їхні обмеження щодо адаптивності, відмовостійкості та інтеграції інтелектуальних методів обробки даних.

2. Удосконалити архітектуру планувальника задач шляхом залучення зовнішнього модуля машинного навчання та зворотного каналу самоадаптації, що забезпечує збирання багатовимірної телеметрії, її тензорне подання та адаптивне оновлення параметрів планування без втручання користувача.

3. Розробити метод динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції, який враховує взаємозв'язки між параметрами задач і ресурсів та забезпечує підвищення ефективності використання обчислювальних ресурсів.

4. Розробити метод превентивного відновлення компонентів ОСРЧ на основі низькорозмірної марковської моделі прогнозування станів у поєднанні з багаторівневим сторожовим контролем, що дозволяє завчасно виявляти деградаційні процеси та ініціювати локальні відновлювальні дії.

5. Розробити метод виявлення фальсифікацій та аномалій у комп'ютерних системах для сигналів КФС на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів і селективним пригніченням підозрілих вимірів, придатний для реалізації в умовах обмежених ресурсів мікроконтролерів.

6. Провести експериментальну перевірку розроблених методів і архітектурних рішень на вбудованій платформі під керуванням ОСРЧ та оцінити їх ефективність за показниками використання процесора й пам'яті, часу виконання задач, тривалості простою системи, кількості хибних спрацювань, швидкодії відновлення та обчислювальної складності.

Методи дослідження. Для розв'язання поставлених задач використовуються тензорний аналіз і лінійна алгебра, марковські процеси та теорія ймовірностей, теорія

оцінювання і фільтрації (фільтр Калмана), методи оптимізації й теорія розкладів, теорія графів, машинне навчання та аналіз часових рядів, теорія комп'ютерних мереж і ОСРЧ, методи захисту інформації та кіберстійкості, а також імітаційне моделювання, прототипування на FreeRTOS і статистична обробка результатів.

Наукова новизна отриманих результатів полягає в наступному:

1) *розроблено удосконалену архітектуру* планувальника задач із залученням зовнішнього модуля машинного навчання, яка, на відміну від традиційних архітектур ОСРЧ, передбачає винесення аналітичних обчислень за межі мікроконтролера та формування зворотного каналу самоадаптації, що забезпечує збирання багатовимірної телеметрії, її перетворення у тензорні подання та адаптивне оновлення параметрів планувальника без втручання користувача і дає змогу враховувати багатовимірну динаміку навантаження під час керування ресурсами, усуваючи обмеження, зумовлені обчислювальними ресурсами мікроконтролерних платформ, при цьому зберігаючи детермінованість виконання задач у критичному контурі реального часу;

2) *вперше розроблено метод динамічного розподілу задач і ресурсів в ОСРЧ* на основі тензорної декомпозиції, який використовує багатовимірні моделі системного навантаження для прогнозування поведінки задач з урахуванням взаємозв'язків між параметрами з наступним формуванням рішення на основі їх комплексного аналізу, що на відміну від існуючих алгоритмів розподілу ресурсів, які базуються переважно на одномірних метриках або локальних характеристиках задач, дає змогу скоротити час виконання, зменшити навантаження на процесор та підвищити ефективність використання ресурсів системи;

3) *вперше розроблено метод превентивного відновлення компонентів ОСРЧ*, який поєднує низькорозмірну марковську модель прогнозування станів із багаторівневим сторожовим контролем, що на відміну від традиційних watchdog-схем, які реагують лише після зупинки або зависання, дозволяє завчасно оцінювати ризик відмови та локально програмно ініціювати апаратний перезапуск з формалізацією індексу стану, перехідних ймовірностей та політик ескалації,

зберігаючи при цьому низьку обчислювальну складність, придатну для мікроконтролерів у реальному часі;

4) набув подальшого розвитку метод виявлення фальсифікацій та аномалій у комп'ютерних системах для сигналів КФС на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів, який, на відміну від поширених багатомодельних або адаптивних фільтрів, не потребує одночасного запуску набору моделей і забезпечує селективне пригнічення підозрілих вимірів на основі аналізу інновацій, що дає змогу відрізняти нормальні режимні зміни від зловмисних відхилень шляхом оперативної зміни режиму оцінювання, зберігати низьку обчислювальну складність, необхідну для мікроконтролерів у системах реального часу, забезпечувати швидке реагування, зменшувати кількість хибних тривог і робить метод придатним до впровадження у ресурсообмежених КФС.

Обґрунтованість і достовірність наукових положень, висновків і рекомендацій. Наукові положення, висновки і рекомендації дисертації обґрунтовані коректним та доцільним використанням математичного апарату, зокрема методів багатовимірної аналізу телеметричних даних, марковських моделей прогнозування станів і методів статистичної обробки вимірювань, застосуванням алгоритмів планування задач, динамічного розподілу ресурсів, превентивного відновлення компонентів і виявлення аномалій та фальсифікацій у сигналах КФС, а також програмною реалізацією розроблених алгоритмів і їх експериментальною перевіркою на вбудованих обчислювальних платформах.

Практичне значення отриманих результатів. За результатами виконаних досліджень здобувачем удосконалено архітектуру планувальника задач шляхом залучення зовнішнього модуля машинного навчання, з адаптивним керуванням розподілу задач і ресурсів без збільшення навантаження на мікроконтролер і без порушення детермінованості виконання задач реального часу. Це дало змогу створювати ОСРЧ з покращеними характеристиками використання обчислювальних ресурсів, що проявляється у зменшенні сумарного часу виконання задач, зниженні

навантаження на процесор і скороченні використання оперативної пам'яті під час роботи системи.

Застосування методу динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції забезпечило покращення використання обчислювальних ресурсів за рахунок більш рівномірного розподілу обчислювальних фаз задач у часі. За результатами першої серії експериментів середнє значення завантаженості процесора зменшилося на 3,94 %, середній час виконання задач зменшився на 10,86 %, тоді як використання пам'яті збільшилося на 0,87 %. У другій серії експериментів зниження використання процесора становило 13,08 %, зменшення часу виконання задач становило 3,61 %, при збільшенні використання пам'яті на 0,51 %. Інтегральний показник ефективності, визначений як середнє відносне покращення показників використання процесора, пам'яті та часу виконання задач, за результатами двох серій експериментів становив 5,19 %, що підтверджує доцільність застосування запропонованого методу для підвищення ефективності функціонування ОСРЧ в умовах обмежених обчислювальних ресурсів.

Застосування методу превентивного відновлення компонентів ОСРЧ на основі низькорозмірної марковської моделі та дворівневого програмно-апаратного сторожового механізму дозволило зменшити частку простою системи з 4,5 % до 1,37 %, що відповідає скороченню простою на 69,63 % та прискоренню відновлення у 3,29 рази порівняно з традиційним апаратним сторожовим таймером. Додаткове навантаження на процесор не перевищувало 1,4 %.

Застосування методу виявлення фальсифікацій та аномалій у сигналах КФС на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів дозволило підвищити ефективність виявлення аномалій та фальсифікацій при умовах ЗПЗ у цифрових сигналах, забезпечивши найвищу інтегральну ефективність ($I = 9,17$) серед порівнюваних методів, перевищивши класичний фільтр Калмана (5,71), адаптивний фільтр Калмана (7,52), H_∞ -фільтр (1,71) та подієво-керований фільтр Калмана (5,18). При цьому відносна ефективність запропонованого

методу порівняно з адаптивним фільтром Калмана, який показав найкращу ефективність серед інших зросла приблизно на 21,94%.

Теоретичні та практичні результати дослідження впроваджені у прототипі ОСРЧ на базі STM32F407 з використанням FreeRTOS в ТОВ Ультра ІТ (Акт від 13.11.2025), ТОВ ДЕВІКС ДІДЖИТАЛ (Акт від 18.11.2025), ТОВ Nolt technologies (Акт від 30.12.2025), в освітньому процесі Хмельницького національного університету (Акт від 02.12.2025) на кафедрі комп'ютерної інженерії та інформаційних систем при викладанні дисциплін «Технічна діагностика і надійність комп'ютерних пристроїв», «Безпека та захист комп'ютерних систем», а також, в освітньому процесі у блоці військово-спеціальних дисциплін та використані при удосконаленні навчально-лабораторного комплексу другої кафедри Другого навчально-наукового інституту Військової академії імені Євгена Березняка (Акт від 18.12.2025).

Особистий внесок здобувача. Всі основні результати дисертаційного дослідження, які представлені до захисту, отримані автором особисто.

У роботах, опублікованих одноосібно автором, отримано наступні результати: [5] – розроблено метод тензорної декомпозиції для адаптивного розподілу ресурсів у системах реального часу.

У роботах, які опубліковані у співавторстві, автору належать основні ідеї, теоретична та практична розробка положень, відображених у характеристиці наукової новизни отриманих результатів, а саме: [138] – розроблено програмний застосунок для аналізатора ймовірності відмов у реальному часі для ОСРЧ для КФС з інтенсивними змінами станів; [136] – розроблено метод виявлення зловмисних атак на сенсори та підробки телеметрії в КФС на основі модифікованого фільтра Калмана; [3] – розроблено зовнішню адаптивну систему кластеризації для підвищення гнучкості ОСРЧ на основі динамічного планування задач; [4] – розроблено дворівневу стратегію підвищення відмовостійкості ОСРЧ з використанням ймовірнісного аналізу; [2] – розроблено метод виявлення аномалій і фальсифікацій в умовах впливу ЗПЗ у цифрових сигналах від КФС на основі комбінованого фільтра Калмана; [6] –

модель інтелектуальної системи кластеризації з зовнішнім модулем для архітектури ОСРЧ з інтенсивними змінами станів з погляду їх гнучкості та балансування; [7] – розроблено модель ймовірнісного моніторингу та превентивного перезапуску ОСРЧ в умовах інтенсивних змін станів у КФС; [9] – розроблено проактивний механізм інформаційної безпеки в ОСРЧ з використанням гібридного сторожового таймера; [137] – розроблено гібридний метод захисту ОСРЧ від відмов і кібератак з використанням компактних марковських моделей.

У роботах, які опубліковані у співавторстві, співавторам належать такі результати: [138] – запропонована концепція організації зовнішнього збору телеметричних даних для програмного забезпечення (Савенко Б.); [136] – запропоновано концепцію модельно-орієнтованого захисту сенсорних даних від зловмисних дій (Савенко Б.); [2] – запропоновано концепцію формування тестових сценаріїв із різкими режимними змінами для експериментальної перевірки комбінованого фільтра Калмана (Савенко Б.); [3] – запропоновано підхід до варіювання параметрів навантаження під час експериментів для аналізу поведінки планувальника ОСРЧ (Савенко Б.); [4] – запропоновано концепцію використання апаратного сторожового таймера як резервного механізму (Савенко О.), запропоновано формалізований підхід до взаємодії компонентів (Савенко Б.); [6] – проведено огляд і порівняльний аналіз існуючих підходів до кластеризації задач в ОСРЧ. (Стецюк В.), розроблено концептуальну модель систем на основі визначальних характеристик, здійснено апробацію моделі та візуалізацію процесу роботи процесів в середовищі пакета прикладних програм DSView (Каштальян А.), розробка різних сценаріїв взяття телеметричних даних (Мартинюк Д.), запропоновано методику перетворення зібраних даних у формат, придатний для використання в TinyML-орієнтованих моделях. (Саченко А.); [7] – визначено показники операційних систем реального часу, які впливають на зміну стану операційного середовища систем (Дрозд А.), запропонована концепція порівняльного аналізу та вибору механізмів відмовостійкості (Савенко Б.), здійснено збір та уніфікація даних в експериментальній системі (Гай П.); [9] – запропонована концепція архітектурного підходу до

використання гібридного сторожового таймера (Савенко О.); [137] – запропоновано концепцію локалізації збоїв замість глобального перезапуску ОСРЧ (Савенко Б.), запропоновано концепцію адаптації методів відмовостійкості до ресурсно-обмежених вбудованих платформ (Коробчинський М.), запропоновано підхід до інтеграції системи виявлення вторгнень у механізм превентивного відновлення ОСРЧ (Сочор Т.), здійснення експериментальної перевірки запропонованого підходу та аналізу отриманих результатів (Гай П.).

Апробація результатів дисертації. Апробацію основних положень, ідей, висновків дисертаційної роботи проведено на науковому семінарі кафедри комп'ютерної інженерії та інформаційних систем у Хмельницькому національному університеті. Наукові результати роботи доповідалися також на: 1st International Workshop on Advanced Applied Information Technologies (AdvAIT-2024) (Khmelnyskyi, Ukraine - Zilina, Slovakia, December 5, 2024); 2nd International Workshop on Intelligent & CyberPhysical Systems: (ICyberPhyS 2025) (Khmelnyskyi, Ukraine, July 4, 2025); International Workshop on Applied Intelligent Security Systems in Law Enforcement (AISSLE-2025) (Vinnytsia, Ukraine, October 30 – November 02, 2025).

Публікації. За результатами проведених досліджень основні наукові результати опубліковано у шести наукових статтях у фахових наукових журналах України [2; 3; 4; 5; 9; 136]. Апробація засвідчена публікаціями трьох праць в матеріалах міжнародних конференцій [6; 7; 137], з яких 3 праці індексовані у наукометричній базі Scopus [6; 7; 137]. Опубліковано одне свідоцтво про реєстрацію авторського права на твір (програму) [138].

Структура та обсяг дисертації. Дисертаційна робота складається з анотації, змісту, переліку умовних скорочень, вступу, чотирьох розділів, висновку, списку використаних джерел та шести додатків. Повний обсяг роботи містить 285 сторінок друкованого тексту, з них анотація – на 15 с., зміст – на 3 с., перелік умовних скорочень – на 1 с., основний текст – на 144 с., список із 153 використаних джерел – на 21 с., додатки – на 89 с. Дисертація містить 28 рисунків та 10 таблиць.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ, НАДІЙНОСТІ ТА ЗАХИСТУ ОПЕРАЦІЙНИХ СИСТЕМ РЕАЛЬНОГО ЧАСУ, ПРИЗНАЧЕНИХ ДЛЯ КФС З ІНТЕНСИВНИМИ ЗМІНАМИ СТАНІВ

1.1 Особливості функціонування та проблеми забезпечення ефективності ОСРЧ у КФС з інтенсивними змінами станів

КФС представляють собою складні інтегровані рішення, що поєднують обчислювальні компоненти з фізичними процесами, забезпечуючи тісну взаємодію між ними в реальному часі. Вони є основою для багатьох сучасних технологій, включаючи автоматизоване керування промисловими процесами, інтелектуальні транспортні системи, медичні пристрої та багато інших галузей, де висока швидкість реагування на зміни в навколишньому середовищі є критично важливою [11; 12; 150], застосування мереж далекого радіозв'язку (LPWAN) для збору та передавання телеметричних даних у розподілених КФС [141]. З огляду на ці особливості, ОСРЧ, що використовуються в КФС, повинні не тільки забезпечувати виконання критичних задач у визначений час, але й мати здатність до адаптації у відповідь на інтенсивні зміни станів [13]. У межах даного дослідження під інтенсивними змінами станів розглядатимемо режим функціонування кіберфізичної системи, за якого значення ключових параметрів (завантаження процесора, час виконання задач, затримки, параметри сигналів) досягають граничних рівнів, при яких порушуються або можуть бути порушені гарантії детермінованого виконання задач у реальному часі. Наприклад, у системах автономного транспорту, де кожен елемент системи може взаємодіяти з іншими елементами та навколишнім середовищем, необхідно забезпечити постійний обмін інформацією та оперативне прийняття рішень, що вимагає високої надійності та гнучкості від ОСРЧ, що є нелегким завданням в умовах інтенсивних змін станів [14]. На сьогоднішній день КФС стикаються з численними викликами, серед яких особливо важливими є забезпечення стабільної роботи при

високих динамічних навантаженнях, захист від збоїв та забезпечення безпеки даних [15; 16].

У межах ОСРЧ ключову роль у забезпеченні часової передбачуваності відіграють алгоритми планування задач. Класичними підходами є планування з фіксованими пріоритетами та алгоритми з динамічними пріоритетами, серед яких найбільш поширеним є планування за найбільш раннім граничним терміном завершення (EDF). Такі алгоритми дозволяють гарантувати дотримання дедлайнів для періодичних і неперіодичних задач за умови стабільних навантажень і коректної моделі виконання. Водночас у КФС з інтенсивними та нерівномірними змінами станів, стохастичними збуреннями і можливими навмисними впливами, класичні алгоритми планування не забезпечують достатньої гнучкості, оскільки не враховують ймовірнісну деградацію станів системи та не мають механізмів прогнозування порушень часових обмежень до моменту фактичного пропуску дедлайну. Крім того, класичні методи керування ресурсами оперують переважно одномірними або слабко пов'язаними метриками, що не дозволяє виявляти приховані закономірності у навантаженні. Перевантаження і деградація в КФС часто виникають не через перевищення окремого ресурсу, а внаслідок несприятливої комбінації часових, ресурсних і структурних характеристик задач. Тензорне подання телеметричних даних дає змогу виконувати комплексний аналіз багатовимірних взаємозв'язків між задачами, ресурсами та часовими параметрами, формуючи рішення на основі структури навантаження, а не лише його величини.

Особливу складність становлять системи, в яких зміни станів відбуваються з високою частотою та інтенсивністю, що створює додаткові вимоги до ефективності управління ресурсами, пам'яттю [17] та реагування на зміни в режимі реального часу. Неабияку важливість має ефективне управління енергоспоживанням, оскільки його неефективність призводить до швидкого виснаження ресурсів живлення та скорочення часу автономної роботи [18].

Операційна система реального часу (ОСРЧ) вважається детермінованою, якщо виконання її операцій відбувається у фіксований момент часу або в межах наперед заданих часових інтервалів. Важливим поняттям при аналізі часових характеристик є резерв часу виконання задач (WCET slack), який визначається як різниця між найгіршим оціненим часом виконання задачі та її фактичним часом роботи або між моментом завершення і дедлайном. Наявність такого резерву дозволяє переносити другорядні обчислення, перевірки цілісності даних або моніторинг стану системи у вільні інтервали без порушення часових обмежень. Це дає змогу ефективніше використовувати ресурси та підвищувати надійність системи в умовах жорстких часових вимог [145]. У вільні інтервали часу, що утворюються за рахунок резерву виконання, доцільно переносити допоміжні процедури контролю коректності, зокрема перевірки цілісності та автентичності [1] потоків даних, цілісності керуючого потоку та цілісності вказівників. Такі механізми базуються на теорії планування задач із фіксованими пріоритетами та динамічним розподілом ресурсів і дозволяють виконувати моніторинг та перевірки без порушення часових обмежень, використовуючи доступний резерв процесорного часу. При конкуренції процесів за володіння обчислювальними ресурсами та процесорним часом система не є повністю детермінованою, а ступінь детермінізму визначається максимальною затримкою реакції на зовнішні та внутрішні події. У табл. 1.1 наведено порівняння основних алгоритмів планування задач в ОСРЧ.

Таблиця 1.1 – Переваги та недоліки основних алгоритмів планування задач в ОСРЧ

Алгоритм	Принцип роботи	Переваги	Недоліки
Rate Monotonic (RM)	Пріоритет визначається періодом задачі	Простота; передбачуваність	Негнучкість; обмежена адаптивність
Deadline Monotonic (DM)	Пріоритет визначається відносним дедлайном	Підтримка різних дедлайнів; передбачуваність	Статичність; залежність від параметрів

Кінець таблиці 1.1

Earliest Deadline First (EDF)	Пріоритет за найближчим дедлайном	Висока ефективність; адаптивність	Нестабільність при перевантаженні
Least Slack Time (LST)	Пріоритет за найменшим запасом часу	Висока точність; адаптивність	Часті перемикання контексту; складність
First Come First Served (FCFS)	Виконання в порядку надходження	Простота; низькі накладні витрати	Відсутність гарантій реального часу
Cyclic Executive	Фіксований розклад	Детермінованість; простота	Негнучкість; відсутність адаптивності

Незважаючи на різноманітність підходів, класичні алгоритми планування орієнтовані на роботу в умовах відносної стабільності параметрів задач. У разі інтенсивних змін станів системи їх ефективність знижується, що обмежує можливість забезпечення гарантованого виконання задач у реальному часі.

В умовах інтенсивних змін станів традиційні операційні системи реального часу не завжди здатні забезпечити необхідний рівень гнучкості та адаптивності [19]. Основні проблеми, з якими стикаються такі системи, пов'язані з недостатньою швидкістю реакції на зміни станів [20]. ОСРЧ повинна миттєво реагувати на зміни в навколишньому середовищі або у внутрішніх процесах [21]. Проте багато існуючих систем мають обмежену швидкість адаптації [22], що може призводити до затримок у виконанні критичних задач або навіть до збоїв у роботі всієї системи [23], наприклад через раптове зростання навантаження [24; 25]. Задача з низьким пріоритетом може раптово стати критично важливим, якщо виникає загроза, наприклад, в безпеці [26].

Наведена ілюстрація на рис. 1.1 демонструє, що навіть класичні алгоритми планування, зокрема EDF та планування з фіксованими пріоритетами, забезпечують дотримання дедлайнів лише за умов стабільного навантаження та коректної моделі

виконання задач. У КФС з інтенсивними змінами станів та стохастичними збуреннями порушення часових обмежень можуть виникати до моменту реакції планувальника, що обмежує застосовність виключно реактивних підходів [134; 135].

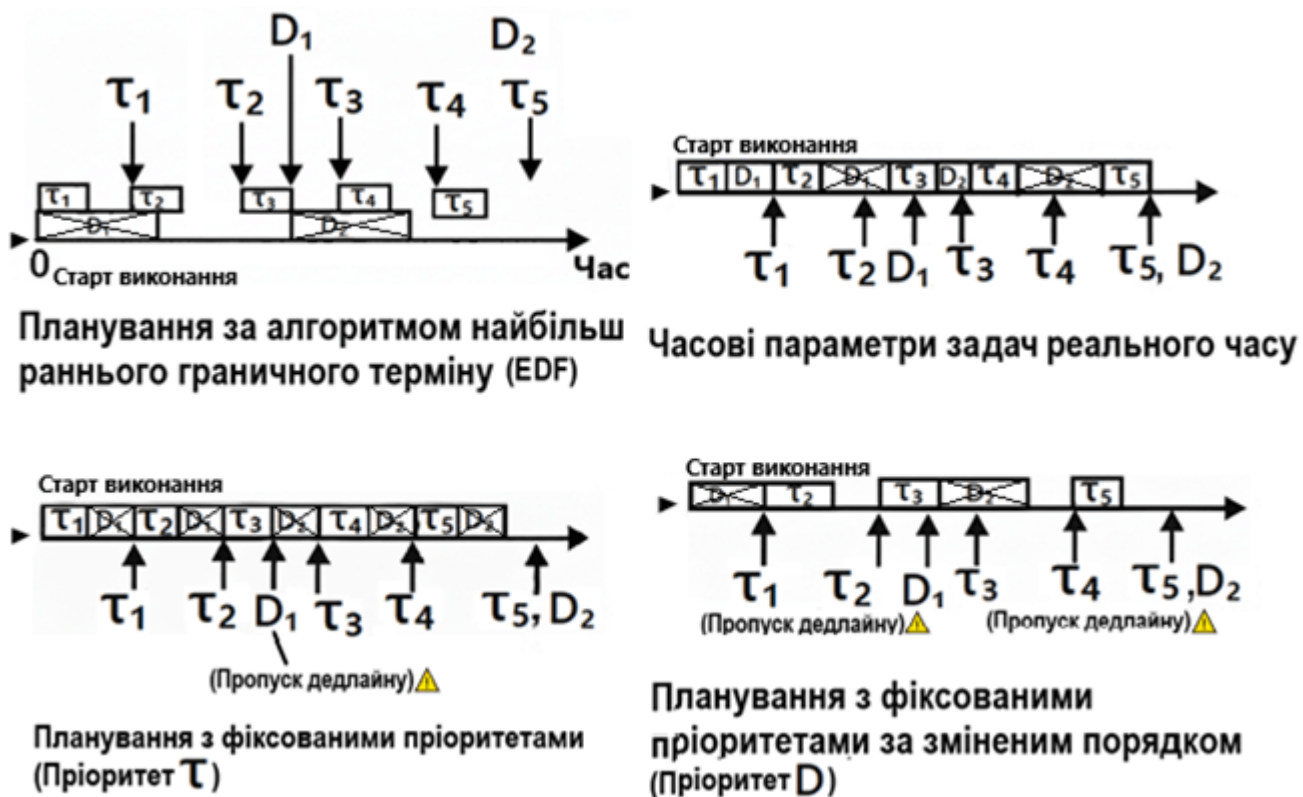


Рисунок 1.1 - Порівняльна ілюстрація виконання періодичних задач реального часу за алгоритмами планування з фіксованими та динамічними пріоритетами з урахуванням граничних термінів завершення [135]

Одним із практичних рішень у таких умовах є застосування системи ROS 2 разом із спеціальним ядром Linux, що підтримує повне переривання задач. Правильне налаштування потоків, пріоритетів і мережевої взаємодії дозволяє суттєво зменшити затримки та підвищити передбачуваність реакції, але ця система не є строго детермінованою. Її реакції наближаються до строгого детермінування, але навіть за умов повного примусового переривання задач і належного налаштування режиму реального часу детермінованість не стає абсолютною [144].

Швидкість реакції ОСРЧ на зміни станів визначається не лише алгоритмом планування, а й здатністю системи коректно перерозподіляти ресурси в умовах змінної критичності задач. У практичних сценаріях задача з формально низьким пріоритетом або пізнішим дедлайном може раптово перейти в критичний стан, наприклад у разі загрози безпеці чи відмови суміжного модуля. У таких умовах навіть оптимальні з точки зору класичної теорії планування алгоритми, наприклад EDF, потребують доповнення механізмами адаптивного управління та моніторингу стану системи.

Кластеризація в ОСРЧ є методом групування даних або об'єктів на основі схожих характеристик чи властивостей. У цій сфері використовується багато різноманітних методів, кожен із яких має свої переваги та недоліки.

До основних методів кластеризації, які можуть бути застосовані, належать:

- часове розділення (TDM);
- методи машинного навчання (ML);
- генетичні алгоритми (ГА);
- методи на основі графів.

TDM широко використовується для керування ресурсами в ОСРЧ [27]. Цей метод полягає у виділенні конкретних часових інтервалів різним задачам або групам задач [27; 28]. Завдяки цьому забезпечується передбачуваний і регулярний доступ до ресурсів, що є надзвичайно важливим для підтримки характеристик систем реального часу [29]. У рамках TDM кожна задача отримує ексклюзивний доступ до процесорних ресурсів на заздалегідь визначений часовий інтервал у циклічному порядку. Такий підхід сприяє високій продуктивності та надійності системи, гарантуючи послідовне виконання задач. Однією з головних переваг цього методу є те, що задача виконується у точно визначені моменти часу, що забезпечує регулярність та передбачуваність роботи системи. Наприклад, у промислових процесах TDM може використовуватися для виділення фіксованих часових інтервалів різним сенсорам і механізмам, запобігаючи можливим конфліктам та забезпечуючи безперебійну

роботу обладнання [28]. Простота реалізації TDM робить його привабливим для вбудованих систем, де важлива легкість впровадження та налаштування [30]. Крім того, контрольоване розподілення часових інтервалів сприяє ефективному використанню ресурсів, забезпечуючи рівномірніше завантаження та зменшення додаткових витрат. Проте існують і недоліки, які необхідно враховувати. Якщо задача не використовує повністю виділений йому часовий інтервал, невикористані ресурси просто втрачаються, що призводить до неефективності системи. У середовищах зі змінним навантаженням фіксовані часові інтервали можуть бути неефективними, оскільки не адаптуються до поточних потреб [30]. Масштабованість стає серйозною проблемою, і виникає потреба в динамічних методах перерозподілу ресурсів між задачами. Врахування пріоритетів задач є також важливим аспектом. Для забезпечення своєчасного виконання критичних задач, особливо в системах, де затримки можуть вплинути на безпеку або ефективність, необхідно застосовувати механізми пріоритетів. Наприклад, у медичних пристроях або системах керування транспортом затримка у виконанні важливого завдання може мати серйозні наслідки. Динамічне планування задач та алгоритми зворотного зв'язку дозволяють системі адаптуватися до змінних умов у режимі реального часу, підвищуючи ефективність використання ресурсів [31]. Проте такі підходи можуть ускладнювати систему та вимагати додаткових обчислювальних ресурсів. Сучасні ОСРЧ часто застосовують гібридні методи, що поєднують переваги TDM з іншими стратегіями, такими як пріоритетне планування або управління на основі подій. Це забезпечує більш гнучке управління ресурсами та покращує загальну продуктивність системи. З розвитком багатоядерних процесорів та паралельних обчислень виникають нові можливості для підвищення ефективності розподілу задач: їх розміщення між ядрами може зменшити залежність від жорстко визначених часових інтервалів та забезпечити ефективніше використання апаратних ресурсів. Незважаючи на певні недоліки, TDM залишається важливим інструментом в управлінні ресурсами в системах реального часу, особливо коли потрібна висока передбачуваність та регулярність виконання задач [29]. Проте

сучасні виклики, такі як динамічні навантаження та потреба в масштабованості, вимагають розробки нових методів та підходів, які можуть доповнити або замінити традиційні методи часового мультиплексування [32]. У підсумку, ефективне управління ресурсами в ОСРЧ є складним завданням, що потребує балансу між передбачуваністю, ефективністю та гнучкістю. Глибоке розуміння переваг і недоліків таких методів, як TDM, дозволяє розробникам обирати найбільш відповідні підходи для конкретних застосувань. Це забезпечує оптимальну роботу системи та задовольняє вимоги користувачів, сприяючи розвитку технологій і підвищенню якості сервісів у різних галузях. Додатково, інтеграція методів штучного інтелекту та машинного навчання може відкрити нові напрями в управлінні ресурсами ОСРЧ [33]. Використання прогностичних моделей для передбачення навантаження та адаптивного планування задач може значно підвищити ефективність і надійність системи [34]. Такі інновації допоможуть подолати обмеження традиційних методів і підготувати ОСРЧ до майбутніх викликів у сфері технологій.

ML стають дедалі більш популярними та широко застосовуються для задач кластеризації та балансування навантаження в ОСРЧ [35]. Це зумовлено їхньою здатністю автоматично визначати оптимальні рішення та гнучко адаптуватися до постійно змінних умов роботи системи [36]. Серед таких методів використовуються різноманітні алгоритми кластеризації, навчання з підкріпленням та нейронні мережі, які разом забезпечують ефективний розподіл ресурсів і управління задачами в режимі реального часу [37; 38].

Алгоритми машинного навчання мають можливість автоматично виявляти оптимальні шаблони та рішення, аналізуючи великі обсяги даних. Це сприяє більш ефективному розподілу навантаження між різними компонентами системи, що, в свою чергу, підвищує загальну продуктивність та стійкість системи. Завдяки здатності ML-алгоритмів динамічно змінювати стратегії розподілу ресурсів у відповідь на змінні вимоги та умови, вони можуть забезпечувати високий рівень продуктивності навіть у непередбачуваних або мінливих ситуаціях. Крім того, методи машинного навчання

сприяють ефективнішому використанню ресурсів шляхом прогнозування майбутніх навантажень на систему та динамічного коригування розподілу ресурсів. Наприклад, аналізуючи історичні дані, алгоритми можуть передбачити періоди пікового навантаження і завчасно переналаштувати систему для їх обробки [39]. Це дозволяє не лише запобігти можливим перевантаженням, але й підвищити загальну ефективність роботи системи і навіть підвищити загальну безпеку системи.

Проте, попри всі переваги, методи ML мають і певні недоліки.

По-перше, для їх ефективного навчання необхідні великі обсяги високоякісних даних. Збирання та опрацювання таких даних може бути складним завданням у багатьох ОСРЧ, особливо якщо вони розподілені географічно або мають обмежені канали зв'язку. Відсутність достатньої кількості даних може призвести до неточних прогнозів і неефективних рішень.

По-друге, обчислювальна складність алгоритмів ML, особливо тих, що базуються на глибоких нейронних мережах, потребує значних обчислювальних ресурсів. Це може бути проблемою для систем з обмеженими можливостями або для тих, де енергоефективність є критично важливою [40]. Використання таких алгоритмів може вимагати додаткового апаратного забезпечення або спеціалізованих процесорів, що підвищує загальну вартість системи.

По-третє, впровадження та налаштування методів машинного навчання може бути складним процесом, який вимагає високого рівня експертизи та спеціальних знань. Це стосується як розробки самих алгоритмів, так і їх інтеграції в існуючі системи. Помилки на етапі налаштування можуть призвести до неефективної роботи або навіть до збоїв у системі.

Таким чином, хоча методи машинного навчання пропонують потужні інструменти для покращення балансування навантаження та кластеризації в ОСРЧ, їх впровадження супроводжується низкою викликів. Необхідність у великій кількості якісних даних, високі обчислювальні вимоги та складність налаштування можуть обмежувати їх використання в певних контекстах. Важливо ретельно оцінювати всі

аспекти перед впровадженням ML-методів, щоб забезпечити їх ефективність та відповідність потребам конкретної системи.

ГА, які ґрунтуються на принципах еволюційної біології та природного відбору, набувають все більшої популярності для вирішення складних задач керування ресурсами, зокрема кластеризація та балансування навантаження в ОСРЧ [41]. Ці алгоритми імітують процеси природного відбору, включаючи схрещування (кросинговер) та мутацію, щоб ефективно шукати оптимальні або близькі до оптимальних рішення в просторі можливих варіантів. Завдяки цим механізмам, ГА здатні ефективно вирішувати різні типи задач та враховувати обмеження ресурсів, що дозволяє їм знаходити глобальні оптимуми та адаптуватися до різноманітних проблем без необхідності суттєвих змін у самому алгоритмі [42]. Однією з ключових переваг генетичних алгоритмів є їхня здатність стійко функціонувати в умовах змін та невизначеності в системі. Однак, незважаючи на численні переваги, генетичні алгоритми мають ряд суттєвих недоліків. Перш за все, вони характеризуються високою обчислювальною складністю, що вимагає значних обчислювальних ресурсів, особливо при роботі з великими популяціями та складними проблемами. Крім того, для досягнення оптимальних результатів необхідне ретельне налаштування параметрів алгоритму, таких як розмір популяції, ймовірності кросинговеру та мутації. Ще однією проблемою є повільна збіжність генетичних алгоритмів, що може стати критичною у системах реального часу, де важливо отримувати рішення у визначені строки. Масштабованість також є серйозним викликом для генетичних алгоритмів. Зі збільшенням розміру системи та складності задачі, зростають і обчислювальні вимоги. Визначення ефективних критеріїв завершення роботи алгоритму та адаптація до динамічних змін у системі залишаються складними завданнями, що потребують подальших досліджень та вдосконалення.

Методи, що базуються на теорії графів, стають все більш популярними для моделювання взаємодій між задачами та ресурсами в ОСРЧ [43]. Застосування графових методів дає можливість моделювати складні взаємодії та залежності між

задачами і ресурсами, що сприяє більш точній кластеризації та балансуванню навантаження [44]. Ці методи відзначаються високою масштабованістю і здатністю ефективно працювати з великими системами. Однак варто враховувати, що методи на основі графів характеризуються високою обчислювальною складністю. Побудова та обробка великих графів вимагає значних обчислювальних ресурсів, що може становити серйозну проблему для ОСРЧ з обмеженими ресурсами. Крім того, обробка динамічних змін у системі також залишається викликом, оскільки ці методи часто припускають статичну структуру графа, що може бути непридатним у умовах постійних змін та оновлень. Масштабованість також залишається проблемою, оскільки зі збільшенням розміру та складності системи зростають вимоги до обчислювальних ресурсів. Для подолання цих проблем розробка більш ефективних алгоритмів та впровадження методів паралельної обробки може значно покращити продуктивність і зменшити витрати на обробку великих графів [45].

Крім того, складність обробки динамічних змін у системі потребує розробки адаптивних графових методів, які здатні швидко реагувати на зміни в структурі графа без необхідності повної перебудови моделі. Це може включати використання динамічних алгоритмів кластеризації та балансування навантаження, які можуть оперативно адаптуватися до нових умов та вимог системи [46].

Необхідність точної інформації про залежності між задачами та ресурсами також може ускладнювати застосування графових методів у складних системах, де такі залежності не завжди чітко визначені або можуть змінюватися з часом. У таких випадках може знадобитися додаткове моделювання або використання методів машинного навчання для визначення та оновлення цих залежностей у реальному часі. Впровадження більш ефективних алгоритмів та використання паралельних обчислювальних технік можуть значно покращити продуктивність графових методів і зробити їх більш придатними для використання у великих та складних розподілених системах [47].

З проаналізованих методів для їх застосування в ОСРЧ наявні недоліки, які подано в табл. 1.2.

Таблиця 1.2 – Недоліки існуючих методів кластеризації для їх застосування в ОСРЧ

Метод	Опис	Недоліки
TDM	Виділяє конкретні часові інтервали для задач, забезпечуючи передбачуваність і регулярність роботи системи.	Неефективне використання ресурсів, якщо задача не використовує повністю виділений час. Фіксовані інтервали не адаптуються до змінного навантаження. Проблеми масштабованості при збільшенні кількості задач.
ML	Використовуються для задач кластеризації та балансування навантаження з можливістю адаптації до змінних умов.	Необхідність великої кількості високоякісних даних для навчання. Висока обчислювальна складність, потреба в значних ресурсах. Складність налаштування.
ГА	Еволюційний підхід для пошуку оптимальних рішень у задачах кластеризації та балансування.	Висока обчислювальна складність, потреба в великих ресурсах для великих популяцій. Повільна збіжність, що може бути проблемою в реальному часі. Складність масштабування та адаптації до динамічних змін.
Методи на основі графів	Використовують графову модель для моделювання взаємодій між задачами та ресурсами.	Висока обчислювальна складність при побудові та обробці великих графів. Проблеми з адаптацією до динамічних змін у структурі графа. Необхідність точної інформації про залежності, що може ускладнювати застосування в складних системах.

Таким чином, навіть за наявності ефективних алгоритмів планування та кластеризації, у системах з інтенсивними змінами станів неможливо гарантувати

безперервне дотримання часових обмежень винятково за рахунок реактивних механізмів. Пропуск дедлайнів, зациклення задач або некоректна взаємодія між компонентами можуть виникати ще до того, як планувальник виявить критичну ситуацію. Це визначає потребу в доповненні класичних підходів програмними механізмами відмовостійкості та прогнозування збоїв. Аналіз існуючих методів кластеризації в ОСРЧ показує, що кожен із них має свої переваги та недоліки. Вибір оптимального методу залежить від конкретних вимог системи, її розміру, складності та ресурсних обмежень. Тому, необхідно подолання існуючих недоліків та розроблення більш ефективних, адаптивних і масштабованих методів кластеризації для ОСРЧ [139]. Це може включати впровадження гібридних підходів [48; 49], що поєднують переваги різних методів [50], та розробку нових алгоритмів, здатних ефективно працювати в динамічних та складних середовищах ОСРЧ [51], а також моніторингу безпеки проти зловмисного програмного забезпечення, наприклад з BOTNET [52; 53; 54].

1.2 Аналіз методів забезпечення відмовостійкості та прогнозування збоїв в ОСРЧ

У сучасних ОСРЧ і також у вбудованих системах зростає кількість задач та рівень інтеграції, що ускладнює забезпечення безперервної роботи і дотримання жорстких часових обмежень [149]. Особливо це актуально для КФС, де помилка або запізнення у керуванні фізичними процесами може призвести до критичних наслідків, яка може бути пов'язана з виходом з ладу обладнання, аварійними ситуаціями чи недопустимими затримками у виконанні функцій безпеки [55; 56]. А у випадку цілеспрямованих кібератак – створює додаткове вікно вразливості, яке може бути використане зловмисниками для дестабілізації чи блокування роботи системи. Сучасні огляди підкреслюють, що зі зростанням складності середовищ реального часу зростає потреба в інтегрованих програмних механізмах відмовостійкості з урахуванням взаємодії зі планувальниками та змішаною критичністю [57].

У сфері забезпечення відмовостійкості вбудованих систем реального часу існує кілька підходів, спрямованих на виявлення та усунення збоїв. Традиційно розрізняють апаратні та програмні методи, а також їхні комбіновані рішення:

До основних методів відмовостійкості належать:

- апаратний сторожовий таймер;
- резервування;
- формальні методи та верифікація;
- методи на основі машинного навчання.

АСТ зазвичай реалізується як вбудований блок у мікроконтролері або як окремий зовнішній чип з автономним годинником. Його головна функція полягає в контролі програми щодо передачі керуючого сигналу апаратному таймеру у встановлені часові інтервали. Якщо програма зависає або виконується некоректно й не виконує запланованого оновлення таймера, то після завершення встановленого часового терміну сторожовий таймер ініціює повний перезапуск системи [58]. Основна перевага цього механізму полягає в тому, що він функціонує незалежно від основного ядра й не залежить від його коректності. Навіть у разі повного зависання процесорної логіки або зациклення виконання коду АСТ продовжує відлік часу і неминуче виконує перезапуск. Проте недолік полягає у повній реактивності цього методу. Щоб АСТ спрацював, система вже має бути у стані фактичного зависання або серйозного збою, коли не відбувається виклик оновлення таймера. Кожне повне перевантаження означає втрату поточного контексту, перезапуск усієї системи й повторну ініціалізацію апаратних блоків, що може займати значний час, особливо в складніших системах. Наприклад, в автоматизованій виробничій лінії з десятками взаємопов'язаних контролерів локальна деградація окремого вузла або збільшення затримок виконання задач за нерівномірного навантаження може призвести до зупинки всієї лінії у разі глобального перезапуску системи.

Поширеною стратегією підвищення відмовостійкості є резервування, при якому критичні компоненти системи дублюються N-кратно. У разі відмови головного

модуля системи резерв може миттєво або за короткий час перебрати на себе його функції [59; 60]. Такий метод дозволяє уникнути навіть короткочасних зупинок у критичних додатках, де основним критерієм є безперервність функціонування та мінімізація часу відновлення, наприклад, авіаційні системи [61], високошвидкісні промислові конвеєри.

Однак резервування вимагає суттєвих додаткових ресурсів. Це не лише дублювання апаратури, а й складні механізми підтримання узгодженого стану між копіями [62]. Це досить масштабно і вартісно, а також супроводжується зростанням енергоспоживання, що є критичним насамперед для автономних і ресурсно обмежених систем [63; 64]. У простих вбудованих системах з малим бюджетом чи обмеженими фізичними габаритами резервування часто є надмірним [65]. Тому резервування залишається актуальним у висококритичних сферах (авіоніка, медичні системи), але там, де ресурси вкрай обмежені, шукають більш легкі й менш дорогі методи покращення відмовостійкості [56].

У середовищах, де безпека критично важлива (авіоніка – DO-178C, автомобільна галузь – ISO 26262), широко застосовують формальні підходи, як-от часові автомати, мережі Петрі чи моделі на базі перевірки. Ідея полягає в тому, щоб ще на етапі проектування створити математичну модель системи та довести, що певні види збоїв, наприклад, дедлоки, пропуск жорсткого дедлайну, неможливі [66]. Водночас у задачах онлайн-виявлення та локалізації збоїв під час виконання формальні методи часто є надто складними або ресурсомісткими для практичного застосування в ОСРЧ з обмеженими апаратними можливостями.

Ще одне обмеження – комбінаторний вибух, коли система складається з великої кількості задач, подій та проміжних станів, формальні методи можуть вимагати непомірної обчислювальної потужності [67]. Для великої частини реальних вбудованих застосунків (роботизовані платформи, розумні сенсори, IoT) це виявляється непрактичним [68; 69]. Тому формальна верифікація зазвичай

залишається корисною на етапі офлайн-аналізу, а не як механізм онлайн-контролю збою [70].

У області хмарних обчислень та великих кластерів (HPC) алгоритми ML активно застосовують для передбачення навантаження, виявлення аномалій та превентивного перенесення задач, завдяки чому знижують кількість зупинок чи збоїв [71; 72]. У розподілених системах ML-підходи можуть аналізувати телеметрію з багатьох вузлів, виявляючи тонкі закономірності. Проте у вузькоспеціалізованих вбудованих контролерах, наприклад типу STM32, AVR, ESP32, з декількома сотнями кілобайт ОЗП і частотою в десятки-сотні мегагерц, глибокі чи навіть середні нейронні мережі реалізувати проблематично, взяти наприклад проблему візантійської відмовостійкості [73]. Навіть прості ML-моделі можуть забирати відчутну частку ресурсів, а отже зменшувати можливість ОСРЧ виконувати основні функції вчасно, або використовувати додаткові зовнішні ресурси, наприклад хмару [74].

До того ж ML-підходи складно формально сертифікувати, оскільки поведінка моделі статистично нестабільна при рідкісних або неочікуваних збоях. У багатьох сферах, де потрібні жорсткі гарантії (авіація, медтехніка), до ML ставляться з обережністю. Тому машинне навчання лишається привабливим для великих систем, але не завжди підходить для вбудованих рішень з дефіцитом пам'яті [75].

Особливо актуальним є прогнозування відмов у контексті багатозадачних ОСРЧ, де збої можуть проявлятися у вигляді зациклення окремих задач, взаємоблокувань або пропуску критичних дедлайнів. Дослідження показують, що найбільш поширені причини зависань вбудованих систем включають пошкодження пам'яті і застрягання в нескінченному циклі [76], зависання зовнішніх апаратних компонентів, взаємне блокування задач через неправильне використання м'ютексів [77], або ситуації, коли задача високого пріоритету монополізує CPU і не дає іншим задачам виконатися. На мережному рівні раннє виявлення ботнет-активності досягається аналізом DNS-трафіку [78]. Для запобігання обходу механізмів виявлення застосовують спеціалізовані методи

протидії ухиленню на рівні DNS [79]. У корпоративних мережах самоадаптивні механізми підвищують резилентність під час ботнет-атак через локалізацію та ізоляцію вузлів [142]. Дослідження показують, що формування контрольованих зон у локальних бездротових комп'ютерних мережах (наприклад, Wi-Fi) із використанням WIDS, WPA3 та джаммерів значно підвищує безпеку мережі, перешкоджаючи несанкціонованому доступу та атакам [80]. Огляди ботнет-DDoS в IoT-мережах узагальнюють наявні набори даних і методи, демонструючи еволюцію векторів атак [81].

Тому винятково АСТ, попри простоту і надійність, не забезпечує превентивної (проактивної) реакції, а виконує загальну перезавантажувальну функцію. Програмний сторожовий таймер пропонує більшу гнучкість, але теж має обмеження, коли зависання охоплює все ядро. Резервування підходить має високу ціну й вимагає додаткові ресурси, а ML або складні формальні методи часто виявляються надмірними з точки зору обчислень для мікроконтролерів [82].

Найперспективнішою проміжною стратегією видається подвійний підхід. Суть його в тому, щоб залишити АСТ як останній спосіб захисту від фатальних зависань, а програмний шар навчити проактивно виявляти збої ще до їх фактичного настання і виконувати локальний перезапуск лише проблемних модулів. Для реалізації цього програмного шару пропонують різні механізми оцінки стану системи, зокрема використання емпіричних порогів та простих аналітичних моделей. З огляду на обмеження ресурсів та потребу в швидкому обчисленні й відсутності складних алгоритмів, спрощені марковські ланцюги (низькорозмірні) виглядають оптимальним компромісом, порівняно зі звичайними [83; 84; 85], які використовуються доволі активно в інших системах, наприклад в автоматизованих керованих транспортних засобах [86], чи ремонту заміни машинного обладнання [87] та інших [88]. Вони дають змогу з урахуванням минулих переходів [89] у робочих станах системи обчислювати ймовірність наближення до критичної ситуації. Завдяки цьому вдається уникнути повного скидання від апаратного сторожового таймера, зекономити час і забезпечити

безперервне функціонування. З проаналізованих методів для їх застосування в ОСРЧ наявні недоліки, які подано в табл. 1.3.

Таблиця 1.3 – Недоліки існуючих методів забезпечення відмовостійкості для їх застосування в ОСРЧ

Метод	Опис	Недоліки
АСТ	Мінімальне навантаження на ЦП. Незалежність від ядра системи. Простота реалізації	Реагує вже після зависання. Повний reset означає втрату контексту. Не запобігає збоям, а лише перезапускає.
Резервування	Висока відмовостійкість за рахунок дублювання. Швидкий перехід на резервний канал (гарячий або холодний резерв)	Потреба у додаткових ресурсах (пам'ять, апаратні модулі). Складна синхронізація станів. Збільшена вартість та енергоспоживання
Формальні методи	Формальні, часто суворі гарантії коректності. Виявлення логічних помилок на етапі проєктування	Складність і трудомісткість моделювання. Комбінаторний вибух для великих систем. Застосовуються переважно офлайн, а не для онлайн-контролю
ML-підходи	Можливість виявлення складних аномалій та патернів. Прогнозування відмов ще до їх появи	Високе навантаження на процесор/пам'ять. Складність у сертифікації критичних систем. Потреба у великих обсягах даних та тривалому налаштуванні

1.3 Забезпечення виявлення аномалій і фальсифікацій у КФС та існуючі обмеження

КФС, що забезпечують інтеграцію обчислювальних ресурсів із фізичними процесами, є вразливими до різних типів кібератак, які можуть мати серйозні наслідки для їхньої роботи та безпеки. Ці атаки можуть бути спрямовані на порушення

функціонування системи, викрадення або компрометацію даних, а також захоплення управління системою [90].

Кібербезпека є одним із найважливіших аспектів захисту КФС. Оскільки ці системи часто інтегровані з глобальними мережами або мають доступ до критично важливих інфраструктур, вони стають привабливою мішенню для зловмисників [91].

Загрози, пов'язані зі зловмисним програмним забезпеченням (ЗПЗ), такі: зловмисне програмне забезпечення (віруси, трояни, черв'яки) може проникати в КФС через вразливості у програмному забезпеченні або через інші вектори, такі як соціальна інженерія. Наприклад, атака з використанням зловмисного програмного забезпечення може призвести до викрадення конфіденційних даних або до пошкодження системи, що може вимагати повної зупинки її роботи для відновлення. У випадку успішного зараження, шкідливе програмне забезпечення може поширюватися через мережу, порушуючи роботу пов'язаних систем і підвищуючи ризик масштабних збоїв[18].

Атаки на відмову в обслуговуванні (DoS/DDoS): Атаки на відмову в обслуговуванні (Denial of Service, DoS) і розподілені атаки на відмову в обслуговуванні (Distributed Denial of Service, DDoS) є особливо небезпечними для КФС, які повинні функціонувати безперервно. Ці атаки можуть перевантажити систему великою кількістю запитів, що перевищує її можливості, викликаючи зупинку або значне уповільнення роботи [93]. У критично важливих середовищах, таких як транспорт або медицина, атака DoS може призвести до фатальних наслідків, оскільки система не зможе своєчасно обробляти запити або реагувати на зовнішні сигнали [94]. Кібератаки будемо розглядати винятково на рівні сенсорних даних і вимірювальних сигналів, що надходять до системи та використовуються в алгоритмах оцінювання стану й керування. Такі атаки моделюються як фальсифікації, спотворення або аномальні відхилення вимірів, які надходять через сенсорні канали незалежно від способу їх реалізації.

КФС з інтенсивними змінами стану потребують надійних методів моніторингу, щоб вчасно виявляти збої, аномалії чи навіть цілеспрямовані атаки [8; 148]. У сучасних умовах поширені атаки на цифрові командні сигнали керування, що можуть призводити до неправильного функціонування системи або небезпечних станів. Класичний підхід до виявлення таких проблем полягає в моніторингу вихідних даних системи та порівняння їх з очікуваними значеннями на основі математичної моделі. Фільтр Калмана широко застосовується для оцінювання стану об'єкта в реальному часі та може слугувати основою для виявлення аномалій [95] за рахунок аналізу інновацій, резидуалів, тобто різниці між виміряним і прогнозованим станом [96; 97; 98]. Наприклад, у сфері розумних електричних мереж запропоновано ансамблевий фільтр Калмана, який показав високу ефективність порівняно з традиційними методами при атаках з підробленням даних [99].

Однак стандартний фільтр Калмана має фіксовані параметри і може не враховувати раптові зміни динаміки чи нові типи шумів, що ускладнює чітке розмежування між реальною атакою і просто нестабільністю сигналу [100]. Реальний час та обмежені ресурси вбудованих пристроїв створюють додаткові вимоги. Алгоритм повинен виконуватись швидко, стабільно та займати мінімум обчислювальних ресурсів. Традиційні методи на основі статистичних порогів, наприклад, критерій X^2 для резидуалів або фіксованих порогових значень можуть бути ненадійними в умовах динамічно змінюваних станів, приводячи до численних хибних тривог, пропущених аномалій чи фальсифікацій в умовах впливу ЗПЗ. Новітні підходи пропонують адаптувати фільтр Калмана в ході роботи. Зокрема, онлайн-адаптивне налаштування параметрів фільтра дозволяє йому пристосовуватися до зміни статистичних властивостей сигналів у реальному часі. Дослідження показали, що такий адаптивний підхід може досягати ~95% точності виявлення при дуже швидкій роботі, напр., 0,008 с на вибірку навіть на вузлах з обмеженими ресурсами. Це свідчить про потенціал застосування удосконаленого фільтра Калмана для виявлення аномалій чи фальсифікацій в умовах впливу ЗПЗ в ОСРЧ на практиці [101].

Базовий фільтр Калмана припускає постійність статистик шумів і незмінність моделі системи. У реальних КФС ці умови часто порушуються, зокрема, характеристики шуму й динаміка об'єкта можуть змінюватися з часом [102]. Якщо модель не оновити вчасно, оцінки стану стають неточними, а сам фільтр може розбіжно реагувати на зміни [103; 104]. Наприклад, різкі маневри або стрибкоподібні зміни стану здатні викликати великі залишкові похибки, інновації, що виходять за межі очікуваних рівнів. Це призводить до помилкових оцінок або навіть до втрати стійкості фільтра. Також стандартний фільтр Калмана намагаються поєднувати з іншими алгоритмами, наприклад з авторегресійною моделлю [105; 106]. У випадку атак з підробкою сигналів стандартний фільтр не має механізму відрізнити зловмисне відхилення від справжнього сигналу.

Існують підходи покращення фільтра Калмана за рахунок адаптації його параметрів. Зокрема, онлайн-адаптивні фільтри Калмана динамічно налаштовують коваріації процесу та вимірювань на основі поточних даних [101]. Це дозволяє підтримувати високу якість фільтрації навіть при зміні статистичних властивостей шуму або динаміки об'єкта, що важливо для виявлення аномалій [107]. А також додають роботу нейромереж якщо потрібно [108]. Проте такий підхід має дві суттєві вади:

1) обчислювальне навантаження через безперервне оновлення матриць в реальному часі вимагає додаткових обчислень на кожному кроці, що може бути проблематично для малопотужних контролерів, навіть якщо сам алгоритм залишається рекурсивним [101; 143];

2) затримка адаптації, бо метод поступово підлаштовує параметри, тому при різких стрибках стану або при раптовій атаці може виникнути проміжок часу, поки фільтр навчиться новим умовам, а за цей час аномалія може пройти непоміченою або встигнути вплинути на систему.

Багатомодельні режимні фільтри є відомим підходом для систем зі змінними режимами та ґрунтуються на одночасному використанні кількох моделей динаміки

[110]. Алгоритми на кшталт IMM або MMAE запускають кілька фільтрів Калмана паралельно, причому кожен зі своєю моделлю, і комбінують їх результати [110; 111]. Такий метод покращує ймовірність, що хоча б одна з моделей відповідатиме поточному режиму роботи системи. Однак ціною цього є суттєве зростання обчислень, бо фактично доводиться виконувати кілька фільтрів за кожен такт вимірювання [111]. Крім того, для об'єднання результатів потрібні складні ймовірнісні розрахунки, наприклад, оцінка ваг моделей, апостеріорних ймовірностей режимів тощо [112]. У реальному часі на мікроконтролері це може призводити до перевищення доступного часу виконання або енергобюджету [113].

Робастні методи та фільтри сильного стеження застосовуються для підвищення стійкості оцінювання в умовах невизначеностей моделі. До таких підходів належать, зокрема, H_∞ -фільтри та ФСС [114]. Такі підходи справді роблять фільтр більш робастним до розузгодження моделі і дозволяють виявляти навіть малі імпульсні відхилення з малою затримкою [115; 116]. Проте класичні реалізації ФСС орієнтовані переважно на задачі керування і навігації (напр. стеження за маневровими цілями) і вимагають ретельного налаштування коефіцієнтів згасання під конкретну систему [143]. Для загальних КФС зі швидкими режимними перестроюваннями це не завжди тривіально, а неправильне налаштування може призвести або до надмірної чутливості (фільтр шумітиме, породжуючи хибні тривоги), або до недостатньої, коли аномалії пропускаються.

З проаналізованих методів в ОСРЧ наявні недоліки подано в табл. 1.4.

Отже, наявні методи мають такі недоліки: стандартний фільтр Калмана недостатньо гнучкий до нестабільних умов, адаптивні варіанти підвищують складність і можуть запізнюватись, багатомодельні рішення надто ресурсоємні [117], робастні фільтри складні в налаштуванні.

Таблиця 1.4 – Недоліки існуючих методів оцінювання стану виявлення фальсифікацій

Метод	Опис	Недоліки
Стандартний фільтр Калмана	Використовує постійні статистики шуму і статичну модель системи, ефективний за стабільних умов.	Недостатня адаптивність, висока чутливість до змін станів, не розпізнає зловмисні атаки.
Адаптивний фільтр Калмана (онлайн-адаптація Q, R)	Динамічно налаштовує параметри шумів на основі поточних даних для кращого врахування змін.	Підвищене обчислювальне навантаження, затримки адаптації при різких змінах.
Багатомодельні фільтри (ІММ, ММАЕ)	Паралельне використання кількох фільтрів Калмана із різними моделями з подальшою комбінацією результатів.	Значне зростання обчислювальної складності, велике навантаження на CPU та пам'ять.
Робастні фільтри (ФСС, H_∞ -фільтр Калмана)	Використання коефіцієнтів згасання для швидкої реакції на несподівані зміни або аномалії.	Складність налаштування під конкретні умови, ризик надмірної чутливості або навпаки пропуску аномалій.

1.4 Постановка задачі дослідження

ОСРЧ є особливим класом програмного забезпечення, яке виконує критично важливі завдання у точно визначені часові інтервали. У сучасних технологіях, де кожна секунда має значення, ОСРЧ відіграють ключову роль у багатьох галузях, від авіоніки та автомобільної промисловості до медичних приладів і телекомунікацій [122; 123; 124].

Однією з найважливіших вимог до ОСРЧ є детермінованість і передбачуваність. Це означає, що система повинна виконувати задачі з високою точністю в чітко визначені часові інтервали. Детермінованість забезпечує впевненість у тому, що кожна задача буде виконана вчасно, що є критично важливим для багатьох застосувань.

В автомобільній промисловості, наприклад, система управління двигуном повинна миттєво реагувати на зміни умов руху. Це означає, що будь-яка затримка може вплинути на безпеку і ефективність транспортного засобу. Своєчасне виконання задач у таких системах забезпечує безперервний контроль над процесами і запобігає можливим аваріям.

У медичних приладах детермінованість ще більш критична. Обладнання, яке підтримує життя пацієнтів, повинно обробляти сигнали і реагувати на них у реальному часі. Будь-яка затримка в обробці може призвести до серйозних наслідків для здоров'я пацієнта. Тому ОСРЧ у медичних приладах мають забезпечувати максимально швидку і точну реакцію на змінні умови.

Детермінованість також важлива в авіації, де системи управління польотом повинні безперервно обробляти дані з датчиків і приймати рішення в реальному часі. Найменша затримка може призвести до критичних ситуацій, тому ОСРЧ у цій галузі розробляються з урахуванням високих вимог до точності і передбачуваності виконання задач.

Ефективне управління ресурсами є ще однією важливою вимогою до ОСРЧ. Система повинна оптимально використовувати доступні ресурси, такі як процесорний час і пам'ять, щоб забезпечити максимальну продуктивність. Це потребує використання спеціалізованих алгоритмів планування та управління, що дозволяють розподілити ресурси таким чином, щоб всі задачі були виконані вчасно та ефективно. Наприклад, у системах управління промисловими роботами, де кілька задач повинні виконуватися паралельно, ефективне управління ресурсами забезпечує синхронізовану та безперебійну роботу всіх компонентів. В сучасних телекомунікаційних системах, де обробка великих обсягів даних вимагає високої пропускної здатності і швидкості, ефективне використання ресурсів дозволяє уникнути перевантажень і забезпечити стабільну роботу мережі.

Гнучкість та масштабованість ОСРЧ забезпечують її здатність адаптуватися до змінних умов та вимог. Система повинна легко масштабуватися і адаптуватися до нових задач і підвищеного навантаження, зберігаючи при цьому високу

продуктивність і надійність. Це особливо важливо у сучасних інформаційних системах, де обсяги даних та обчислювальних задач постійно зростають. Здатність ОСРЧ швидко адаптуватися до змінних умов дозволяє ефективно інтегрувати нові функції та розширювати можливості системи без зниження її продуктивності. У галузях, де технологічний прогрес рухається швидкими темпами, така гнучкість дозволяє системам залишатися актуальними та ефективними.

Крім того, ОСРЧ повинні забезпечувати високу стабільність і надійність, щоб гарантувати безперебійне виконання критичних завдань. Використання передових технологій моніторингу та управління дозволяє ОСРЧ вчасно виявляти і усувати потенційні проблеми, забезпечуючи тим самим стабільну роботу системи навіть у складних умовах. Наприклад, впровадження технологій низькопотужних широкосмугових мереж, зокрема мереж на основі LoRa, дозволяє досягати високої енергоефективності та стабільної передачі даних на відстані 3-14 кілометрів для моніторингу датчиків і розподілених систем, що є важливим аспектом ефективного використання ресурсів у великомасштабних інформаційних системах [109].

Розробка архітектури ОСРЧ має одночасно вирішити три взаємопов'язані завдання, які не можуть бути повністю розв'язані лише шляхом вдосконалення класичних алгоритмів планування з фіксованими або динамічними пріоритетами. Перше завдання полягає у підвищенні ефективності, гнучкості та масштабованості під ресурсні й часові обмеження [118]. Обчислювально інтенсивні процедури доцільно винести на зовнішній модуль або у хмарне середовище, а на рівні операційної системи залишити лише легкі онлайн-процедури. Друге завдання стосується забезпечення відмовостійкості без дорогого повного резервування. Доцільним є поєднання апаратного сторожового таймера як останньої лінії захисту з інтелектуальним програмним шаром, який виконує превентивний аналіз стану й локальні перезапуски проблемних компонентів, що скорочує простой та зменшує ризик повного перезапуску. Третє завдання спрямоване на зміцнення захисту та стійкості до атак. Класичні фільтри стану недостатньо надійні в умовах нестабільних режимів, а багатомодельні та робастні рішення часто надмірно складні для легких платформ.

Потрібна інтегрована схема оцінювання стану та виявлення аномалій чи фальсифікацій в умовах впливу ЗПЗ, яка поєднує адаптивність, точність і робастність без надмірних витрат часу та пам'яті. Також потрібно не залишати в стороні що впровадження нових технологій, автоматизація мають бути конкурентними в економічному плані використовуючи доступні напрацювання щодо автоматизації чи швидкості впровадження [119; 120]. Це може вимагати значних змін у програмному забезпеченні або навіть заміни частини апаратного забезпечення, що збільшує витрати і час на впровадження нових рішень [121].

З урахуванням зазначеного формулюються такі задачі дослідження:

1. Узагальнити результати аналізу існуючих архітектур ОСРЧ для КФС з інтенсивними змінами станів та визначити їхні обмеження щодо адаптивності, відмовостійкості та інтеграції інтелектуальних методів обробки даних.

2. Розробити архітектуру планувальника задач ОСРЧ з використанням зовнішнього модуля машинного навчання та зворотного каналу самоадаптації, що забезпечує збирання багатовимірної телеметрії, її тензорне подання та адаптивне оновлення параметрів планування без втручання користувача.

3. Розробити метод динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції багатовимірних даних навантаження, який враховує взаємозв'язки між параметрами задач і ресурсів та забезпечує підвищення ефективності використання обчислювальних ресурсів.

4. Розробити метод превентивного відновлення компонентів ОСРЧ на основі низькорозмірної марковської моделі прогнозування станів у поєднанні з багаторівневим сторожовим контролем, що дозволяє завчасно виявляти деградаційні процеси та ініціювати локальні відновлювальні дії.

5. Розробити метод виявлення фальсифікацій в умовах впливу ЗПЗ та аномалій у сигналах КФС на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів і селективним пригніченням підозрілих вимірів, придатний для реалізації в умовах обмежених ресурсів мікроконтролерів.

6. Провести експериментальну перевірку розроблених методів і архітектурних

рішень на вбудованій платформі під керуванням ОСРЧ та оцінити їх ефективність за показниками використання процесора й пам'яті, часу виконання задач, тривалості простою системи, кількості хибних спрацювань, швидкодії відновлення та обчислювальної складності.

1.5 Висновки до першого розділу

Проведений огляд предметної області дослідження засвідчив обмеження традиційних підходів до планування, кластеризації та забезпечення відмовостійкості в операційних системах реального часу за умов дефіциту ресурсів і жорстких часових вимог. Повне резервування виявилось надмірно затратним для легких платформ, а класичні засоби оцінювання стану виявляють недостатню надійність у нестабільних режимах і за наявності навмисних впливів. Обґрунтовано доцільність гібридної архітектури, яка поєднує АСТ як останню лінію захисту з інтелектуальним програмним рівнем превентивного реагування, що забезпечує локальні перезапуски проблемних компонентів і знижує частоту повних перезавантажень.

Запропоновано використання низькорозмірних марковських моделей для онлайн-оцінювання ризику критичних станів із мінімальними часовими та пам'яттєвими витратами, а також винесення обчислювально інтенсивних процедур навчання і кластеризації на зовнішній модуль або у хмарне середовище, залишаючи в ядрі операційної системи лише легкі процедури прийняття рішень. Сформовано вимоги до робастно-адаптивного контуру оцінювання стану та виявлення аномалій чи фальсифікацій в умовах впливу ЗПЗ, визначено ключові метрики ефективності, зокрема час відновлення, частоту хибних перезапусків, дотримання дедлайнів, латентність детекції та вплив на продуктивність. Таким чином, необхідно здійснити інтеграцію марковського прогнозування, подвійного сторожового механізму та розподіленого інтелектуального аналізу, що стане основою для подальшого синтезу архітектури, удосконалення алгоритмів, побудови прототипу та експериментальної валідації у типових сценаріях КФС.

РОЗДІЛ 2

АРХІТЕКТУРА ПЛАНУВАЛЬНИКА ЗАДАЧ ІЗ ЗАЛУЧЕННЯМ ЗОВНІШНЬОГО МОДУЛЯ ТА МЕТОД ДИНАМІЧНОГО РОЗПОДІЛУ ЗАДАЧ І РЕСУРСІВ НА ОСНОВІ ТЕНЗОРНОЇ ДЕКОМПОЗИЦІЇ

2.1 Особливості архітектури та принципів функціонування модифікованої ОСРЧ для КФС

Для сучасних ОСРЧ керування ресурсами та балансування навантаження є ключовими передумовами стабільності й високої продуктивності. Із поширенням КФС та Інтернету речей (IoT) виникає потреба в оперативному аналізі та ефективній обробці великих масивів даних у реальному часі.

Класична архітектура ОСРЧ для КФС складається з кількох основних компонентів, кожен із яких виконує визначену роль. Ядро системи є центральним елементом, який відповідає за управління задачами, розподіл ресурсів і обробку подій у реальному часі. Планувальник задач забезпечує виконання задач у порядку їхніх пріоритетів. Модулі пам'яті займаються виділенням оперативної пам'яті для задач, а системи обробки переривань і модулі вводу-виводу забезпечують взаємодію із зовнішніми пристроями.

Дані передаються між компонентами через односторонні зв'язки. Наприклад, системи обробки переривань передають події до ядра, яке аналізує ці дані та розподіляє ресурси. Планувальник задач формує черговість виконання задач і повідомляє ядро про зміни. Модулі вводу-виводу надають інформацію про стан периферійних пристроїв. Однак ці дані не накопичуються для подальшого аналізу.

Ця архітектура добре підходить для базових задач реального часу, але має низку обмежень. Відсутність централізованого збору даних ускладнює проведення комплексного аналізу. Планувальник задач працює за статично заданими правилами, не враховуючи змін у навантаженні чи конфліктів ресурсів. Крім того, комунікація між компонентами є фрагментарною, що обмежує адаптивність системи до змін.

Відсутність механізмів прогнозування та машинного навчання робить систему реактивною, а не проактивною. На рис. 2.1 зображено загальну архітектуру, що включає основні компоненти типової ОСРЧ для КФС.

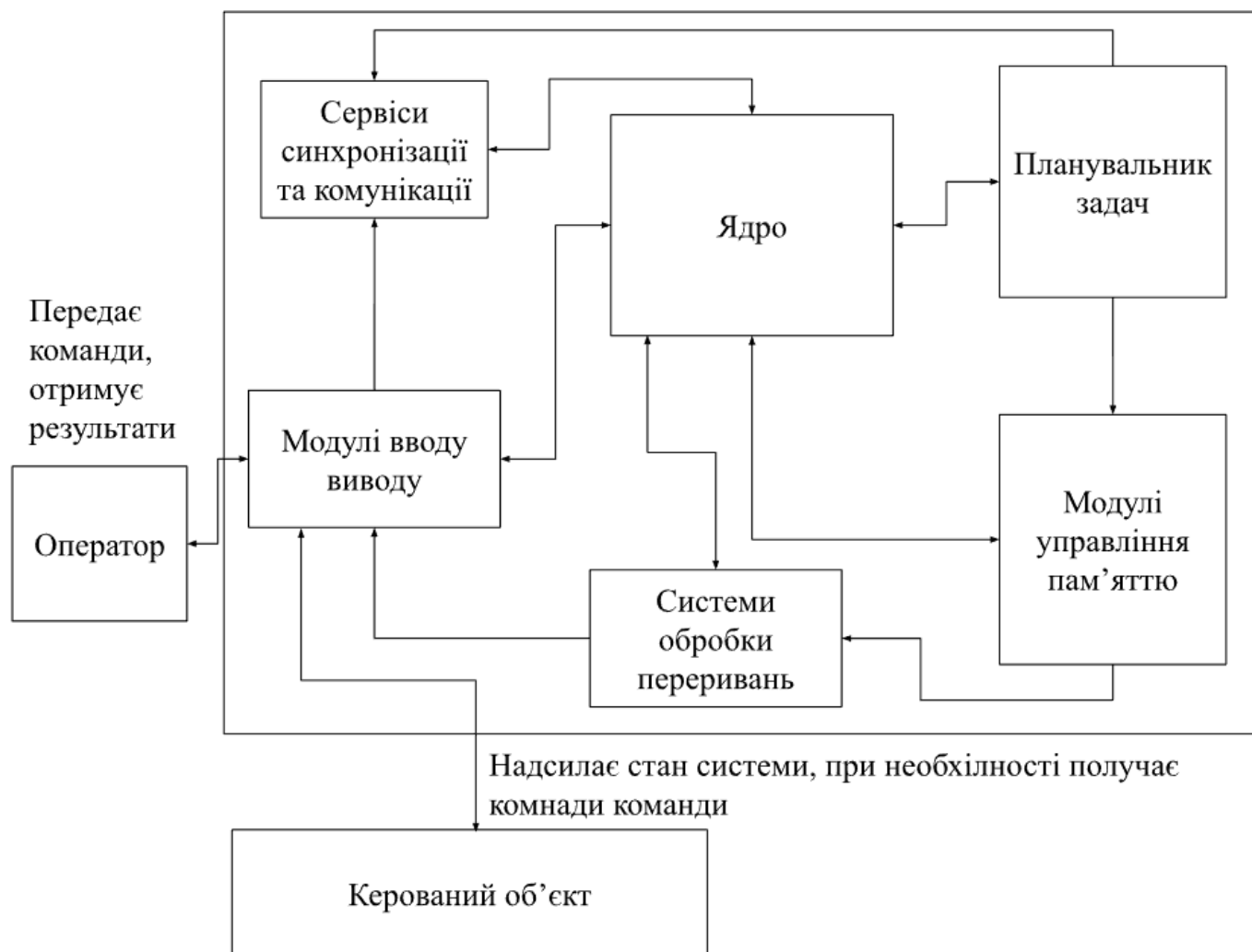


Рисунок 2.1 - Загальна архітектура ОСРЧ для КФС, що включає основні компоненти системи та зв'язки між ними

На рис. 2.2 зображено загальну архітектуру ОСРЧ для КФС у графовому представленні, де вузли відповідають основним компонентам системи, а орієнтовані ребра – зв'язкам між ними.

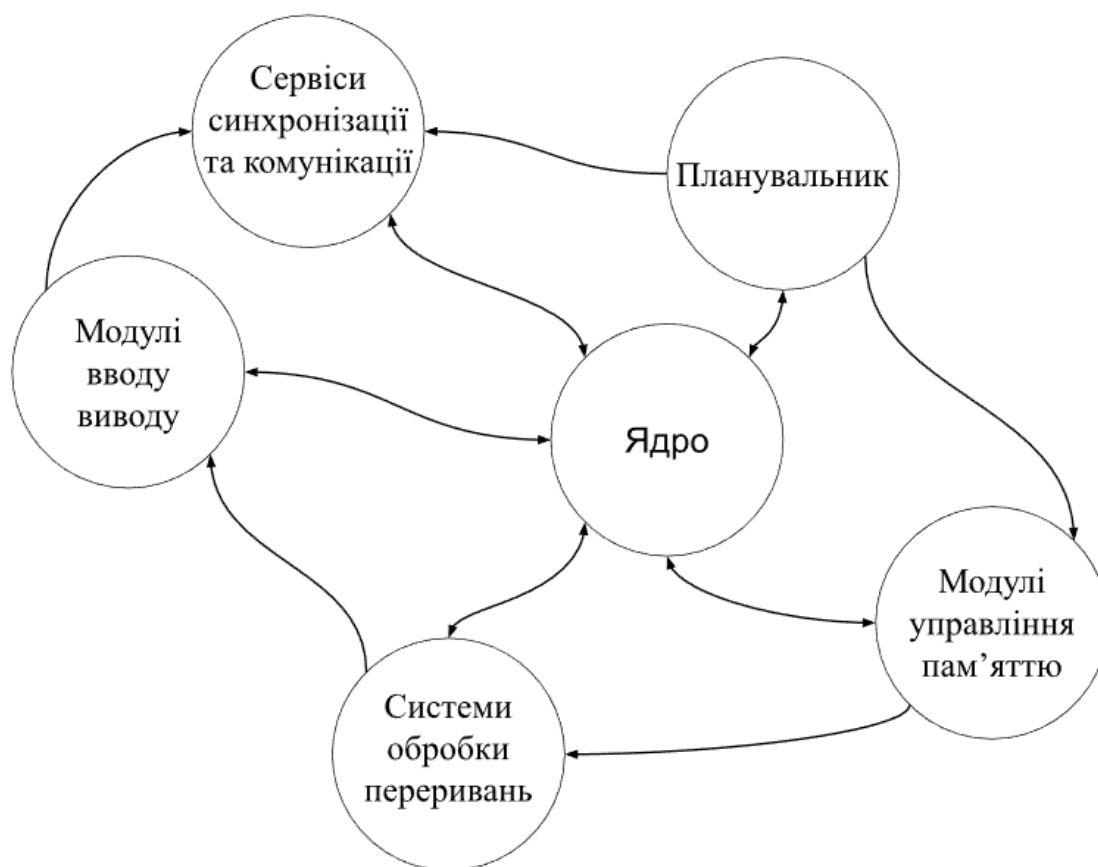


Рисунок 2.2 - Графове представлення архітектури ОСРЧ для КФС із відображенням основних компонентів та їхніх зв'язків

Такі обмеження створюють необхідність у вдосконаленні архітектури для підвищення її адаптивності, ефективнішої роботи і використання сучасних інструментів аналізу даних.

З метою підвищення адаптивності, надійності та передбачуваності функціонування операційної системи реального часу (ОСРЧ) було розроблено модифіковану архітектуру, яка доповнює класичну структуру ядра новими інтелектуальними компонентами. Основна відмінність полягає у переході від реактивної моделі керування, що спрацьовує лише після виникнення події чи збою, до проактивної моделі, здатної передбачати зміну станів системи та попереджати критичні ситуації. Інтенсивні зміни станів розглядаються як такий режим функціонування КФС, за якого зміна її операційних станів відбувається з частотою,

співрозмірною з часовими характеристиками роботи ОСРЧ. Тобто ситуації, коли система не встигає стабілізуватися в одному режимі, оскільки нові зміни навантаження, структури задач або умов керування виникають до завершення реакції ОСРЧ на попередні події. Інтенсивність у цьому контексті означає не абсолютну кількість змін за секунду, а відносну швидкість зміни станів щодо періоду виконання критичних задач і механізмів планування. Якщо перехід між станами відбувається швидше або з тією самою швидкістю, з якою ОСРЧ приймає рішення про розподіл ресурсів, то статично задані параметри планування та реактивні механізми відмовостійкості втрачають адекватність. Для практичних КФС це означає, що зміни станів можуть відбуватися як у межах десятків мілісекунд, так і в інтервалах, що безпосередньо накладаються на цикл роботи ОСРЧ. У таких умовах система перебуває в режимі постійної перебудови, де затримка виявлення деградації, некоректна оцінка сенсорних даних або несвоєчасне перепланування задач призводять до каскадних порушень роботи.

Розроблювана архітектура планується шляхом інтеграції аналітичного модуля на основі тензорної декомпозиції, який забезпечує багатовимірний аналіз параметрів навантаження, часу виконання, споживання ресурсів та міжпроцесних залежностей. Тензорне подання даних дозволяє виявляти приховані закономірності, групи взаємопов'язаних процесів та прогнозувати тенденції зміни стану системи. Це створює основу для адаптивного планування задач і динамічного балансування навантаження між процесами та обчислювальними вузлами.

Для забезпечення достовірності аналізу й підвищення стійкості до спотворених або скомпрометованих сенсорних даних було реалізовано модуль моніторингу цілісності даних на основі комбінованого фільтра Калмана. Цей модуль порівнює очікувані параметри системи з фактичними вимірюваннями, обчислює нормалізовану інновацію та виявляє відхилення, що перевищують статистично допустимі межі. Таким чином, система отримує можливість раннього виявлення дрейфу, шумів або спуфінгу сенсорних сигналів до моменту їх впливу на процес керування. Наприклад,

зломисний зсув або імпульсне спотворення показників висотоміра буде виявлено запропонованим модулем за аналізом інновацій як статистично неприпустиме відхилення, що унеможливить некоректну реакцію системи на хибний вимір висоти. Атака розглядається як фактор, що ініціює аномальні зміни станів КФС, насамперед на рівні сенсорних вимірів. Зловмисний вплив або відмова сенсорів призводять не до негайної зупинки системи, а до порушення статистичних і динамічних властивостей сигналів, що використовуються для оцінювання стану та керування. Типовою метою зловмисника є нав'язування некоректної картини стану об'єкта керування шляхом підміни, зсуву або імпульсного спотворення вимірів, що може викликати помилкові керуючі дії без явних ознак відмови. Наприклад, при підміні барометричних вимірів, додаючи зсув, система вважає, що апарат знаходиться вище реального рівня. При імпульсній атаці на сенсор висоти або тиску система виконує некоректні керуючі дії. Також, при підміні телеметрії створюється ілюзія стабільного навантаження в енергетичній системі. Це призводить до того, що окремі вузли перевантажуються без формального порушення лімітів.

Кібератаки будемо розглядати винятково на рівні сенсорних і вимірювальних даних, що використовуються в алгоритмах оцінювання стану та керування. Атаки моделюються як фальсифікації, імпульсні спотворення або аномальні відхилення вимірів незалежно від способу їх реалізації. Такий підхід дозволяє зосередитись на захисті найбільш вразливої ланки КФС, тобто каналу “датчик, оцінювання, керування”, де навіть незначні спотворення можуть призвести до помилкових керуючих дій. Такий механізм виявлення не залежатиме від природи впливу (кібератака чи апаратна відмова), оскільки базується на статистичній невідповідності між прогнозованими та фактичними вимірюваннями. Аналогічно, фізичне від'єднання датчика призводить до появи відсутніх чи невірогідних даних, які проявляються як різкі нестабільні зміни оціненого стану. Такий збій також своєчасно ідентифікується модулем моніторингу цілісності. Отже, раптові стрибки сигналів, невідповідність між прогнозом і фактичним виміром, а також порушення часової

узгодженості даних слугують ознаками для спрацювання запропонованого методу виявлення незалежно від того, чи зумовлені вони кібератакою, чи апаратною відмовою сенсорів.

Крім того, у структурі модифікованої ОСРЧ необхідне впровадження задачі прогнозного сторожового контролю (ПЗСК), який базується на моделі низькорозмірних ланцюгів Маркова. На відміну від класичного апаратного сторожового таймера (АСТ) з фіксованим тайм-аутом, прогнозуючий таймер повинен оцінювати ймовірності переходів між станами системи (нормальний, передаварійний, аварійний), прогнозувати час до можливої відмови та ініціювати превентивні дії. Наприклад, перезапуск задачі, переключення на резервний канал або зміщення пріоритетів у планувальнику. Це дозволяє скоротити середній час реакції системи на деградаційні процеси та мінімізувати ймовірність критичних збоїв.

У модифікованій архітектурі взаємодія між основними модулями, тобто аналітичним, агрегуючим, моніторинговим і планувальником, організована у вигляді замкненого циклу з прямим і зворотним каналом обміну даними. На кожному такті t_k агрегуючий модуль формує вектор станів системи $x(t_k)$, який містить поточне завантаження процесора, використання пам'яті, кількість активних задач і їхні пріоритети. Цей вектор передається аналітичному модулю, який виконує попередню нормалізацію, прогнозує майбутні навантаження $\hat{R}(t_k + 1)$ і передає результати до планувальника. Планувальник, використовуючи прогноз, динамічно коригує порядок виконання задач та розподіл ресурсів. Моніторинговий модуль у свою чергу відстежує відхилення фактичних параметрів $R^{fact}(t_k + 1)$ від прогнозованих та передає інформацію про похибку $e(t_k + 1)$ до аналітичного модуля для подальшого уточнення моделі. Таким чином може бути сформовано адаптивний зворотний зв'язок, що забезпечить самонавчання системи та зменшення середньоквадратичної похибки прогнозу.

Часова послідовність взаємодії модулів відповідає періоду планування $T_{\text{план}}$ і складається з п'яти фаз:

- 1) збір даних агрегуючим модулем;
- 2) аналіз і прогнозування станів;
- 3) корекція планувальника;
- 4) контроль достовірності результатів;
- 5) адаптація моделі на основі отриманих відхилень.

Загальний час реакції системи визначимо так:

$$\tau_{\Sigma} = \tau_{зб} + \tau_{ан} + \tau_{пл} + \tau_{конт}, \quad (2.1)$$

де τ_{Σ} – загальний час реакції системи, тобто сумарна тривалість одного циклу обробки даних від моменту збору інформації до формування скоригованого плану виконання задач; $\tau_{зб}$ – час збору даних агрегуючим модулем із сенсорів і системних лічильників; $\tau_{ан}$ – час аналітичної обробки, необхідний для виконання тензорного аналізу, декомпозиції та прогнозування ресурсних потреб; $\tau_{пл}$ – час планування, що включає оновлення пріоритетів задач, побудову черги виконання й призначення ресурсів; $\tau_{конт}$ – час контролю, який витрачається на перевірку достовірності результатів, порівняння прогнозованих параметрів з фактичними та передачу відхилень до аналітичного модуля.

При цьому загальний час реакції системи не перевищує $T_{план}$, що гарантує стабільність роботи в режимі реального часу.

На відміну від класичної архітектури ОСРЧ, у якій інформаційні потоки здебільшого мають односторонній характер і спрямовані від задач до планувальника, у розроблюваній архітектурі потрібно реалізувати багато напрямлений обмін даними між компонентами, щоб забезпечити зворотний зв'язок і адаптивне оновлення параметрів системи.

Комунікація між ядром ОС, аналітичним модулем, агрегуючим елементом і модулями контролю необхідно організувати через єдину інформаційну шину, синхронізовану із системним таймером, що гарантуватиме узгодженість часових зрізів.

Інформаційна взаємодія між компонентами може бути здійснена у вигляді трьох основних типів потоків даних, що функціонують паралельно:

1) операційний потік, який відповідає за передачу сигналів і даних, необхідних для безпосереднього виконання задач, і він функціонує в межах ядра ОСРЧ та характеризується мінімальною латентністю, оскільки оперує даними з жорсткими часовими обмеженнями;

2) аналітичний потік, що формується агрегуючим модулем і надходить до аналітичного блоку, містить інформацію про часові характеристики задач, навантаження на ресурси, частоту контекстних перемикань та пріоритети виконання і обробка цього потоку може здійснюватися з нижчим пріоритетом, оскільки вона виконується у фоновому режимі або в зовнішньому аналітичному процесі;

3) потік керування, який реалізує механізм зворотного зв'язку між аналітичним модулем і планувальником, вирішує процес передачі прогнозних параметрів, результатів тензорного аналізу та скоригованих коефіцієнтів адаптації, що визначають оновлені значення пріоритетів задач у наступному циклі планування.

Таке тристороннє розмежування потоків дає змогу підтримувати незалежність аналітичних операцій від критичних у часовому сенсі процесів. Кожен тип потоку має власну частоту оновлення, для операційного потоку вона дорівнює періоду системного такту, для аналітичного – кратна кільком циклам виконання задач, а для керуючого – синхронізована з моментом завершення прогнозного розрахунку.

У результаті ОСРЧ отримує властивості самоузгодженості та самопідлаштування, коли кожен наступний цикл роботи враховує результати попереднього, поступово мінімізуючи відхилення від оптимального режиму функціонування.

Таким чином, розроблена схема потоків даних і механізм взаємодії модулів забезпечують баланс між точністю прогнозу, швидкістю планувальника та стабільністю часових характеристик системи. Це дозволяє розглядати модифіковану

архітектуру як основу для побудови інтелектуальних ОСРЧ, здатних до самоналаштування та адаптації в умовах змінних навантажень.

На рис. 2.3 зображено модифіковану загальну архітектуру із компонентами, які можуть бути удосконалені. Основний акцент сфокусовано на роботі агрегуючого елемента та аналітичного модуля на основі тензорної декомпозиції, прогнозуючого сторожового таймера, що використовує ланцюги Маркова і модуля моніторингу цілісності даних на основі комбінованого фільтра Калмана.

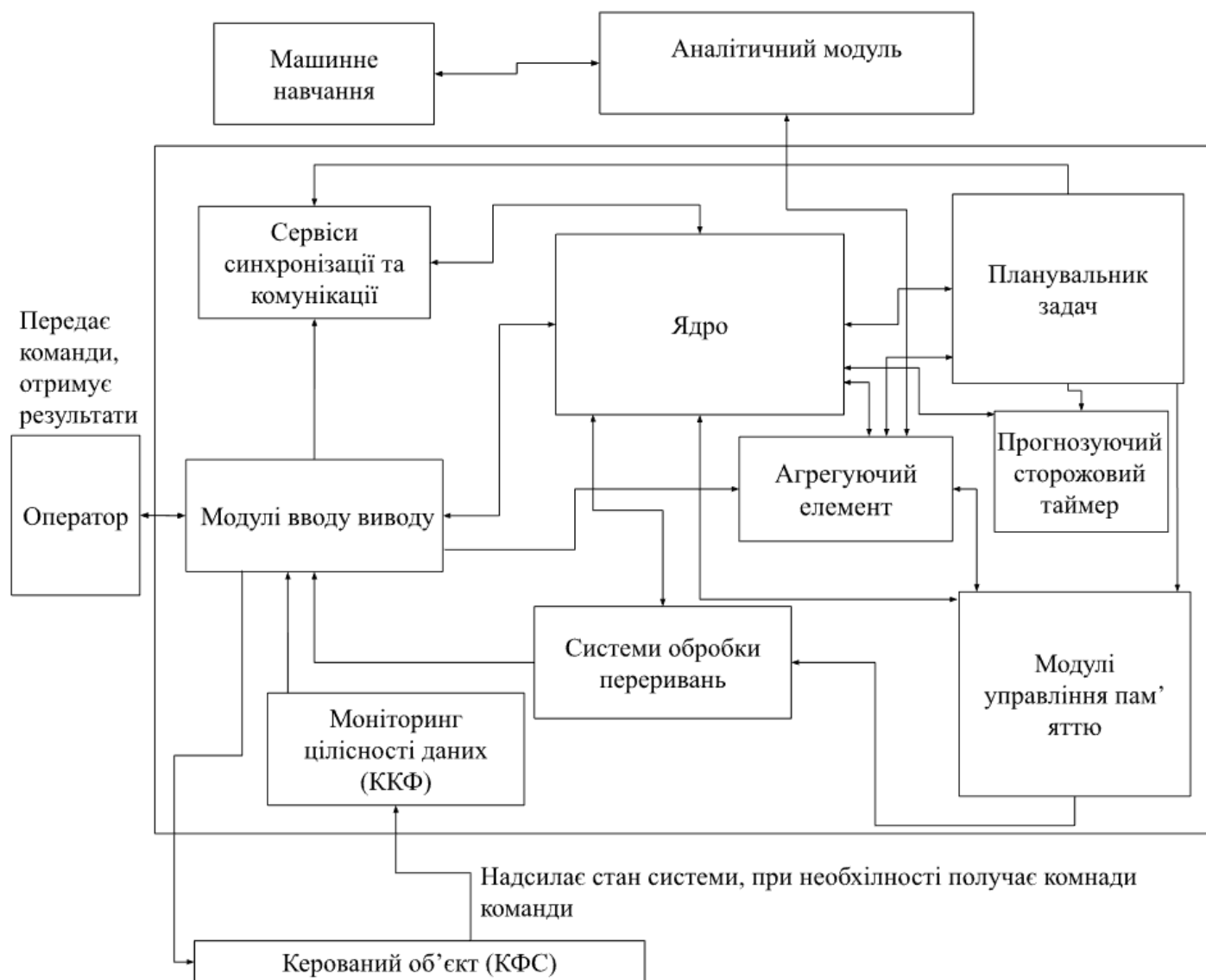


Рисунок 2.3 - Модифікована архітектура ОСРЧ для КФС, що включає основні компоненти системи та зв'язки між ними із новими модулями

На рис. 2.4 зображено загальну архітектуру ОСРЧ для КФС у графовому представленні, де вузли відповідають основним компонентам системи, а орієнтовані ребра – зв'язкам між ними.

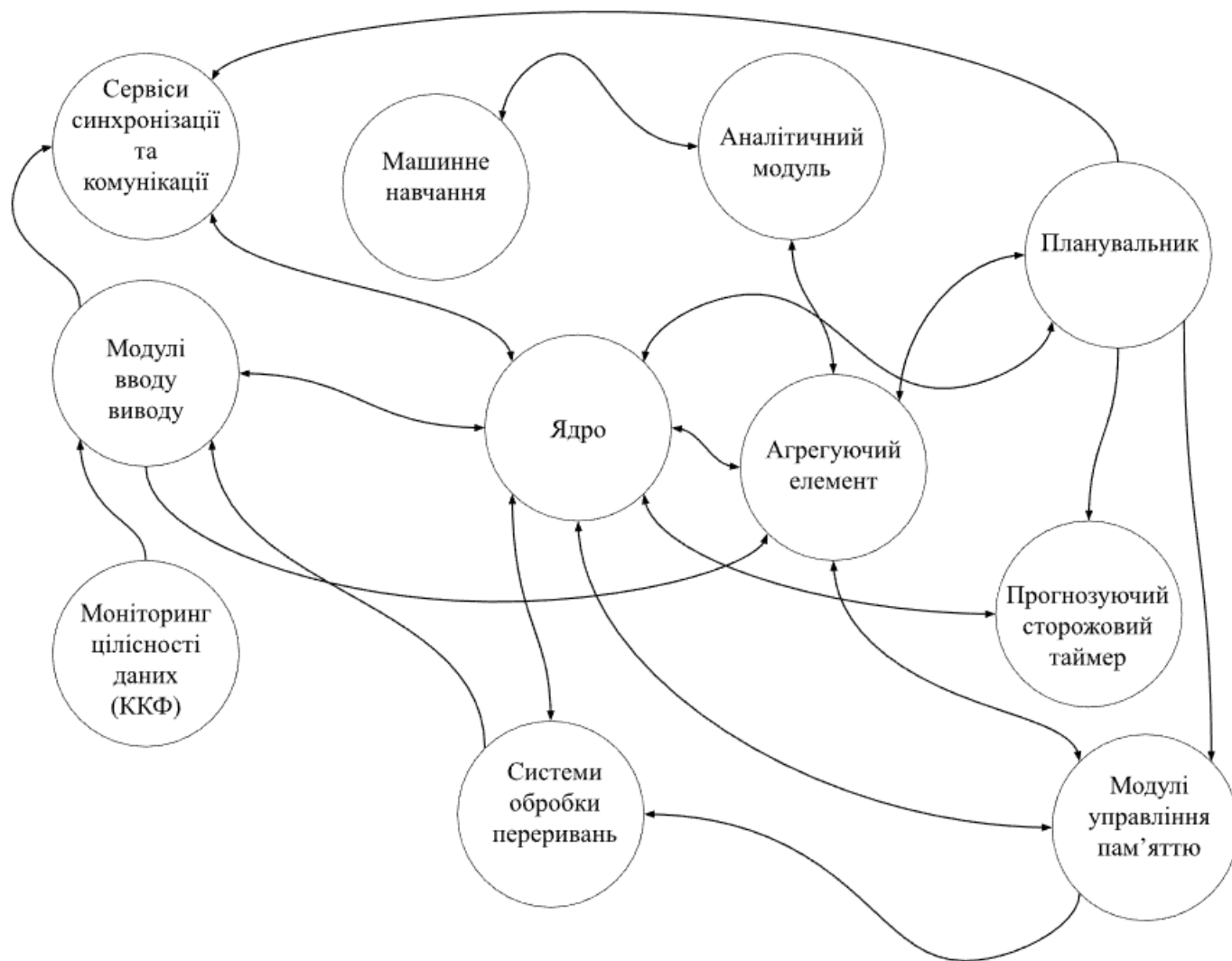


Рисунок 2.4 - Графове представлення модифікованої архітектури ОСРЧ для КФС із відображенням основних компонентів та їхніх зв'язків

Завдяки такій організації взаємодії забезпечується модульність, масштабованість і передбачуваність роботи ОСРЧ. Кожен компонент може оновлюватися незалежно, не порушуючи функціонування ядра, а асинхронний обмін даними між модулями дозволяє ефективно балансувати навантаження й забезпечувати стійкість системи до збоїв у реальному часі.

Таким чином, запропоновані модулі, тобто аналітичний, моніторингу цілісності даних та ПЗСК, формують єдину інтелектуальну підсистему підтримки прийняття рішень у режимі реального часу. Їх спільна робота забезпечує підвищення надійності та безперервності функціонування ОСРЧ, передбачуваність параметрів керування та узгодженість дій операційної системи при інтенсивних змінах стану кіберфізичного середовища. Порівняльна характеристика ефективності класичної та модифікованої архітектур наведена у табл. 2.1 і 2.2.

Таблиця 2.1 – Порівняння класичної та модифікованої архітектури ОСРЧ

Критерій	Класична архітектура ОСРЧ	Модифікована архітектура ОСРЧ з аналітичним і агрегуючим модулями
1	2	3
Принцип збору даних	Локальний, обмежений межами задачі або модуля; дані не зберігаються для подальшого аналізу	Централізований через агрегуючий модуль; усі параметри зберігаються у єдиному сховищі для аналізу та прогнозування
Обробка інформації	Реактивна – дії виконуються лише після виникнення події або переривання	Проактивна – аналітичний модуль аналізує тенденції та формує прогноз майбутніх станів системи
Планування задач	Статичне або напівдинамічне; не враховує динаміку навантажень	Адаптивне, із корекцією пріоритетів на основі результатів тензорного аналізу та машинного навчання
Балансування навантаження	Виконується лише між задачами ядра; немає глобального узгодження навантаження	Виконується глобально на рівні всієї системи; застосовується кластеризація процесів і прогноз навантажень
Комунікація між компонентами	Одностороння або точка-точка; складна синхронізація потоків.	Багатонаправлена, через агрегуючий модуль; взаємодія компонентів відбувається асинхронно з контролем пріоритетів

Кінець таблиці 2.1

1	2	3
Передбачуваність і детермінованість	Залежить від статичних параметрів планувальника; чутлива до збоїв	Підвищена завдяки інтеграції прогнозних моделей і динамічному коригуванню параметрів ядра
Гнучкість і масштабованість	Обмежена жорсткими зв'язками між компонентами системи	Висока – можлива інтеграція нових модулів і сенсорів без зміни ядра ОС
Використання ресурсів	Розподіл фіксований, часто не оптимальний при змінному навантаженні	Адаптивний у реальному часі завдяки аналізу тензорів ресурсних показників

Таблиця 2.2 – Функціональні модулі модифікованої архітектури ОСРЧ

Модуль	Призначення	Основні функції	Очікуваний ефект
1	2	3	4
Аналітичний модуль (на основі тензорної декомпозиції)	Централізований збір і багатовимірний аналіз даних про стан системи	Виконує тензорну декомпозицію показників навантаження, продуктивності, стану пам'яті та процесів; виявляє латентні взаємозв'язки та тенденції	Забезпечує основу для прогнозного планування, адаптивного балансування навантажень і виявлення відхилень у роботі системи
ПЗСК	Превентивне виявлення збоїв і деградації системних процесів	Моделює ймовірності переходів між станами «норма – передаварійний – аварійний», оцінює час до відмови	Забезпечує раннє реагування, скорочення часу простою й підвищення надійності ОСРЧ

Кінець таблиці 2.2

1	2	3	4
Модуль моніторингу цілісності даних (ККФ)	Контроль достовірності та узгодженості сенсорних і системних даних	Реалізує алгоритм комбінованого (подієво-резидуально-адаптивний) фільтра Калмана для аналізу інновацій і виявлення аномалій	Раннє виявлення дрейфу, шумів і спуфінгу; підвищення точності вимірювань і надійності обчислень

2.2 Архітектура планувальника задач із залученням зовнішнього модуля

У контексті удосконаленої архітектури розглянемо використання тензорної декомпозиції та методів машинного навчання, що представляє новий підхід до вирішення проблем класифікації задач і балансування навантаження в ОСРЧ.

Модель планувальника задач із залученням зовнішнього модуля базується на принципі збирання наборів даних, які відправляються у окрему систему, що аналізує дані і на основі цього намагається покращити ефективність управління ресурсами в майбутньому. Використання тензорного подання в архітектурі планувальника зумовлене необхідністю аналізу не окремих параметрів, а їх взаємозалежностей у багатовимірному просторі. У КФС перевантаження або деградація продуктивності часто виникають не через перевищення одного ресурсу, а внаслідок несприятливої комбінації часових, ресурсних і структурних характеристик задач. Тензорне подання дозволяє формалізувати ці взаємозв'язки в єдиній структурі даних, що створює основу для виявлення латентних закономірностей і стабільних шаблонів навантаження. Саме ця властивість робить тензорну декомпозицію придатною для архітектурної інтеграції в контур планування, а не лише для офлайн-аналізу. Таким чином, тензор у межах запропонованої архітектури виступає не математичним ускладненням, а засобом

узгодженого опису стану системи, необхідного для проактивного керування ресурсами.

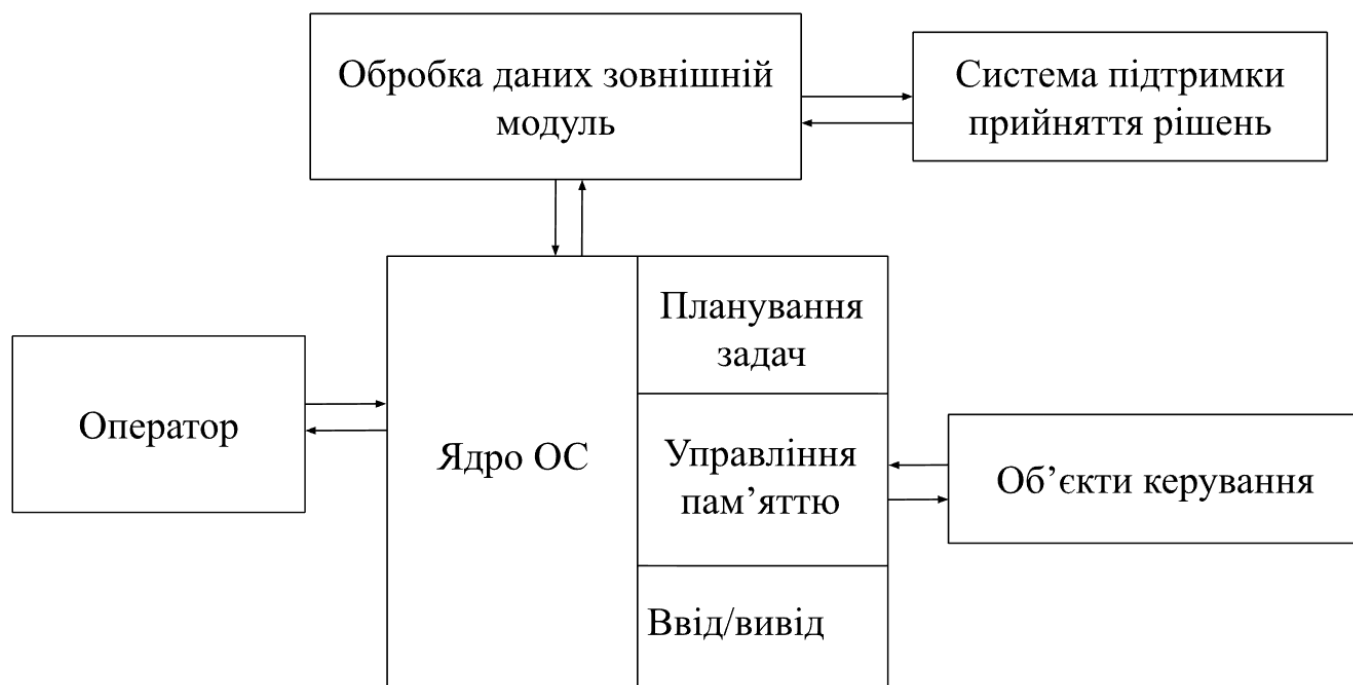


Рисунок 2.5 - Архітектура системи інтелектуальної кластеризації із зовнішнім модулем

Завдяки агрегуючому модулю та аналітичному модулю можна виділити ряд особливостей:

1) централізоване збирання даних (агрегуючий модуль), за якого всі параметри задач (тривалість виконання, рівень використання ресурсів, пріоритети тощо) зосереджуються в єдиному інформаційному просторі, що запобігає втраті або дублюванню даних;

2) адаптивне планування з використанням машинного навчання, де інтелектуальна кластеризація на основі тензорної декомпозиції дозволяє виявляти приховані закономірності у характеристиках виконання задач і на цій основі в режимі реального часу коригувати пріоритети та розподіл ресурсів;

3) прогнозне реагування, яке забезпечується аналізом багатовимірних даних і дає змогу завчасно виявляти вузькі місця та пікові навантаження, підвищуючи ймовірність дотримання часових обмежень;

4) масштабованість і гнучкість архітектури, за якої нові датчики або задачі підключаються до агрегуючого модуля, а аналітичний блок автоматично адаптується до нових потоків даних без суттєвих змін у ядрі системи;

5) ефективне використання ресурсів, при якому планувальник отримує узагальнені дані та рекомендації щодо пріоритетів виконання, що дозволяє уникати неефективного використання обчислювальних ресурсів і забезпечує більш точний їх розподіл.

Взаємодія між планувальником задач і аналітичним модулем у модифікованій архітектурі ОСРЧ реалізується у вигляді замкненого адаптивного циклу управління, у якому аналітичний модуль виступає джерелом прогнозної інформації про стан системи, а планувальник – виконавчим елементом, що здійснює корекцію розподілу ресурсів на основі цих прогнозів. У разі суттєвих змін навантаження або появи нових типів задач аналітичний модуль у періодичному режимі ініціює повторну тензорну декомпозицію для оновленого набору даних, що забезпечує адаптивне оновлення кластерів і періодичне уточнення параметрів системи. Збереження результатів кластеризації в базі даних агрегуючого модуля дає змогу планувальнику працювати не з окремими процесами, а з групами задач із подібними характеристиками, що зменшує обсяг обчислень у реальному часі та скорочує час прийняття рішень. У такій схемі інтелектуальна кластеризація на основі тензорної декомпозиції виконує роль проміжної ланки між аналітичним модулем і ядром ОСРЧ, формуючи узагальнені характеристики задач, які використовуються планувальником для динамічного розподілу ресурсів.

Реалізація замкненого циклу забезпечує динамічну реакцію системи на зміну навантаження, пріоритетів і зовнішніх умов без втрати детермінованості виконання. Аналітичний модуль отримує від агрегуючого компонента дані про поточний стан

процесорних ядер, пам'яті, черги задач, пріоритети, частоту переривань та інші параметри. Ці дані передаються у вигляді багатовимірного тензора, який оновлюється з певною періодичністю Δt . На основі тензорної декомпозиції та навченої моделі машинного навчання аналітичний модуль формує вектор прогнозних параметрів так:

$$\hat{R}(t) = [\hat{r}_1(t), \hat{r}_2(t), \dots, \hat{r}_m(t)], \quad (2.2)$$

де $\hat{r}_i(t)$ – прогнозована потреба у ресурсах i -го типу, наприклад, процесор, пам'ять, канал зв'язку на наступний інтервал часу; m – кількість типів ресурсів у системі; t – дискретний момент часу або період планування.

Планувальник отримує прогнозні значення $\hat{R}(t)$ та оновлює таблицю пріоритетів задач $P_i(t)$, виконуючи корекцію порядку планування згідно з очікуваним навантаженням. Для кожної задачі t_i розраховується адаптивна вага пріоритету так:

$$w_i(t) = \alpha P_i + \beta \frac{\hat{r}_i(t)}{R_{\text{сер}}} + \gamma \frac{C_i}{D_i}, \quad (2.3)$$

де $w_i(t)$ – поточна зважена величина пріоритету задачі t_i у момент часу t ; α, β, γ – коефіцієнти значущості; P_i – базовий пріоритет задачі, визначений планувальником або користувачем; $\hat{r}_i(t)$ – прогнозована потреба у ресурсах для задачі t_i ; $R_{\text{сер}}$ – середнє навантаження на систему за попередній інтервал часу; C_i – обчислювальна складність задачі; D_i – період виконання задачі, тобто граничний момент часу її завершення; $\frac{C_i}{D_i}$ – нормоване співвідношення, що враховує співвідношення складності задачі до доступного часу її виконання.

Таким чином, система автоматично зміщує пріоритети у бік задач із прогнозованими піковими навантаженнями або зростаючим ризиком перевищення часу виконання.

Після виконання задач планувальник передає до аналітичного модуля зворотний зв'язок у вигляді фактичних значень споживаних ресурсів R_i^{fact} та часу виконання

T_i^{fact} . На основі цього визначимо вектор похибки прогнозу, який використовується аналітичним модулем для адаптації моделі прогнозування, так:

$$e_i = \hat{r}_i(t) - R_i^{fact}, \quad (2.4)$$

де e_i – похибка прогнозу для задачі t_i ; R_i^{fact} – фактичне значення використаного ресурсу під час виконання задачі; $\hat{r}_i(t)$ – прогнозоване значення ресурсу, сформоване аналітичним модулем.

Зменшення середньоквадратичної похибки прогнозу визначимо так:

$$E = \frac{1}{n} \sum_{i=1}^n e_i^2, \quad (2.5)$$

де E – інтегральна середньоквадратична похибка прогнозу, що характеризує точність моделі та використовується як узагальнена оцінка якості прогнозу; n – кількість задач, врахованих у циклі планування; $\sum e_i^2$ – сума квадратів похибок, що визначає сумарне відхилення усіх прогнозів від реальних значень за цикл планування; e_i – індивідуальна похибка прогнозу для задачі t_i , що визначена у формулі (2.4), тобто різниця між прогнозованим і фактичним значенням ресурсу.

Зменшення значення E у наступних ітераціях свідчить про самонавчання системи та підвищення точності оцінювання станів. Цикл взаємодії має три етапи:

- 1) прогнозування, під час якого аналітичний модуль формує прогноз майбутнього стану системи на основі історичних і поточних даних;
- 2) адаптивне планування, у межах якого планувальник коригує порядок виконання задач відповідно до отриманого прогнозу;
- 3) зворотний зв'язок і самонавчання, де результати виконання задач використовуються для уточнення моделей прогнозування.

Завдяки такій взаємодії формується динамічна екосистема управління ресурсами, у якій аналітичний модуль виступає як когнітивне ядро системи, що навчається на власних даних, а планувальник забезпечує оперативну реакцію в

реальному часі. Це дозволяє досягти не лише стабільності, але й еволюційної здатності операційної системи адаптуватися до змін зовнішнього середовища та внутрішніх характеристик навантаження.

Винесення аналітичного модуля за межі ядра ОСРЧ є принциповим архітектурним рішенням, зумовленим обмеженнями мікроконтролерних платформ і жорсткими вимогами реального часу. Безпосередня інтеграція тензорної декомпозиції, машинного навчання та прогнозних моделей у ядро ОСРЧ призводить до зростання латентності планування та ризику порушення дедлайнів. У запропонованій архітектурі аналітичний модуль не входить до критичного контуру реального часу та функціонує асинхронно. ОСРЧ використовує лише результати аналізу у компактному вигляді (коефіцієнти, кластери, прогнозні параметри), що не потребує виконання обчислювально складних операцій у режимі реального часу. Такий підхід дозволяє поєднати інтелектуальні методи аналізу з детермінованою поведінкою ОСРЧ і є ключовою умовою практичної реалізації запропонованої архітектури у вбудованих КФС.

У разі втрати зв'язку операційна система реального часу залишається повністю працездатною та переходить до детермінованого режиму, використовуючи останні валідні параметри планування й керування, сформовані в результаті збирання телеметрії, її тензорної декомпозиції та подальшого навчання на основі отриманих латентних ознак. У разі, якщо процес тензорної декомпозиції взагалі не виконувався, планувальник функціонує роботу у штатному режимі з базовими статичними налаштуваннями, які були до оновлення роботи планувальника після запуску першого циклу тензорної декомпозиції, інференсу, модифікацією базових параметрів і застосування відповідних політик роботи планувальника. Також втрата зв'язку із зовнішнім модулем не призводить до зупинки задач, порушення дедлайнів або деградації керування, а лише тимчасово блокує подальшу адаптацію параметрів до моменту відновлення каналу. Жодне рішення реального часу не залежить від негайної відповіді аналітичного модуля. Втрата каналу зв'язку призводить лише до

тимчасового призупинення процесу самоадаптації, але не до зупинки задач або порушення часових обмежень. Таким чином, зв'язок із зовнішнім аналітичним модулем має допоміжний характер і не є критичним для безперервної роботи КФС. Жодне рішення реального часу не залежить від миттєвої відповіді зовнішнього модуля.

Щодо можливості перенесення зовнішнього аналітичного модуля всередину КФС, у межах запропонованої архітектури аналітичний модуль розглядається як компонент, що не входить до критичного контуру реального часу та призначений для обробки агрегованої телеметрії й формування рекомендацій щодо адаптації параметрів планування. Базова реалізація передбачає винесення цього модуля за межі мікроконтролерної платформи, що дозволяє зберегти детермінованість виконання задач ОСРЧ та уникнути негативного впливу складних обчислень на часові обмеження. При цьому, архітектура допускає можливість інтеграції аналітичного модуля безпосередньо в склад КФС для окремого класу платформ із достатнім обчислювальним ресурсом. У такому випадку модуль реалізується як допоміжний фоновий компонент із пониженим пріоритетом, не бере участі у формуванні рішень реального часу, так як працює лише як модифікатор роботи планувальника. Такий підхід дозволяє адаптувати архітектуру до конкретних апаратних обмежень без порушення її базових принципів.

Інтеграція методу динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції у модель планувальника забезпечує двонаправлену взаємодію між аналітичним модулем і ядром ОСРЧ. Агрегуючий елемент передає до аналітичного модуля узагальнені дані про стан усіх задач і ресурсів, після чого, використовуючи результати тензорного аналізу, формуються прогнозні параметри $\hat{R}(t_k + 1)$, визначені у формулі (2.2). Планувальник застосовує ці параметри для корекції черги задач і пріоритетів, що дозволяє забезпечити баланс навантаження в реальному часі.

Взаємодія між модулями відбувається у вигляді замкненого контуру зворотного зв'язку. У прямому напрямку аналітичний модуль передає прогностні значення, а у зворотному – отримує фактичні результати виконання задач та похибку прогнозу, визначену за формулою (2.4). На основі цієї інформації модель коригує власні параметри, мінімізуючи середньоквадратичну похибку прогнозу E (формула (2.5)). Таким чином, система набуває властивостей самонавчання, забезпечуючи поступове підвищення точності прогнозів без втрати детермінізму.

Механізм динамічного коригування пріоритетів задач ґрунтується на адаптивній залежності (формула (2.3)), де планувальник змінює порядок виконання задач у відповідності до прогнозованих навантажень. Такий підхід дозволяє забезпечити рівномірний розподіл ресурсів, уникати перевантажень процесора і підтримувати стабільний час реакції системи. Особливістю інтеграції є розподіл обчислювальних навантажень між ядром ОСРЧ і зовнішнім аналітичним модулем. Тензорна декомпозиція та кластеризація виконуються поза межами ядра, тоді як у системі реального часу здійснюється лише застосування прогностних результатів. Це дозволяє зберігати передбачуваність поведінки ОСРЧ і стабільність часу реакції, визначеного у підрозділі. Обмін даними між модулями реалізується через спільну пам'ять або чергу повідомлень, синхронізовану із системним таймером. Інтеграція розробленого методу робить планувальник не лише реактивним, а й проактивним елементом архітектури ОСРЧ. Система отримує здатність прогнозувати зміни стану навантаження, адаптувати пріоритети задач і балансувати ресурси до виникнення критичних ситуацій. Це зменшує кількість контекстних перемикань, підвищує ефективність використання процесорного часу та сприяє підвищенню стійкості системи до пікових навантажень. Реалізований зворотний зв'язок між аналітичним і планувальним модулями забезпечує гнучку перебудову параметрів системи без необхідності її перезапуску, що є ключовою перевагою модифікованої архітектури ОСРЧ.

У існуючих ОСРЧ критично важливо ефективно розподіляти ресурси й вирівнювати навантаження, щоб гарантувати надійну й швидкодіючу роботу. Розвиток КФС і Інтернету речей зумовлює необхідність безперервної аналітики та ефективного керування потоками даних у реальному часі. У цьому контексті використання тензорної декомпозиції та методів машинного навчання [125] представляє новий підхід до вирішення проблем класифікації задач і балансування навантаження в ОСРЧ.

2.3 Метод динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції

Метод реалізуємо послідовністю етапів. Суть методу полягає у використанні тензорного подання телеметричних даних для побудови моделі, здатної прогнозувати майбутній стан навантаження системи та визначати оптимальний розподіл ресурсів на основі поточних параметрів її функціонування.

У межах методу задачі класифікуються за їх характеристиками, що дає змогу визначати задачі, які потребують негайного виконання, та задачі, виконання яких може бути відкладене без порушення часових обмежень. На основі отриманих ознак модель прогнозує майбутнє навантаження і забезпечує динамічний розподіл задач між процесорами або ядрами для підтримання збалансованого використання ресурсів. Реалізація методу включає такі основні етапи.

Етап 1. Збір даних.

У процесі функціонування системи здійснюється збір телеметричних даних про виконувані задачі та доступні ресурси. До таких даних належать час виконання задач, завантаження процесора, використання пам'яті, пріоритети, частота активації задач, а також взаємозалежності між ними. Збір даних виконується паралельно з основним виконанням задач у режимі реального часу та не впливає на критичні часові характеристики системи, оскільки обчислювально складна обробка може бути

винесена за межі ядра ОСРЧ. Наприклад, у зовнішній аналітичний модуль або окремий мікросервіс.

2. Періодичний аналіз.

Накопичені дані обробляються у подієвому або періодичному режимі з використанням тензорного подання багатовимірної інформації про стан системи. Тензорна декомпозиція дозволяє виділити латентні структури та характерні шаблони навантаження, зменшуючи розмірність задачі аналізу. Аналітичний етап виконується з інтервалом, що не накладає жорстких часових обмежень, наприклад, кожні кілька хвилин або годин, що забезпечує актуалізацію моделей без втручання в роботу системи реального часу.

3. Навчання моделі.

Результати тензорної декомпозиції використовуються як вхідні ознаки для навчання або коригування моделей машинного навчання, призначених для прогнозування майбутніх потреб у ресурсах. Навчання виконується поза контуром реального часу та може включати класифікацію задач, оцінювання ймовірних сценаріїв навантаження та визначення оптимальних стратегій розподілу ресурсів. Таким чином формується узагальнена модель поведінки системи в умовах змінного навантаження.

4. Використання результатів.

Отримані прогнозні результати передаються до планувальника у компактному вигляді, тобто у формі оновлених коефіцієнтів, правил або кластерів задач. У режимі реального часу планувальник використовує ці дані для коригування пріоритетів і розподілу задач між процесорами або ядрами без виконання складних обчислень. Це дозволяє забезпечити проактивне управління ресурсами, зменшити ризик перевантажень і зберегти передбачуваність часових характеристик ОСРЧ.

Початковим етапом у використанні підходу є збір даних, що містять інформацію про задачі, ресурси, часові параметри та інші важливі фактори. Це можуть бути показники тривалості виконання задач, використання ресурсів, пріоритети,

взаємозв'язки між задачами та інші ключові характеристики. Відбувається дана операція у агрегуючому модулі.

Кожна задача пов'язується з відповідними параметрами, які відображають його вимоги та поведінку в системі.

Нехай $X = \{x_1, x_2, \dots, x_n\}$ – множина задач. Кожна задача $x_i \in X$ описується набором характеристик $F_i = \{f_1, f_2, \dots, f_k\}$, де f_j представляє один з параметрів, таких як час виконання, споживання ресурсів, пріоритети та взаємозв'язки.

Основним завданням є збір даних, що визначають набір ознак F_i для кожної задачі x_i , тобто кожна задача пов'язується з набором характеристик так:

$$F_i = \{f_1(x_i), f_2(x_i), \dots, f_k(x_i)\}, \forall x_i \in X, \quad (2.6)$$

де $X = \{x_1, x_2, \dots, x_n\}$ – множина всіх задач, що виконуються в операційній системі реального часу, тобто кожен елемент x_i відповідає окремій задачі, яке має свої характеристики; $F_i = \{f_1, f_2, \dots, f_k\}$ – множина функцій або параметрів, що описують характеристики задачі x_i ; $f_j(x_i)$ – значення j -го параметра (ознаки) для задачі x_i .

Кожен параметр може відображати певну властивість, наприклад $f_1(x_i)$ – час виконання задачі, $f_2(x_i)$ – рівень пріоритету, $f_3(x_i)$ – споживання процесорного часу, $f_4(x_i)$ – обсяг пам'яті, що використовується, $f_5(x_i)$ – взаємозв'язки з іншими задачами або залежності від ресурсів.

Перший крок у розробці алгоритму полягає у створенні тензора X , який містить багатовимірну інформацію про задачі, ресурси та час, тощо.

Нехай X – це тензор розмірності $I \times J \times K$, який містить дані про різні аспекти задач і ресурсів у ОСРЧ. Кожен елемент тензора X_{ijk} представляє значення певного параметра для i -ї задачі, j -го ресурсу і k -го часу.

$$X = [X_{ijk}], \text{ де } i = 1, \dots, I, j = 1, \dots, J, k = 1, \dots, K, \quad (2.7)$$

де X – тут тривимірний тензор, який зберігає багатовимірну інформацію про задачі, ресурси та часові параметри в операційній системі реального часу, він є основною структурою даних для подальшого аналізу, кластеризації та ефективного опрацювання; X_{ijk} – елемент тензора, що представляє значення певного параметра або метрики для i -ї задачі, j -го ресурсу та k -го моменту часу (наприклад, це може бути споживання процесорного часу i -ю задачею на j -му ресурсі в момент часу k); i – індекс задачі, що визначає порядковий номер задачі у множині всіх задач системи; I – загальна кількість задач у системі; j – індекс ресурсу, який вказує на ресурс, що розглядається (процесор, пам'ять, канал зв'язку тощо); J – загальна кількість ресурсів, доступних у системі; k – індекс часу, що відображає дискретний момент або інтервал часу, у який здійснюється спостереження або вимірювання параметрів; K – кількість часових відрізків або моментів спостереження, протягом яких збираються дані.

Таким чином, тензор X розмірності $I \times J \times K$ описує багатовимірну взаємодію між задачами, ресурсами та часом. Кожен його елемент містить числове значення певного параметра, що дозволяє відстежувати динаміку змін навантаження системи у процесі виконання задач у реальному часі:

$$X \in R^{I \times J \times K}, \quad (2.8)$$

де R – множина дійсних чисел; I – кількість задач; J – кількість ресурсів; K – кількість часових інтервалів.

Перед виконанням тензорної декомпозиції здійснимо нормування даних так:

$$X'_{ijk} = \frac{X_{ijk} - \min(X)}{\max(X) - \min(X)}, \quad (2.9)$$

де X_{ijk} – вихідне значення елемента тензора, що відображає певний параметр для i -ї задачі, j -го ресурсу та k -го моменту часу (наприклад, рівень завантаженості процесора певною задачею або обсяг використаної пам'яті в конкретний момент); X'_{ijk} –

нормоване значення відповідного елемента тензора після масштабування, яке приводиться до інтервалу від 0 до 1, що забезпечує порівнянність даних із різних діапазонів; $\min(X)$ – мінімальне значення серед усіх елементів тензора X , що визначає нижню межу діапазону значень; $\max(X)$ – максимальне значення серед усіх елементів тензора X , що визначає верхню межу діапазону значень; i – індекс задачі, який приймає значення від 1 до I ; j – індекс ресурсу, який приймає значення від 1 до J ; k – індекс часу, який приймає значення від 1 до K .

Для виділення основних компонентів та зменшення розмірності багатовимірних даних, поданих у вигляді тензора, застосуємо метод СР-декомпозиції (CANDECOMP/PARAFAC) [125]. Подальший теоретичний опис подано для СР-моделі. Вхідними даними методу є тензор X , що містить багатовимірну інформацію про задачі, ресурси та час, а результатом є набір факторних векторів, які описують основні компоненти даних і можуть бути використані для подальшого аналізу. Вона дозволяє розкласти тензор на компактну суму ранк-одиничних компонент, що спрощує його аналіз та обробку. Основна ідея СР-декомпозиції полягає у представленні багатовимірного тензора у вигляді суми компонент, кожна з яких є зовнішнім добутком векторів, що відповідають окремим вимірам тензора. Практична реалізація СР-декомпозиції зазвичай виконується ітераційним методом почергових найменших квадратів, який послідовно оновлює матриці факторів до досягнення мінімальної похибки відновлення тензора. Припустимо, у нас є тензор X порядку N , який представляє дані з N вимірами. Для кожного виміру n існує певна кількість факторів $a_r^{(n)}$, де $r = 1, 2, \dots, R$ – це кількість ранк-одиничних компонент. Тензор X задамо так:

$$X \approx \sum_{r=1}^R a_r^{(1)} \circ a_r^{(2)} \circ \dots \circ a_r^{(n)}, \quad (2.10)$$

де $\circ$ – оператор зовнішнього добутку; $a_r \in R_n^I$ – вектор факторів для n -го виміру; R – ранг тензора, мінімальна кількість ранк-одиничних компонент для точного або наближеного представлення X .

У випадку багатовимірною тензора X , кожен елемент тензора можна виразити через вектори факторів усіх вимірів:

$$x_{i_1, i_2, \dots, i_N} \approx \sum_{r=1}^R a_{i_1, r}^{(1)} \circ a_{i_2, r}^{(2)} \circ \dots \circ a_{i_N, r}^{(N)}, \quad (2.11)$$

де $x_{i_1, i_2, \dots, i_N}$ – елемент тензора для індексів $i_1, i_2, \dots, i_N$; $a_{i_n, r}^{(n)}$ – елемент вектора $a_r^{(n)}$, що відповідає n -му виміру.

Цей підхід узагальнює концепцію тензорного розкладу для довільної кількості вимірів N . На основі отриманих у результаті декомпозиції векторів факторів формуються вектори характеристик задач, які використовуються для їх подальшої класифікації. Нехай z_i – це вектор характеристик для i -ї задачі, сформований з відповідних факторів декомпозиції. Отримані компоненти використовуються для навчання моделі машинного навчання, яка буде використовуватись для класифікації задач та прогнозування потреб ресурсів. Модель машинного навчання $f(z_i)$ навчається на основі векторів факторів для прогнозування класу задачі або її пріоритету. Нехай y_i – це клас задачі, тоді:

$$y_i = f(z_i), \quad (2.12)$$

де z_i – вектор характеристик для задачі i ; y_i – клас задачі.

Для динамічного балансування навантаження використаємо модель для прогнозування майбутніх ресурсних потреб на основі поточного стану системи. Нехай $r_j(t)$ – це ресурсні потреби для j -го ресурсу в момент часу t . Модель прогнозування $g(z_i, t)$ визначатиме ці потреби так:

$$r_j(t) = g(z_i, t), \quad (2.13)$$

де r_j – обсяг ресурсів, необхідний для j -го ресурсу в момент часу t (наприклад, споживання процесорного часу, пам'яті або енергії); $g(z_i, t)$ – модель прогнозування, яка визначає майбутні потреби в ресурсах на основі поточного стану системи, тобто вона враховує вплив параметрів задачі z_i та часу t ; z_i – вектор ознак або параметрів i -ї задачі, що описує його характеристики: пріоритет, тип обчислення, попередній час виконання, обсяг даних тощо; t – дискретний момент часу, для якого здійснюється прогнозування потреби в ресурсах.

Для забезпечення рівномірного навантаження сформулюємо задачу оптимального розподілу задач між ресурсами так:

$$\min \sum_{j=1}^J |\sum_{i=1}^I r_j(t) - \bar{r}_j|, \quad (2.14)$$

де $\bar{r}_j$ – це середнє навантаження на j -й ресурс, яке розраховується для оцінки рівномірності розподілу, тобто для розв'язання цієї задачі можна застосувати лінійне програмування, субградієнтний спуск або інші методи розв'язання задачі залежно від властивостей моделі g [126; 127]; I – кількість задач, які розподіляються між ресурсами; J – кількість ресурсів у системі; $\sum_{i=1}^I r_j(t)$ – сумарне навантаження на j -й ресурс у момент часу t , отримане в результаті виконання всіх задач; $\min$ – оператор мінімізації, який визначає цільову функцію, мета якої полягає в мінімізації різниці між фактичним і середнім навантаженням на всі ресурси, забезпечуючи рівномірне балансування системи.

Після виконання тензорної декомпозиції формуємо простір ознак, у якому кожному задачу описуємо вектором факторних характеристик. Для оцінки схожості між задачами використаємо косинусну міру подібності так:

$$\text{sim}(t_i, t_j) = \frac{\langle x_i, x_j \rangle}{\|x_i\| \|x_j\|}, \quad (2.15)$$

де x_i, x_j – вектори факторних ознак для задач t_i, t_j ; $\langle x_i, x_j \rangle$ – скалярний добуток векторів ознак; $\|x_i\|, \|x_j\|$ – евклідові норми векторів, що визначають довжину

векторів у факторному просторі; $\text{sim}(t_i, t_j)$ – числовий коефіцієнт подібності, який набуває значень від 0 до 1.

Якщо значення $\text{sim}(t_i, t_j)$ перевищує заданий поріг подібності λ_{sim} , задачі t_i, t_j вважаються подібними і належать до однієї групи G_k . Чим ближче значення $\text{sim}(t_i, t_j)$ до одиниці, то тим більша схожість профілів навантаження задач, тобто вони мають близькі характеристики споживання ресурсів та часової активності. Таким чином, кластери формуються за принципом подібності профілів ресурсоспоживання, часових характеристик і взаємозалежностей між процесами, що дозволяє виділяти групи задач із подібною поведінкою в реальному часі.

Для оцінювання стабільності кожного кластера використаємо середній показник похибки прогнозу, який є частковим випадком середньоквадратичної похибки (формула (2.5)), але обчислюється для підмножини задач, що належать кластеру G_k :

$$R_k = \frac{1}{|G_k|} \sum_{i \in G_k} e_i^2. \quad (2.16)$$

де R_k – інтегральна оцінка похибки для k -го кластера; e_i – індивідуальна похибка прогнозу для задачі t_i , описана у формулі (2.4); $|G_k|$ – кількість задач у кластері.

Кластери з високим значенням R_k позначаються як нестабільні та підлягають уточненню або розподілу, що дозволяє системі підтримувати оптимальну структуру кластерів у динамічному середовищі.

Оновлення політик керування планувальником здійснюється шляхом вибору режиму функціонування системи на основі значень коефіцієнтів α_{cpu} , α_{ram} та α_{exes} отриманих у результаті тензорної декомпозиції телеметричних даних. На основі їх співвідношення визначається домінуючий ресурс або їх комбінація, що дозволяє класифікувати поточний стан системи як штатний, процесорний, пам'яттєвий, часовий або деградаційний. Відповідно до визначеного стану застосовується одна з політик керування параметрами виконання задач.

1. Політика штатного режиму. Якщо всі коефіцієнти мають низькі значення і жоден ресурс не домінує, система працює у базовому режимі. У цьому випадку параметри задач залишаються незмінними, а планування виконується відповідно до стандартної конфігурації.

2. Політика процесорного перевантаження. Домінування коефіцієнта α_{cpu} , інтерпретується як перевантаження процесора. У цьому випадку зменшується активність фонових або некритичних задач, що забезпечує пріоритетний доступ критичних задач до процесорного часу.

3. Політика обмеження використання пам'яті. Домінування коефіцієнта α_{ram} свідчить про підвищене використання пам'яті. У цьому випадку зменшується інтенсивність створення нових некритичних задач та обмежується використання додаткових буферів і тимчасових структур даних.

4. Політика контролю часу виконання задач. Переважання коефіцієнта α_{exec} вказує на зростання тривалості виконання задач. У цьому випадку зменшується безперервна інтенсивність виконання задач із надмірним часом обчислення, що дозволяє стабілізувати часові характеристики системи.

5. Політика деградаційного режиму. При одночасному зростанні коефіцієнтів α_{cpu} та α_{ram} система переходить у деградаційний режим, у якому виконуються лише критичні задачі, тоді як фонові процеси тимчасово призупиняються.

6. Політика відновлення роботи системи. Повернення до штатного режиму здійснюється поступово із застосуванням механізму гістерезису. Обмеження знімаються лише після стабільного зниження коефіцієнтів протягом визначеного інтервалу часу.

7. Політика підвищеного моніторингу. Якщо співвідношення коефіцієнтів не відповідає жодному з типових режимів, активується режим додаткового моніторингу для подальшого аналізу поведінки системи.

Схему методу динамічного розподілу задач і ресурсів на основі тензорної декомпозиції зображено на рис. 2.5.

Метод включає формування тензорного подання параметрів задач і ресурсів, виконання тензорної декомпозиції, побудову факторних ознак, оцінювання подібності між задачами, їх групування у кластери та оцінювання стабільності груп. Отримані узагальнені показники використовуються планувальником для корекції розподілу задач і ресурсів.



Рисунок 2.5 - Схема методу динамічного розподілу задач і ресурсів

Для зручності реалізації та відтворення процесу інтелектуальної кластеризації метод подано у вигляді узагальненої таблиці (табл. 2.3), яка демонструє взаємозв'язок між етапами обробки даних, входними параметрами, операціями аналізу та результатами, що передаються планувальнику.

У табл. 2.3 послідовно відображено формування тензора на основі параметрів задач і ресурсів, виконання тензорної декомпозиції, побудову факторних векторів ознак, обчислення мір подібності між задачами, групування задач у кластери, оцінку стабільності кластерів і похибок, а також адаптивне оновлення структури кластерів у циклі планування з формуванням узагальнених показників, що використовуються планувальником.

Таблиця 2.3 – Взаємозв’язок етапів алгоритму тензорної кластеризації задач

Етап	Вхідні дані	Основна операція	Результат
1	Параметри задач і ресурсів, зібрані агрегуючим модулем	Формування тензора X	Багатовимірний масив спостережень
2	Тензор X	Виконання тензорного розкладу (CP-декомпозиції, див. формулу (2.10))	Факторні матриці, що відображають латентні зв’язки між задачами, ресурсами та часом
3	Факторні матриці	Побудова векторів ознак для кожної задачі	Простір факторних характеристик
4	Вектори ознак задач	Обчислення міри подібності $\text{sim}(t_i, t_j)$ за формулою (2.15)	Матриця подібностей між задачами
5	Матриця подібностей	Групування задач у кластери G_k	Набір кластерів задач
6	Кластери G_k і похибки e_i	Оцінка стабільності кластерів за формулою (2.16)	Показники R_k і маркування стабільності кластерів
7	Оновлені дані за цикл планування	Адаптивне оновлення кластерів при зміні навантаження	Оновлена структура кластерів для планувальника

Представлені взаємозв’язки етапів алгоритму тензорної кластеризації задач в табл. 2.3 дозволяють формалізувати послідовність обчислювальних етапів і спростити

реалізацію алгоритму у вигляді окремих функціональних модулів. Така форма подання забезпечує чітке розділення етапів збору даних, аналітичної обробки та використання результатів у контурі планування, що є важливим для систем реального часу, у яких обчислювально складні процедури повинні бути винесені за межі критичного контуру виконання задач. Завдяки модульній структурі метод може бути інтегрований у різні архітектури ОСРЧ без зміни принципів їх роботи, а також масштабований залежно від кількості задач, ресурсів і інтенсивності навантаження. Зокрема, окремі етапи аналізу можуть виконуватися з різною періодичністю або на окремих обчислювальних вузлах, що дозволяє адаптувати метод до вбудованих систем з обмеженими ресурсами або до розподілених КФС. Крім того, структуризація алгоритму у вигляді таблиці полегшує формалізацію потоків даних між модулями системи та спрощує програмну реалізацію, оскільки результати кожного етапу мають визначений формат і можуть безпосередньо використовуватися на наступному етапі обробки. Це також створює передумови для подальшого вдосконалення методу, зокрема шляхом розширення набору ознак, зміни моделей прогнозування або використання альтернативних алгоритмів кластеризації без зміни загальної структури методу та принципів його функціонування.

2.4 Висновки до другого розділу

У другому розділі сформовано архітектуру механізму самоадаптації ОСРЧ, призначеної для прогнозування навантаження та оптимального розподілу ресурсів у динамічних умовах функціонування. Запропоноване рішення реалізує одну з основних цілей дослідження – побудову архітектури, здатної забезпечувати ефективне використання системних ресурсів шляхом проактивного керування на основі прогнозування навантаження. Основою удосконалення архітектури є впровадження зовнішнього аналітичного модуля на основі тензорної декомпозиції, який виконує багатовимірний аналіз параметрів системи, тобто часу виконання задач, споживання обчислювальних ресурсів і пам'яті, пріоритетів та взаємозалежності між

процесами. Отримані в результаті аналізу латентні ознаки та агреговані метрики дозволяють передбачати зміну станів системи, виявляти критичні тенденції та формувати прогностні параметри для планувальника задач.

Запропонована архітектура планувальника задач із залученням зовнішнього модуля машинного навчання забезпечує реалізацію замкненого контуру самоадаптації, який включає етапи моніторингу стану ОСРЧ, аналітичної оцінки багатовимірних даних і періодичного оновлення параметрів планування та розподілу ресурсів. При цьому обчислювально інтенсивні процедури збору, нормування та аналізу даних виконуються поза межами ядра ОСРЧ, що дозволяє зберігати детермінізм його роботи та гарантії реального часу. Також в системах є можливість перенесення зовнішнього аналітичного модуля всередину КФС, якщо система із достатнім обчислювальним ресурсом. Поєднання тензорного підходу з методами машинного навчання забезпечує формування прогнозів ресурсних потреб, класифікацію задач і адаптивну зміну їхніх пріоритетів. Удосконалена архітектура також включає агрегуючий модуль, модуль моніторингу цілісності даних на основі комбінованого фільтра Калмана та ПЗСК, який використовує марковські моделі для оцінювання ймовірностей переходів між станами системи. Узгоджена робота цих компонентів підвищує ефективність, надійність і безперервність роботи ОСРЧ.

Розроблений метод динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції застосовано для вирішення задач адаптивного планування в операційних системах реального часу.

Таким чином, сформовано завершену концепцію модифікованої архітектури ОСРЧ, яка реалізує механізм самоадаптації на основі прогнозованого навантаження, забезпечує ефективний розподіл ресурсів і створює основу для подальшої реалізації методів відмовостійкості та захисту, розглянутих у наступних розділах дисертації.

Основні результати розділу опубліковані у працях [5; 6; 136]

РОЗДІЛ 3

МЕТОД ПРЕВЕНТИВНОГО ВІДНОВЛЕННЯ КОМПОНЕНТІВ ОПЕРАЦІЙНИХ СИСТЕМ РЕАЛЬНОГО ЧАСУ ТА МЕТОД ВИЯВЛЕННЯ ФАЛЬСИФІКАЦІЙ ТА АНОМАЛІЙ У КОМП'ЮТЕРНИХ СИСТЕМАХ ДЛЯ СИГНАЛІВ КІБЕРФІЗИЧНИХ СИСТЕМ

3.1 Організація превентивного відновлення та алгоритм роботи задачі прогнозного сторожового контролю в ОСРЧ

У сучасних ОСРЧ і вбудованих системах спостерігається зростання кількості задач і складності їх взаємодії, що ускладнює підтримання безперервної роботи та дотримання жорстких часових обмежень. Це особливо важливо для КФС, у яких помилки або затримки під час керування фізичними процесами можуть призводити до серйозних наслідків, зокрема відмови обладнання, аварій або недопустимих затримок у виконанні функцій безпеки.

Одним із ключових механізмів для підтримки надійності та захищеності є сторожовий таймер. Класичний сторожовий таймер перезапускає мікроконтролер у разі зависання або збою в обчислювальному ядрі, однак це реактивний підхід, коли збій уже настав, відбувається повний перезапуск, що може бути надто затратним за часом і ресурсами. Не враховується фактор навмисних впливів, наприклад, ін'єкцій збоїв чи логічних атак. Водночас використання подвійного апаратно-програмного сторожового таймера з можливістю локального програмного перезапуску окремих компонентів системи до настання фатального збою на основі низькорозмірних ланцюгів Маркова міг би дозволити суттєво підвищити відмовостійкість і скоротити загальний час простою, знизити ефективність атак типу відмова в обслуговуванні (DoS) або логічного блокування критичних задач.

Додатковим викликом є обмежені апаратні ресурси, такі як оперативна пам'ять, тактова частота, та необхідність дотримання реального часу. Для проведення глибокого аналізу стану системи часто застосовують складні методи, наприклад,

машинне навчання чи великі формальні моделі, які неможливо ефективно реалізувати на малопотужних мікроконтролерах. Натомість спрощені марковські ланцюги з малою кількістю станів дають змогу швидко та з мінімальними обчисленнями оцінювати ймовірність перебування системи у критичному стані. Завдяки такому підходу предиктивний шар може виявляти загрозу збою чи підозрілу активність ще до фактичного зависання і виконувати превентивний перезапуск лише проблемної задачі, драйвера або модуля.

З урахуванням зазначених обмежень і вимог розроблено метод превентивного перезапуску компонентів вбудованої системи реального часу на основі спрощеної марковської моделі ризику збою, інтегрованої з класичним апаратним сторожовим таймером, що забезпечує зменшення кількості повних перезапусків, скорочення часу простою та підвищення надійності і кіберстійкості КФС з обмеженими ресурсами.

Аналіз існуючих підходів до забезпечення відмовостійкості вбудованих систем реального часу свідчить, що АСТ хоча й гарантує повний перезапуск у разі зависання, проте реагує запізно, коли збій уже настав, і зумовлює втрату всього поточного контексту. Програмний сторожовий таймер може розпізнати відмову чи ознаки атаки конкретної задачі раніше, однак і він повністю залежний від працездатності ядра ОСРЧ. Методи резервування залишаються надто дорогими у сенсі ресурсів та складності для більшості обмежених за ресурсами вбудованих платформ, а застосування формальних методів та ML-підходів часто виходить за межі доступних обчислювальних можливостей або вимагає надмірного часу/зусиль на верифікацію.

Отже, у вбудованих системах із обмеженими ресурсами, як простих, так і середньої складності, доцільно поєднувати мінімальний апаратний механізм аварійного перезапуску сторожового таймеру із інтелектуальним програмним рівнем, який здатний превентивно реагувати на ознаки майбутнього збою та здійснювати локальне відновлення, зменшуючи потребу в повному перезапуску системи. При цьому ключовим викликом стає пошук легкого з точки зору обчислень і пам'яті способу оцінювання стану системи щодо ризику близького зависання, зростання часу

виконання задач чи переповнення черг. Надмірно прості пороги, наприклад, коли черга $< 80\%$, можуть викликати помилкові перезапуски, а складні моделі, такі як ML чи великий формальний аналіз, часто перевищують можливості мікроконтролерів.

Таким чином, актуальним рішенням є розробка методу малоресурсного прогнозування збою, який би дав змогу швидко обчислити ймовірність критичного стану, інтегруючись в програмний сторожовий таймер для локального перезапуску лише проблемного компонента, забезпечував безперебійну роботу системи в більшості нефатальних ситуацій і у випадку глибокого зависання передавав би керування апаратному сторожовому таймеру.

З урахуванням цих вимог можна сформулювати і гібридні підходи, які часто застосовуються для підвищення відмовостійкості чи розпізнавання атак. Низькорозмірний ланцюг Маркова є привабливим рішенням, оскільки дає змогу виконувати онлайн-оцінювання ризику без значного обчислювального навантаження. У межах цього підходу реалізовано подвійний механізм відмовостійкості і кіберстійкості, тобто поєднання апаратного сторожового таймера з програмним марковським рівнем аналізу в типовому середовищі ОСРЧ, що дозволяє запобігати повним перезапускам та скорочувати час простою системи.

У сучасних ОСРЧ ефективне управління ресурсами та балансування навантаження є важливими чинниками забезпечення надійної та стабільної роботи систем. Із розвитком КФС та IoT зростають обсяги телеметричних даних і кількість взаємопов'язаних процесів, що ускладнює своєчасне виявлення перевантажень, деградації продуктивності та передумов збоїв у режимі реального часу. Це обумовлює необхідність застосування методів оперативного моніторингу й прогнозування стану системи, здатних працювати за умов обмежених обчислювальних ресурсів.

У більшості ОСРЧ базова структура включає ядро з планувальником задач, системні сервіси, семафори, черги, таймери, драйвери пристроїв і набір прикладних задач. Для відстеження збоїв часто використовують АСТ, який періодично оновлюється з одного або кількох місць у програмі. Якщо передача керуючого

сигналу не виконується вчасно, АСТ ініціює глобальний перезапуск мікроконтролера, забезпечуючи відновлення системи після потенційного зависання, атаки. Базова архітектура ОСРЧ з апаратним сторожовим таймером містить такі задачі, компоненти та елементи:

1) планувальник визначає задачі та отримуваний ними процесорний час і при цьому має переважно пріоритетний витісняючий тип планування або можливий як варіант EDF;

2) задача у типовій системі виконує певні функції, такі як обслуговування сенсорів, обмін даними, керування виконавчими механізмами і вого працює періодично або на основі подій;

3) системні сервіси (семафори, черги, таймери) забезпечують синхронізацію й комунікацію між задачами, а також можуть бути додаткові служби логування або налаштувань;

4) драйвери пристроїв забезпечують рівень взаємодії із зовнішньою апаратурою (UART, SPI, I^2C , ADC тощо) та здебільшого надають API для задач;

5) АСТ є окремим апаратним блоком у мікроконтролері, який необхідно періодично оновлювати, і якщо протягом заданого часу оновлення не виконується, то таймер ініціює повний перезапуск системи.

У базовій архітектурі АСТ виконує роль останнього рівня захисту від зависання.

У цій стандартній схемі розробник може періодично викликати функцію оновлення апаратного сторожового таймера у головному циклі чи в одному з критичних задач. Якщо все справно, то жодного скидання не відбувається. Але у випадку фатальної помилки, наприклад, зависла шина, зупинився планувальник, система просто перевантажується, втрачаючи поточний контекст. Графову модель зображено на рис. 3.1 а) (зліва).

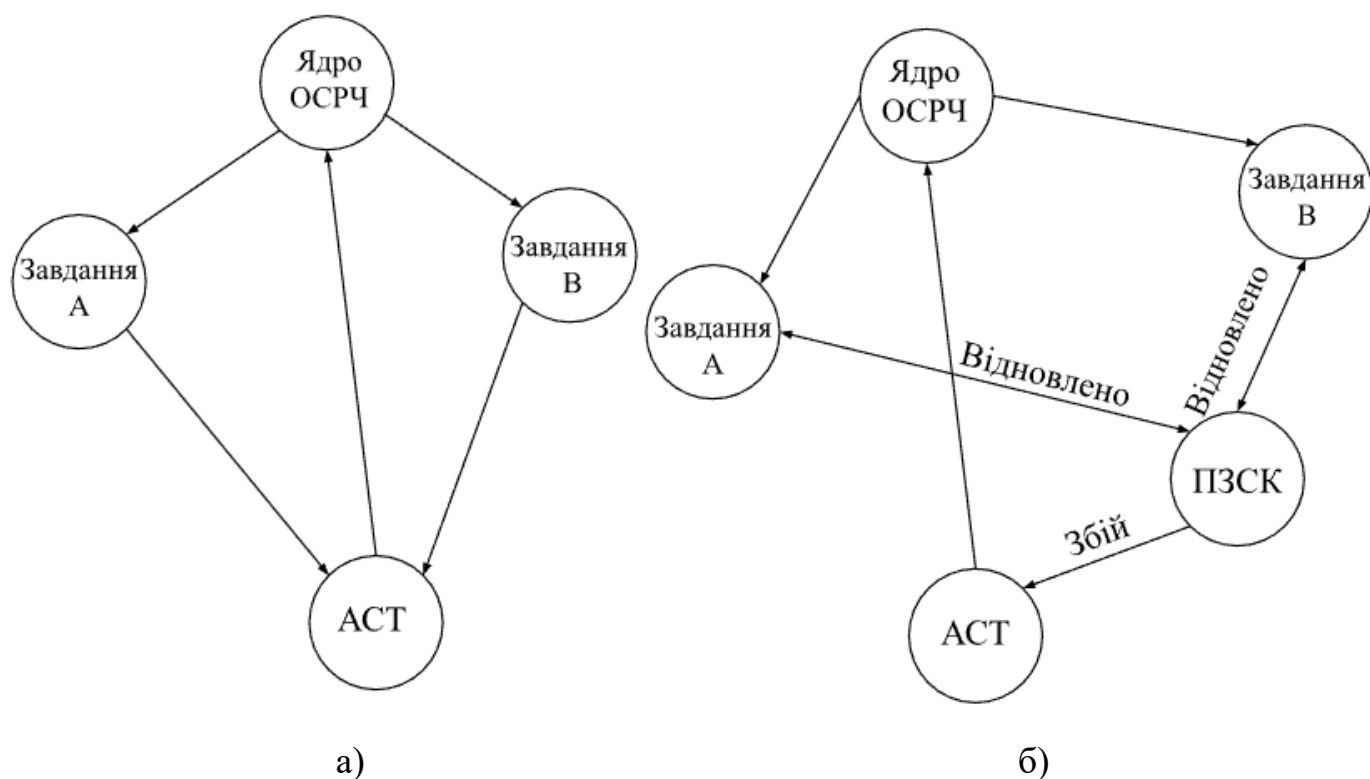


Рисунок 3.1 - Графова модель ОСРЧ із простим сторожовим таймером і модель ОСРЧ з інтегрованим превентивним шаром на основі спрощеної марковської моделі.

Модифікована архітектура з інтегрованим превентивним шаром (на рис. 3.1 б) (справа)):

- 1) планувальник задач, який, без змін у базовій логіці, здійснює пріоритетне планування виконання задач;
- 2) користувацькі задачі, що, як і в типовій архітектурі, виконують власні функціональні обов'язки;
- 3) системні сервіси та драйвери пристроїв, робота яких не зазнає суттєвих змін з боку ядра ОСРЧ;
- 4) АСТ, який виконує функцію апаратного механізму останнього рівня захисту;
- 5) додано спеціальну задачу прогнозного сторожового контролю (ПЗСК).

Компонент ПЗСК працює на рівні користувацького простору у складі ОСРЧ і має такі особливості:

1) збирає сигнали про активність від інших задач, наприклад, через черги або глобальні змінні;

2) використовує спрощену марковську модель для оцінювання ймовірності переходу системи у критичний стан, в якій в ролі вхідних даних можуть виступати показники завантаженості черг, час обробки подій, історія сигналів від задач;

3) якщо ризик збою за марковською моделлю перевищує заданий поріг, то ПЗСК ініціює локальне програмне відновлення підозрілого компонента, що включає завершення відповідної задачі та її повторний запуск, або повторну ініціалізацію пов'язаної підсистеми;

4) якщо система працює стабільно, ПЗСК регулярно передає керуючий сигнал апаратному сторожовому таймеру з фіксованим інтервалом, щоб підтвердити коректну роботу системи та запобігти автоматичному перезавантаженню, а якщо виявлено, що локальні спроби відновлення не дали результату, то можна навмисно не передавати керуючий сигнал апаратному сторожовому таймеру, що призведе до його спрацювання та виконання повного перезавантаження системи.

Логіка взаємодії ПЗСК з іншими компонентами:

- кожна робоча задача періодично передає модулю ПЗСК сигнал про власну активність, який використовується для оцінювання поточного стану задачі;

- ПЗСК виконується з високим пріоритетом кожні 100–200 мс і перевіряє оновлення від задач та обчислює поточний стан марковської моделі (Normal, Warning, Critical, Fail);

- якщо модель вказує на високу ймовірність зависання, то виконується програмний перезавантаження проблемного модуля і якщо після кількох таких спроб система залишається у критичному стані, то задача свідомо не передає керуючий сигнал апаратному сторожовому таймеру, що призводить до його спрацювання та ініціює глобальний перезавантаження системи для повного відновлення;

- якщо система перебуває в нормальному стані, то ПЗСК передає керуючий сигнал апаратному сторожовому таймеру, підтверджуючи стабільну роботу системи.

Запропонована архітектура мінімізує вплив збоїв, дає можливість швидкого часткового відновлення й одночасно не вимагає кардинальних змін у ядрі ОСРЧ, оскільки весь інтелектуальний механізм реалізується на рівні одної додаткової задачі (ПЗСК) та легкого драйвера спрощеної марковської моделі. Такий підхід пропонує практичний компроміс між простотою й адаптивністю, зберігаючи класичний АСТ як остаточний метод захисту.

Для забезпечення детермінованого прийняття рішення про виконання превентивного перезапуску компонентів системи у межах ПЗСК розроблено формальний критерій, який дозволяє оцінювати не лише поточний рівень ризику, але й темп його зростання. Такий підхід забезпечує своєчасну реакцію системи на прогностичні ознаки деградації, не очікуючи фактичного переходу у стан Fail. Інтегральний показник ризику J_k визначимо як зважену суму поточних і динамічних характеристик ймовірностей переходів між станами так:

$$J_k = \lambda_1 P_{CF}(k) + \lambda_2 \frac{dP_{CF}(k)}{dk} + \lambda_3 R_k, \quad (3.1)$$

де $P_{CF}(k)$ – поточна оцінена ймовірність переходу зі стану Critical у стан Fail за даними марковської моделі; $\frac{dP_{CF}(k)}{dk}$ – швидкість зміни цієї ймовірності, що відображає темп наближення системи до критичного стану; R_k – інтегрований індикатор ризику, який може враховувати результати додаткових підсистем моніторингу, зокрема IDS та РНМ; $\lambda_1 \lambda_2 \lambda_3$ – вагові коефіцієнти, що задають пріоритетність окремих складових при оцінюванні загального рівня небезпеки.

Для реалізації адаптивного механізму прийняття рішення визначається порогове значення θ , що задає межу між нормальним і небезпечним рівнем ризику. Якщо виконується умова $J_k > \theta$, то модуль ПЗСК ініціює превентивне програмне перезавантаження окремої задачі, підсистеми або драйвера. У випадку, коли після локального скидання показник ризику продовжує зростати, система передає керування апаратному сторожовому таймеру для повного перезапуску. Такий підхід забезпечує двоступеневу модель реагування. Спочатку активується програмний

рівень відновлення, який дозволяє швидко усунути короточасні або локальні збої, а при відсутності ефекту – апаратний. Введення інтегрального критерію J_k забезпечує формалізований зв'язок між результатами прогнозу та діями відновлення, що підвищує узгодженість поведінки системи й зменшує кількість хибних перезапусків. Завдяки такій формалізації алгоритм ПЗСК набуває властивостей адаптивного керування ризиком, бо він враховує не лише абсолютне значення ймовірності збою, а й її зміну в часі, що дозволяє ініціювати відновлення ще до фактичного переходу у відмовний стан. Це підвищує загальну відмовостійкість системи без суттєвого зростання обчислювального навантаження та забезпечує повну автономність прийняття рішень у реальному часі.

Водночас оцінювання ризику збою лише за внутрішніми параметрами роботи задач не завжди є достатнім, оскільки причини деградації можуть мати програмну, апаратну або вплив ЗПЗ. Тому доцільним є розширення інформаційної бази прийняття рішень за рахунок додаткових підсистем моніторингу, зокрема IDS і PHM.

Сам алгоритм роботи ПЗСК на основі ланцюга Маркова наведено на рис. 3.2 у вигляді блок-схеми, що відображає послідовність дій від моменту ініціалізації до вибору стратегії відновлення. Він ілюструє логіку роботи задачі моніторингу у ОСРЧ:

- 1) запуск моніторингу системи;
- 2) аналіз процесу та входних сигналів моніторингу (внутрішніх параметрів системи, а також, за наявності, даних від IDS і PHM);
- 3) перевірка за марковською моделлю, чи має задача передумови для зависання;
- 4) якщо так, то виконати перезапуск (відновлення) задачі;
- 5) якщо після перезапуску проблему вирішено, то завершити роботу таймера нагляду;
- 6) якщо після перезапуску проблему не вирішено, то перезапустити систему;
- 7) після перезапуску системи знову починається цикл моніторингу.

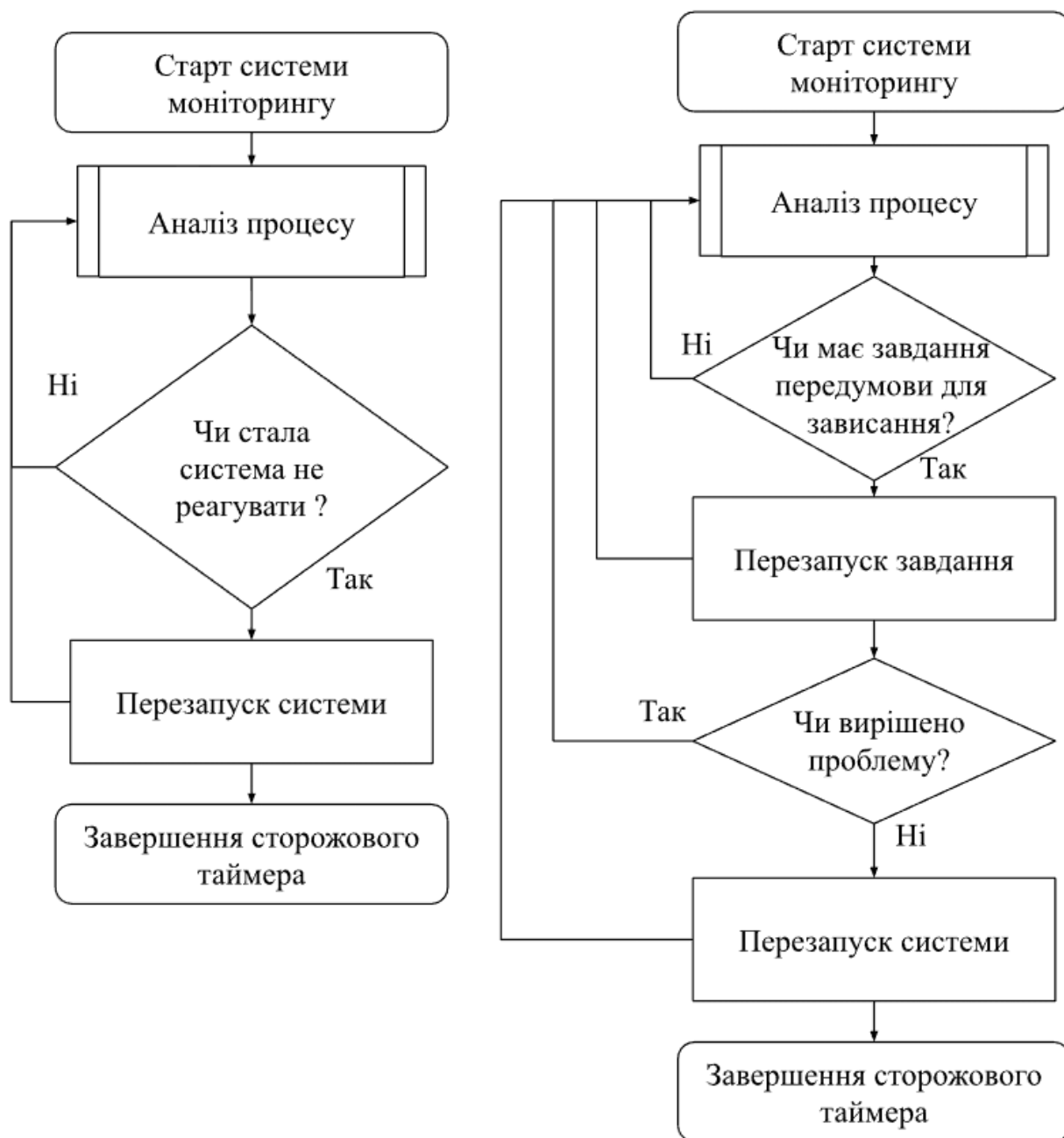


Рисунок 3.2 - Порівняльна схема роботи стандартного й модифікованого сторожового таймера

Таким чином у підсумку математична схема має такі кроки:

- 1) стан системи $x(k)$, що є дискретною змінною у скінченному просторі S ;
- 2) закони переходів визначаються матрицею P , яку оновлюємо інкрементально чи у ковзному вікні згідно формули (3.7);

3) ймовірність збою за один крок, тобто це p_{iF} (з “Critical” у “Fail” чи “Warning” у “Fail”), за потреби з урахуванням сигналів IDS і PHM, а при перевищенні порогового значення θ запускаємо превентивні дії (програмне скидання);

4) за необхідності розглядаємо прогноз на r кроків через P^r ;

5) програмне скидання еквівалентне примусовому поверненню системи з небезпечного стану в “Normal” (або “Recovery”);

6) повне скидання через АСТ відбувається, якщо система сповіщає про критичну проблему, але програмне скидання не дає результату.

У межах запропонованого підходу система виявлення вторгнень (IDS) розглядається як програмний компонент ПЗСК, відповідальний за виявлення аномалій у поведінці задач і цифрових сигналів, тоді як платформа прогнозування і технічного обслуговування за станом (PHM) відповідає за оцінювання технічного стану та деградації апаратних компонентів. Запропонований модуль ПЗСК може бути розширений за рахунок використання зовнішніх сигналів від IDS та PHM. Така інтеграція дозволяє поглибити оцінювання кіберстійкості системи, оскільки об’єднує інформацію як про технічний стан вузлів, так і про можливі кібераномалії. Система IDS формує оцінки аномалій на основі поведінкових патернів мережної або внутрішньої активності, які можуть вказувати на зловмисне втручання, перевищення дозвolenної частоти запитів, зміну шаблонів обміну повідомленнями між задачами чи атипові звернення до периферійних інтерфейсів. У той же час, підсистема PHM генерує оцінки технічного ризику, що базуються на телеметрії з сенсорів та драйверів: температура мікроконтролера, рівень споживання струму, частота помилок у каналах зв’язку, стабільність напруги живлення тощо. Ці сигнали інтегруються з марковським оцінювачем ризику збою, який використовується у ПЗСК для прийняття рішення про превентивне відновлення компонентів. Формально інтегрований індикатор ризику можна записати так:

$$R_k = \alpha P_k (C \rightarrow F) + \beta A_k^{IDS} + \gamma H_k^{PHM}, \quad (3.2)$$

де k - дискретний такт (період опитування ПЗСК); $P_k (C \rightarrow F)$ – оцінена ймовірність переходу системи зі стану Critical у Fail за марковською моделлю; $A_k^{IDS} \in [0,1]$ – нормалізований показник аномальності від IDS; H_k^{PHM} – нормалізований показник технічного ризику від PHM; α, β, γ – вагові коефіцієнти з одиничною сумою ($\alpha + \beta + \gamma = 1$), які визначають відносний внесок кожного джерела інформації.

Інтегрований показник R_k оновлюється під час кожного циклу моніторингу. Якщо він перевищує порогове значення δ , то ініціюється локальне програмне відновлення відповідного компонента або задачі, тоді як АСТ залишається останнім рівнем захисту. З практичного погляду така інтеграція реалізується без зміни ядра ОСРЧ, тобто ПЗСК може підписуватися на черги або повідомлення, що надходять від модулів IDS та PHM, періодично нормалізуючи їхні значення до діапазону $[0,1]$. Таким чином, рішення приймається на основі об'єднаної оцінки ризику, яка враховує як технічні, так і поведінкові ознаки потенційного збою або атаки. Об'єднання сигналів IDS і PHM у межах ПЗСК створює єдиний інформаційний контур, який дозволяє наступне:

- 1) раніше виявляти як технічні, так і кібернетичні збої;
- 2) зменшити кількість помилкових перезапусків завдяки кращій диференціації типів відмов;
- 3) забезпечити проактивне реагування на тенденції деградації системи ще до настання критичного стану;
- 4) реалізувати уніфіковану метрику кіберстійкості системи у реальному часі.

У результаті інтеграція ПЗСК з IDS і PHM формує додатковий рівень кіберзахисту, не потребуючи модифікацій ядра ОСРЧ або машини станів. Вона поєднує переваги програмного моніторингу, статистичного оцінювання ймовірностей та апаратного сторожового таймера, створюючи комплексний підхід до самовідновлення систем із жорсткими часовими обмеженнями. Подальше вдосконалення моделі інтеграції полягає у введенні адаптивного механізму вагових

коефіцієнтів α, β, γ що дозволяє автоматично змінювати внесок кожного джерела інформації залежно від поточного режиму роботи системи. Наприклад, у стабільному стані (Normal) більшу вагу має оцінка PHM, оскільки домінують фізичні деградаційні процеси (зміна температури, напруга живлення, дрейф сенсорів). Натомість у режимі підозрілої активності (Warning або Critical) пріоритетним стає сигнал IDS, що забезпечує підвищену чутливість до мережевих або логічних аномалій.

Функціонально це реалізується через динамічну нормалізацію коефіцієнтів:

$$\alpha_k = f_1(S_k), \beta_k = f_2(S_k), \gamma_k = f_3(S_k), \quad (3.3)$$

де S_k – поточний стан системи за марковською моделлю; функції f_1, f_2, f_3 задаються аналітично або емпірично (наприклад, якщо система переходить зі стану W (Warning) у C (Critical), то коефіцієнт β_k зростає, тоді як γ_k зменшується).

Це дозволяє балансувати між програмним (IDS) і технічним (PHM) рівнем контролю, підтримуючи релевантність оцінки ризику в реальному часі.

Отже, запропоновано підхід до превентивного відновлення в ОСРЧ на основі поєднання апаратного сторожового таймера та задачі прогнозного сторожового контролю, що використовує спрощену марковську модель для оцінювання ризику збою. Введено критерій прийняття рішення про локальне відновлення компонентів і показано можливість інтеграції сигналів IDS і PHM для підвищення точності оцінювання стану системи. Реалізація запропонованої архітектури забезпечує двоступеневий механізм реагування, зменшує кількість повних перезапусків і може бути здійснена без модифікації ядра ОСРЧ.

3.2 Метод превентивного відновлення компонентів в ОСРЧ на основі низькорозмірної марковської моделі

Метод полягає у поєднанні класичного апаратного сторожового таймера як жорсткого рівня захисту та прогнозного програмного шару (ПЗСК), який

використовує спрощену марковську модель для виявлення ймовірного збою ще до його настання й виконує локальне відновлення проблемних компонентів. У разі, якщо програмний перезапуск не допомагає або зависання охоплює ядро загалом, то АСТ робить повний перезапуск системи. Такий двоетапний механізм сприяє зменшенню кількості повних перезапусків системи та скороченню часу простою, що особливо важливо в КФС із жорсткими часовими вимогами та обмеженими ресурсами.

Метод передбачає дискретну оцінку стану системи в моменти часу $t = k \cdot \Delta t = k$, де Δt – період виконання ПЗСК у ОСРЧ. На кожному кроці система визначається в одному з кінцевого набору станів (наприклад, $\{N, W, C, F\}$ – “Normal”, “Warning”, “Critical”, “Fail”), і переходи між цими станами описуються ланцюгом Маркова малої розмірності.

Нехай $S = \{N, W, C, F\}$ – скінчена множина станів. У кожен такт $t^k = k\Delta t$ система (або ПЗСК, який відображає її рівень працездатності) перебуває в одному зі станів $x(k) \in S$. Для спрощення припустимо, що:

- N (Normal) – система в нормальному діапазоні (черги не переповнені, сигнали про активність усіх задач приходять вчасно);
- W (Warning) – виявлено погіршення показників (підвищене завантаження, повільні сигнали про активність), але ще не критично;
- C (Critical) – ризик швидкого збою високий (суттєві затримки, часті пропуски дедлайнів, задача припиняє надсилати сигнали про активність);
- F (Fail) – фактичне зависання / збій (у моделі – це поглинаючий стан або індикатор, що відбувся глобальний reset).

У реалізації можна мати більше/менше станів, наприклад, “Recovery”, “Standby”, проте 3–5 активних станів уже достатньо для більшості спрощених сценаріїв.

Вважаємо, що $x(k + 1)$ залежить лише від $x(k)$ і не залежить від станів у попередні моменти (властивість Маркова). Можна визначити так:

$$P = \begin{pmatrix} P_{NN} & P_{NW} & P_{NC} & P_{NF} \\ P_{WN} & P_{WW} & P_{WC} & P_{WF} \\ P_{CN} & P_{CW} & P_{CC} & P_{CF} \\ P_{FN} & P_{FW} & P_{FC} & P_{FF} \end{pmatrix}, \sum_{j \in S} P_{ij} = 1, \forall i \in S, \quad (3.4)$$

де P – матриця ймовірностей переходів між станами системи (матриця переходів Маркова); P_{ij} - ймовірність переходу системи зі стану i у стан j за один дискретний крок (наприклад, за один такт або цикл); $i, j \in S$ – індекси станів системи, де множина $S = \{N, W, C, F\}$ тобто це всі можливі стани; $\sum_{j \in S} P_{ij} = 1$ – умова нормування, тобто сума ймовірностей усіх можливих переходів зі стану i дорівнює 1 (система обов'язково переходить у якийсь стан).

Якщо F (Fail) розглядається як поглинаючий стан, то виконується $P_{FF} = 1$ і $P_{Fj} = 0$ для $j \neq F$. У такому разі потрапляння в F інтерпретує безповоротне зависання. Якщо ж у фізичній системі після F (Fail) виконується апаратний reset, то логічно вважати, що стан системи повертається до N (Normal) – але вже після перезапуску.

Для оцінки елементів p_{ij} застосуємо простий метод обчислення частот. Припустимо, що на k -му такті систему віднесено до стану $i = x(k)$, а на $x(k + 1)$ - му такті – до стану $j = x(k + 1)$. Тоді лічильник $n_{(i \rightarrow j)}$ збільшуємо на 1 і задамо це співвідношення так:

$$n_{(i \rightarrow j)} := n_{(i \rightarrow j)} + 1, \quad (3.5)$$

де, $n_{(i \rightarrow j)}$ – кількість разів, коли система переходила зі стану i у стан j .

Це лічильник переходів, який збільшується на 1 кожного разу, коли спостерігається такий перехід.

Додатково будемо відслідковувати кількість разів, коли система перебувала у стані i , і задамо це співвідношення так:

$$n_i = \sum_{m \in S} n_{(i \rightarrow m)}, \quad (3.6)$$

де n_i – загальна кількість випадків, коли система перебувала у стані i , тобто це фактично кількість усіх переходів, що починались зі стану i , незалежно від того, у який стан система потім перейшла; $\sum$ – додавання усіх переходів, що виходять зі стану i у даному контексті; m – Індекс, який перебирає всі можливі стани системи (член множини S); S – множина всіх станів системи $S = \{N, W, C, F\}$.

Таким чином, отримуємо таке співвідношення:

$$p_{ij} \approx \frac{n_{(i \rightarrow j)}}{\sum_{m \in S} n_{(i \rightarrow m)}} = \frac{n_{(i \rightarrow j)}}{n_i}, \quad (3.7)$$

де p_{ij} - емпірична ймовірність переходу системи зі стану i у стан j за один дискретний крок; $n_{(i \rightarrow j)}$ – кількість разів, коли система переходила зі стану i у стан j ; n_i – загальна кількість разів, коли система перебувала у стані i .

Це співвідношення дає апроксимацію ймовірності переходу $i \rightarrow j$, що дає змогу оновлювати P без великого додаткового навантаження на процесор. Якщо система тривалий час працює, то можна застосовувати скінчене ковзне вікно [128; 129]. У межах підходу ковзного вікна розміром L зберігаються лише останні спостереження переходів між станами, тоді як найстаріші дані поступово вилучаються. У такому випадку матриця переходів $P(k)$ формується не на основі всієї історії спостережень, а лише за межами поточного ковзного інтервалу, що відображає актуальний стан системи в реальному часі. Це дозволяє усунути вплив застарілих статистичних даних, які втратили релевантність через зміну режиму роботи, динаміки навантаження чи зовнішніх умов. Формально емпіричну ймовірність переходу між станами у межах ковзного вікна задамо так:

$$p_{ij}(k) = \frac{\sum_{t=k-L+1}^k \delta_{(i \rightarrow j)}(t)}{\sum_{m \in S} \sum_{t=k-L+1}^k \delta_{(i \rightarrow m)}(t)}, \quad (3.8)$$

де $\delta_{(i \rightarrow j)}(t)$ – індикатор переходу системи зі стану i у стан j на такті t (дорівнює 1, якщо перехід зафіксовано, і 0 – інакше); L – довжина ковзного вікна, тобто кількість

останніх кроків спостереження, що враховуються під час оцінювання; S – множина всіх можливих станів системи.

Зміщення вікна на кожному кроці часу k реалізується за принципом черги типу FIFO, коли найновіше спостереження додається, а найстаріше – видаляється з множини спостережень. Це забезпечує інкрементне оновлення елементів матриці $P(k)$ з мінімальними обчислювальними витратами, тобто порядку $O(1)$ операцій на цикл.

Такий підхід має дві ключові переваги. По-перше, він гарантує, що ймовірності переходів відображають поточну поведінку системи, а не усереднений історичний розподіл, що важливо в умовах змінного навантаження чи зовнішніх впливів. По-друге, він дає змогу зберегти низьке навантаження на процесор і пам'ять, що є принципово важливим для вбудованих ОСРЧ. Оптимальний розмір вікна L визначається залежно від динаміки змін у системі та періоду опитування ПЗСК. Її вибір здійснюється таким чином, щоб забезпечити баланс між чутливістю до короткочасних змін і стабільністю оцінок, не допускаючи накопичення застарілої статистики. У результаті метод ковзного вікна забезпечує адаптивність і стійкість моделі при збереженні мінімальних обчислювальних витрат.

На кожному кроці часу Δt ПЗСК в ОСРЧ збирає метрики: завантаження черг; затримки; пропущені встановлені терміни виконання. А потім за пороговою логікою відносить систему до одного зі станів:

- N (Normal), де наприклад, якщо заповненість черг перевищує нормалізований поріг 50% і всі сигнали про активність від критичних задач своєчасні;
- W (Warning), де спостерігається часткове або періодичне перевищення рівня навантаження, при цьому відсутні тривалі пропуски дедлайнів, а максимальна нормалізована заповненість черг перебуває у діапазоні 50–80 % або фіксується періодична затримка сигналів активності окремих задач;
- C (Critical), якщо нормалізоване переповнення черг перевищує 80 % чи систематичні пропуски дедлайнів однієї чи кількох важливих задач;

- F (Fail), якщо сигнали про активність відсутні для однієї чи кількох критичних задач упродовж двох-трьох циклів поспіль або якщо сталася фатальна помилка.

Таким чином, визначення N, W, C, F залежить від прикладного сценарію й встановлених порогових значень. Значення таких порогів встановлюють на основі статистичного аналізу робочих параметрів системи, експериментальних вимірювань або вимог до надійності конкретного застосунку, чи на основі аналізу параметрів ризику в ковзному вікні. Граф переходів між станами сторожового таймера зображено на рис. 3.3.

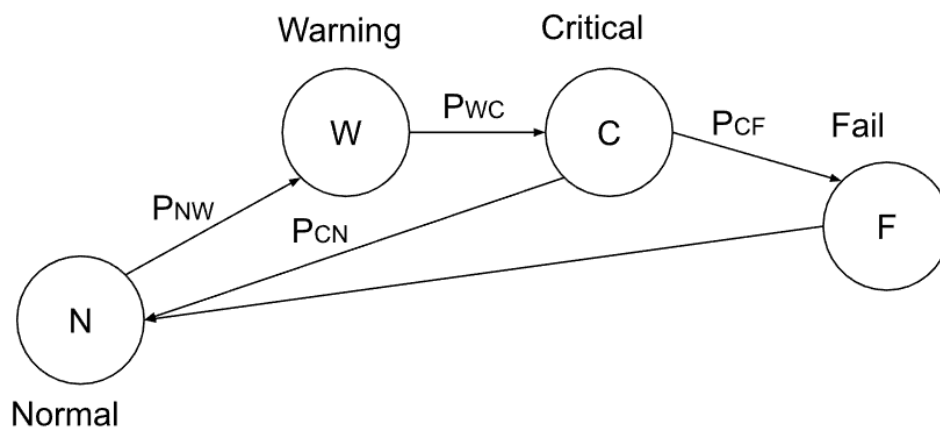


Рисунок 3.3 - Графова модель переходів станів прогнозуючого сторожового таймера

Ймовірності переходів між станами у ПЗСК є динамічними і можуть коригуватися в реальному часі залежно від змін у ключових системних метриках. Основними факторами впливу є заповненість черги, затримка сигналів активності задач, використання обчислювальних ресурсів і пам'яті. Наприклад, якщо рівень заповненості черги перевищує певний поріг, то це може свідчити про наближення критичного стану, що підвищує ймовірність переходу, яку можна бути подано так:

$$P(W \rightarrow C \mid Q, H, M) = f_1(Q, H, M), \quad (3.9)$$

де $P(W \rightarrow C \mid Q, H, M)$ – ймовірність переходу системи зі стану W(Warning) у стан C (Critical); f_1 – емпірична або аналітична функція залежності (регресія чи апроксимація), що описує, як ймовірність переходу змінюється відносно ключових

метрик; індекс “1” позначає, що це перша функція у наборі для переходу $W \rightarrow C$; Q – заповненість черги відображає рівень накопичення запитів або задач у системі; H – затримка сигналів активності, тобто час відгуку системи або середній інтервал між подіями моніторингу; M – використання пам’яті, тобто відсоток або абсолютна кількість зайнятої оперативної пам’яті.

Ймовірність переходу системи між станами зростає, коли система наближається до перевантаження або відмови. Якщо Q перевищує певний поріг, то це свідчить про зниження пропускної здатності. Збільшення H означає, що система працює повільніше або є ризик дедлайнових порушень. Зростання M може свідчити про витоки пам’яті чи неефективне керування буферами. Аналогічно, якщо затримка сигналів активності системи триває довше очікуваного часу, то зростає ймовірність переходу у стан відмови і задамо її так:

$$P(C \rightarrow F | H, T) = f_2(H, T), \quad (3.10)$$

де $P(C \rightarrow F)$ – ймовірність переходу системи зі стану C (Critical) у стан F (Fail), тобто стан повної відмови або зупинки; f_2 – функція залежності, друга у наборі функцій f_n , яка визначає зміну ймовірності переходу відносно поточних метрик системи; тобто у цьому випадку це залежність від затримки сигналів активності та часу; H – затримка сигналів активності, тобто час між очікуваним і фактичним сигналом активності і якщо H збільшується, то це вказує на зниження реактивності системи, ризик дедлайнових збоїв або втрату синхронізації; T – час перевищення очікуваного інтервалу, тобто проміжок, протягом якого система перевищує допустимий час реакції і якщо затримка триває довше T , то система ймовірно переходить у стан відмови, а якщо ж після локального скидання система швидко стабілізується, то збільшується. Тому, задамо цей перехід так:

$$P(C \rightarrow N | R) = f_3(R), \quad (3.11)$$

де $P(C \rightarrow N)$ – ймовірність переходу системи зі стану C (Critical) до стану N (Normal), тобто відновлення нормальної роботи після критичного стану, наприклад, після перезапуску або само-відновлення; f_3 - третя функція у наборі f_n , що описує залежність ймовірності переходу від певних параметрів системи і тут вона залежить від параметра R , що задає кількість успішних перезапусків або рівня відновлюваності системи; R - кількість успішних програмних перезапусків або метрика відновлення.

Чим більша кількість вдалих перезапусків без критичних наслідків, то тим вища ймовірність повернення системи у стан N .

Час до відмови моделюється випадковою змінною, яка підпорядковується експоненційному розподілу, оскільки відмови систем у реальному часі зазвичай є незалежними подіями, що відбуваються із сталою інтенсивністю. Експоненційний розподіл використовується для оцінки ймовірності того, що система залишиться в робочому стані протягом певного часу, тобто що відмова не відбудеться раніше заданого моменту. Основна його властивість полягає у відсутності пам'яті, що означає, що ймовірність відмови у майбутньому не залежить від того, скільки часу система вже працює без збоїв. Інтенсивність відмови задамо параметром λ , який визначає середню частоту відмов, а саму ймовірність збереження працездатності системи протягом часу t визначимо так:

$$P(T > t) = e^{-\lambda t}, \quad (3.12)$$

де T – час до відмови; λ – інтенсивність відмови; $P(T > t)$ - ймовірність того, що система залишиться працездатною (не відмовить) протягом часу t , тобто це функція надійності системи; $e^{-\lambda t}$ – вираз для експоненційного розподілу, що визначає ймовірність того, що система працюватиме без відмови до часу t .

Ймовірність переходу у стан W або C визначимо як:

$$P(N \rightarrow W) = 1 - P(T > t), \quad (3.13)$$

де $P(N \rightarrow W)$ – ймовірність переходу системи зі стану N (Normal) до стану W (Warning), що відображає перехід системи у попереджувальний режим; $1 - P(T > t)$ характеризує зростання ризику виникнення відмови протягом часу t і використовується як індикатор переходу системи у стан W .

Це є доповненням до функції надійності. Якщо розраховане значення ймовірності перевищує певний поріг, то сторожовий таймер переходить до проактивного реагування, викликаючи програмне скидання або готує апаратне скидання.

Розглянемо прогнозування збою та двоступеневий захист. Якщо відомо, що поточний стан $x(k) = i$, тоді $P(x(k+1) = Fail | x(k) = i) = p_{iF}$. Якщо ця ймовірність p_{iF} перевищує заздалегідь встановлений поріг θ , то обираємо θ , тобто порогове значення ймовірності збою. Коли виконується таке співвідношення:

$$p_{iF} > 0 \text{ (або ж } [P^2]_{iF} > 0, \text{ типово } \theta \in [0.5; 0.8]) \quad (3.14)$$

то система вважається близькою до критичного стану, і прогнозний шар ухвалює рішення про превентивне програмне скидання відповідного компонента, наприклад, видалити проблемну задачу та створити його наново, не чекаючи фактичного зависання. Тут p_{iF} – ймовірність переходу зі стану i у стан F (Fail) за один такт, p_{iF} – елемент матриці переходів Маркова, $[P^2]_{iF}$ – елемент матриці квадрату матриці переходів P^2 , який визначає ймовірність переходу з i у F за два кроки. Цей показник враховує опосередковані переходи, наприклад, $i \rightarrow C \rightarrow F$.

Якщо потрібно розглянути ймовірність потрапляння до F (Fail) за кілька кроків, то використовуємо добуток матриці P^r , який визначаємо так:

$$P^r = \underbrace{\{P \times P \times \dots \times P\}}_{r \text{ разів}} \quad (3.15)$$

де P – матриця ймовірностей переходів між станами системи за один дискретний такт; $P^1, P^2, P^3, \dots, P^r$ – послідовні степені матриці переходів, які показують, як змінюються

ймовірності переходів за кілька кроків; r - кількість кроків, дискретних моментів часу, через які аналізується стан системи.

Тоді елемент $[P^r]_{iF}$ встановлює ймовірність щодо появи у стані F через r кроків, якщо в поточний момент часу $x(k) = i$. У практичних мікроконтролерних системах зазвичай вистачає 1–3 кроків прогнозу через обмежені ресурси і невеликі періоди Δt .

У пропонованому двоетапному механізмі можуть бути такі варіанти:

- якщо аналіз марковського ланцюга вказує на ймовірність ($\geq \theta$) швидкого переходу у F , то програмний шар локально відновлює проблемну задачу через $R = \langle T_{delete}, D_{reinit}, T_{create} \rangle$, де T_{delete} – завершення задачі, D_{reinit} – повторна ініціалізація відповідної підсистеми, T_{create} – повторний запуск задачі;

- якщо локальний перезапуск не допомагає (стан C зберігається), то програмний шар не викликає оновлення апаратного сторожового таймера і за час T_{HW} (таймаут АСТ) відбудеться повний reset системи, тобто у марковській моделі це відповідає переходу $C \rightarrow F$, за яким фактично слідує перезавантаження мікроконтролера і повернення до початкового стану N .

Таким чином, якщо відхилення не надто глибоке, програмне скидання дає змогу уникнути глобального перезапуску і зменшити час простою. АСТ зберігається як останній рубіж, якщо система вийшла з-під контролю програмного шару. Формально таку ситуацію задамо так:

- якщо на кроці k запущено процедуру програмного скидання, то на кроці $k + 1$ штучно встановлюємо $x(k + 1) = N$ (якщо перезапуск відбувся успішно);

- задля точності вводимо стан “Recovery” (R) і модифікуєм матрицю переходів.

Оскільки маємо $\Delta t \ll T_{HW}$, тоді програмний шар має кілька спроб, наприклад, 10–20 циклів із кроком Δt , виконати програмне скидання, перш ніж спливе таймаут апаратного сторожового таймера. Якщо ПЗСК вважає, що стан знову став Normal або Warning без прогресу до Critical, то він викликає HW_WDT_Feed(), продовжуючи нормальну роботу. Якщо локальний збій не усувається, ПЗСК припиняє передавання

керуючого сигналу апаратному сторожовому таймеру. Це призводить до завершення інтервалу його очікування та спрацювання механізму апаратного перезавантаження. Формально, якщо стан системи визначено як Critical ($x(k) = C$) і після кількох спроб програмного відновлення показники ризику не зменшуються, програмне скидання вважається неефективним. У такому випадку керуючий сигнал АСТ не передається. Після завершення встановленого інтервалу його спрацювання відбувається повний перезавантаження системи.

У підсумку математична схема визначається наступними співвідношеннями. Стан системи $x(k)$ – дискретна змінна у скінченному просторі S . Закони переходів визначаються матрицею P , яку оновлюємо інкрементально чи у ковзному вікні, так:

$$p_{ij} = \frac{n_{(i \rightarrow j)}}{n_i}, \quad (3.16)$$

де p_{ij} – ймовірність переходу системи зі стану i у стан j за один тактовий крок; $n_{(i \rightarrow j)}$ – кількість спостережених переходів безпосередньо зі стану i до стану j у межах поточного вибіркового вікна; n_i – загальна кількість фіксацій стану i як вихідного протягом того самого вікна, і при цьому ймовірність збою за один крок – це p_{iF} (з “Critical” у “Fail” чи “Warning” у “Fail”), а при перевищенні порога θ – запускаємо превентивні дії, які полягають в програмному скиданні.

За необхідності розглядаємо прогноз на r кроків через P^r . Програмне скидання еквівалентне примусовому поверненню системи з небезпечного стану в “Normal” (або “Recovery”). Повне скидання через АСТ відбувається, якщо система сповіщає про критичну проблему, але програмне скидання не дає результату.

Оскільки модуль ПЗСК виконує функції прогнозування стану системи та прийняття рішень про превентивне відновлення в межах одного циклу опитування ОСРЧ, його ефективність визначається не лише точністю оцінювання ризику, а й впливом на часові та обчислювальні ресурси системи, що є критичним для вбудованих пристроїв реального часу.

Основна мета полягає у тому, щоб реалізувати обчислення й оновлення марковської матриці переходів $P(k)$ і прийняття рішень щодо стану системи протягом фіксованого інтервалу між виконаннями задачі моніторингу. Загальний час одного циклу ПЗСК визначимо сумою трьох складових так:

$$t_{upd} = t_{col} + t_{calc} + t_{act}, \quad (3.17)$$

де t_{col} – час збору даних про стан системи, а саме завантаження черг, сигнали активності, затримки; t_{calc} – час обчислення поточного стану та оновлення матриці $P(k)$; t_{act} – час прийняття рішення та виконання можливої дії, наприклад, програмне скидання або передача сигналу апаратному сторожовому таймеру.

Для марковської моделі з n станами обчислювальна складність оновлення матриці переходів є квадратичною – $O(n^2)$. У типовому випадку $n = 4$ ($\{N, W, C, F\}$), це лише 16 елементарних операцій підрахунку частот, які виконуються послідовно в межах одного циклу ПЗСК. Така кількість обчислень не створює помітного навантаження навіть для мікроконтролерів із частотою 100–200 МГц і з обмеженою пам'яттю.

Додаткове навантаження на процесор можна оцінити як відносну частку часу, витраченого на роботу ПЗСК, від періоду його виклику, і визначимо його так:

$$\eta_{CPU} = \frac{t_{upd}}{T_{task}} \times 100\%, \quad (3.18)$$

де T_{task} – період запуску ПЗСК (наприклад, інтервал між двома перевірками системного стану). За умов типової реалізації t_{upd} залишається сталим і не перевищує кількох мілісекунд, тоді як T_{task} зазвичай становить сотні мілісекунд.

Це забезпечує значення $\eta_{CPU} < 2\%$, що практично не впливає на планування задач у FreeRTOS і не змінює детермінованість системи.

Використання ковзного вікна також сприяє мінімізації ресурсних витрат. На кожному кроці алгоритм додає нове спостереження й видаляє найстаріше, що забезпечує постійний розмір буфера даних та сталу кількість обчислень на ітерацію.

Таким чином, обсяг пам'яті, необхідний для роботи ПЗСК, зростає лінійно з кількістю станів, яку можна оцінити значенням величини $O(n)$. Для більшості практичних конфігурацій достатньо зберігати лічильники переходів та допоміжні змінні, що потребує лише кількох сотень байтів оперативної пам'яті. Ще однією важливою характеристикою є часовий вплив на загальний цикл ОСРЧ. ПЗСК виконується у вигляді окремої задачі з високим пріоритетом, однак його тривалість строго обмежена та не перевищує часові межі одного кванта планувальника. Завдяки цьому не порушується виконання інших критичних задач, а прогнозна функція виконується асинхронно, без блокування ресурсів ядра.

Загальний час реакції системи на потенційну деградацію можна визначити як суму часу збору статистики та обчислення рішення:

$$t_{react} = t_{col} + t_{calc}, \quad (3.19)$$

що у запропонованій архітектурі становить частку від періоду T_{task} .

Це гарантує, що навіть у найгіршому випадку система виявить критичний стан і виконає локальне відновлення протягом одного циклу моніторингу.

Таким чином, розроблено метод превентивного відновлення компонентів ОСРЧ, що базується на використанні низькорозмірної марковської моделі прогнозування стану системи та двоетапного механізму сторожового контролю, який поєднує програмний прогнозний шар і апаратний сторожовий таймер. Запропонований підхід дає змогу оцінювати ризик переходу системи у критичний стан на основі статистики переходів і поточних метрик, своєчасно виконувати локальне відновлення компонентів і, за необхідності, ініціювати повний перезапуск системи. На відміну від традиційних схем сторожового контролю, метод забезпечує проактивне виявлення деградації працездатності та скорочення часу простою за рахунок прогнозування розвитку станів і використання політики ескалації від програмного до апаратного рівня відновлення.

3.3 Метод виявлення фальсифікацій та аномалій у комп'ютерних системах для сигналів КФС на основі модифікованого комбінованого фільтра Калмана

Новий метод базується на класичному циклі фільтра Калмана прогноз–уточнення, але доповнюється дискретним адаптивним перемиканням режимів та вбудованим модулем виявлення аномалій. На відміну від традиційних адаптивних фільтрів, що безперервно підлаштовують параметри, запропонований підхід працює в режимі подій. Він лише тоді вносить корективи, коли виявлено значну невідповідність моделі, тобто коли наявна потенційна аномалія або зміна режиму роботи. Така подіє-орієнтована адаптація дозволяє зменшити обчислення у спокійні періоди, але швидко реагувати під час різких змін.

Запропонований метод можна умовно розбити на кілька модулів чи підсистем:

1. Базовий фільтр оцінки стану. Основою системи є фільтр Калмана, що обчислює оцінку стану $\hat{x}_k$ та оцінює похибку (коваріацію P_k) рекурсивно. У нормальному режимі (без аномалій) він працює подібно до звичайного лінійного дискретного фільтра Калмана, використовуючи актуальну модель динаміки.

2. Банк моделей режимів. Для системи заздалегідь визначено скінченну множину моделей M_i , де $i = 1, 2, \dots, R$ відповідає можливим режимам роботи КФС. Кожна модель M_i – це набір матриць стану $(A_i, B_i, C_i, Q_i, R_i)$ для відповідного режиму. В цій моделі A_i – матриця переходу стану, що описує динаміку системи у i -му режимі, B_i – матриця керування, яка визначає вплив керуючих сигналів на стан системи, C_i – матриця спостереження, що відображає зв'язок між станом системи та вимірюваннями, Q_i – коваріаційна матриця процесного шуму, R_i – коваріаційна матриця шуму вимірювання; індекс i позначає режим роботи системи. Наприклад, для безпілота різні режими можуть відповідати польоту на крейсерській швидкості, маневру ухилення, зависанню тощо, тобто з різними динамікою і шумами. На відміну від IMM, в один момент часу активна лише одна модель, та, яку фільтр вважає поточною. Решта моделей пасивні і не обчислюються паралельно, зберігаються лише

як альтернативні моделі для можливого перемикавання. Це одразу знижує навантаження у R разів порівняно з одночасним виконанням R фільтрів.

3. Модуль перевірки інновацій (виявлення аномалій). На кожному кроці алгоритм аналізує вектор інновації $r_i = z_k - C_{curr}\hat{x}_{k|-1}$, тобто різницю між фактичним виміром z_k та прогнозом фільтра $\hat{x}_{k|-1}$, перетвореним у простір вимірювань. Цей модуль обчислює показники, що характеризують відхилення, або ж накопичувальну суму відхилень. Наприклад, нормалізовану нев'язку (нормалізована статистика інновацій) визначимо так:

$$d_k = r_k^T S_k^{-1} r_k, \quad (3.20)$$

де S_k – коваріаційна матриця інновацій; r_k – інновація, різниця між фактичним вимірюванням сенсорів у момент k і тим, що очікувала модель; k – дискретний номер кроку фільтрації, позначки $(.)^T$ і $(.)^{-1}$ – відповідно транспонування та матричне обернення, операції над матрицями.

Якщо ці показники перевищують заздалегідь визначені пороги, то модуль класифікує ситуацію як потенційну аномалію/атаку або зміну режиму.

4. Логіка розпізнавання аномалії та зміни режиму. Ключова особливість полягає в тому, що після фіксації великого відхилення система повинна визначити причину та стан, тобто встановити раптовий перехід в інший нормальний режим чи некоректний сигнал внаслідок атаки/збою. Для цього застосовуються декілька евристик:

- якщо патерн відхилення відповідає відомому новому режиму, наприклад, зміна швидкості руху на іншу сталу величину, очікувану в режимі M_j , то система трактує це як легітимну зміну режиму;

- якщо ж відхилення нестандартне або хаотичне, не узгоджується з жодним типовим сценарієм, або сигнал виходить за фізичні межі можливого, то ймовірно це фальсифікація чи відмова датчика.

Для більшої впевненості алгоритм може перевірити короткочасну поведінку. При справжньому переключенні режиму відхилення носитиме стрибкоподібний але

одиничний характер, після чого фільтр перебудувавшись знову почне давати малі залишкові помилки. При атаці ж залишкові помилки можуть залишатися високими або набувати нестабільного характеру протягом тривалішого часу, так як атака не узгоджена з реальною динамікою.

5. Адаптивний перемикач режимів. Якщо визначено, що відбулося саме переключення режиму, тоді фільтр оперативно перемикається на нову модель M_j . Це означає, що замість старих матриць $(A_{curr}, Q_{curr}, R_{curr} \dots)$, він підставляє у формули Калмана відповідні $(A_j, Q_j, R_j \dots)$, для нового режиму. Перемикання може включати також переініціалізацію деяких змінних фільтра. Наприклад, матриця помилок P_k може бути збільшена, щоб відобразити невпевненість після зміни, а оцінка стану $\hat{x}_k$ – скоригована, якщо відомий детермінований стрибок, наприклад, якщо режимно змінюється сама структура стану. Важливо, що перемикання виконується миттєво на рівні алгоритму і займає сталий, невеликий час зведений до кількох операцій присвоєння/копіювання і, можливо, обчислення нової коваріації.

6. Обробка аномалії (фальсифікації). Якщо модуль перевірки вирішив, що має місце аномалія/атака, алгоритм може виконати кілька дій для зменшення її впливу.

По-перше – це ігнорування підозрілого виміру. Один із прийомів робастної фільтрації полягає в пропуску оновлення на даному кроці, якщо вимір ненадійний. Тому, коли отримано сигнал про аномалію, поточне вимірювання z_k може бути відкинуте або обмежене, наприклад, використовуючи механізм статистичного відсікання вимірювань, тобто якщо d_k надто велике, оновлення стану не виконується. Таким чином, відверто фальшиві дані не отруюють оцінку стану.

По-друге – це введення віртуального коригувального входу. Інший варіант полягає в розширенні стану або моделі, додавши змінну, що представляє собою невідому аномалію. Наприклад, можна ввести вектор помилки f_k у рівняння стану чи вимірювання, який моделюється як додатковий шум з великою дисперсією. Фільтр Калмана тоді спробує оцінити цю змінну, фактично відфільтрувавши компонент

аномалії з сигналу. Цей прийом аналогічний введенню систематичної похибки в модель. Відомі методи уведення уявного шуму [130] або розширеного стану [131] для відстеження відмов. Перший метод полягає у штучному збільшенні шумів процесу для підвищення чутливості до нестационарностей, а другий – у включенні додаткових змінних до вектора стану для явного оцінювання джерела збурення чи зсуву датчика. Обидва підходи дозволяють підвищити робастність оцінювання, однак потребують збільшення розмірності моделі або емпіричного налаштування дисперсій шумів. Аналогічний ефект досягається аналітичним способом в цих кроках методу. Через адаптивне масштабування апіорної коваріації похибки прогнозу за допомогою коефіцієнта β_k , який визначено за формулою (3.25) без необхідності змінювати структуру рівнянь стану.

7. Генерація тривожного сповіщення. Паралельно, метод реєструє подію аномалії. Це може використовуватися вищим рівнем системи безпеки, щоб вжити заходів, наприклад, перейти в безпечний режим роботи, активувати резервні сенсори, сповістити оператора.

Запропонований метод описується математично як гібридна система з перемиканнями, що включає неперервну оцінку стану та дискретні події перемикання/аномалії.

Нехай $x_k \in R^n$ – вектор стану КФС на дискретному такті k , а $z_k \in R^m$ – вектор вимірювань. Система має скінченну кількість моделей (режимів) $i \in \{1, \dots, N\}$. У режимі i динаміка задається рівняннями стану та виходу так:

$$X_{k+1} = A_i x_k + B_i u_k + w_k^i, \quad (3.21)$$

де $x_{k+1} \in R^n$ – вектор наступного стану системи після одного такту або кроку дискретизації, який визначається на основі попереднього стану x_k , керуючих впливів u_k та збурень w_k^i ; A_i – матриця динаміки системи розмірності $n \times n$ для режиму i , яка визначає, як поточний стан x_k впливає на стан X_{k+1} та відображає внутрішні зв'язки між змінними стану у конкретному режимі роботи, наприклад, нормальному,

критичному тощо; B_i – матриця керування розмірності $n \times p$, що визначає, як керуючий вектор u_k впливає на зміну стану системи у режимі i ; $u_k \in R^p$ – вектор керуючих дій або вхідних сигналів, що впливають на систему в момент часу k і це можуть бути дії контролера, сигнали регулятора або коригувальні впливи на ресурси; $w_k^i \in R^n$ – вектор випадкових збурень (шуму або неконтрольованих впливів) у режимі i , який моделює похибки вимірювання, флуктуації навантаження, збої або інші стохастичні фактори; k – дискретний момент часу або індекс такту, на якому відбувається обчислення стану системи.

Таким чином, формулою (3.21) описано динаміку внутрішнього стану системи в обраному режимі під впливом керуючих дій та випадкових збурень. Проте в реальних КФС вектор стану x_k безпосередньо не спостерігається, а доступний лише через результати вимірювань сенсорів або програмних засобів моніторингу, які містять похибки вимірювання та шуми. Для встановлення зв'язку між внутрішнім станом системи та доступними спостереженнями введемо співвідношення вимірювань так:

$$z_k = C_i x_k + v_k^i, \quad (3.22)$$

де z_k – вектор вимірювань у момент часу k містить дані, що надходять від сенсорів або програмних моніторів системи, наприклад, температура, напруга, використання ресурсів, затримка задач тощо; C_i – матриця вимірювань у режимі i визначає, які саме змінні стану x_k відображаються у векторі вимірювань z_k , причому кожен режим $i \in \{1, \dots, N\}$ може мати власну матрицю C_i , що враховує особливості різних робочих умов системи; x_k – вектор стану системи на дискретному такті k ; v_k^i – вектор вимірювального шуму у режимі i , який моделює похибки сенсорів, втрати точності або завади в каналах зв'язку.

Введемо приховану дискретну змінну режиму $m_k \in \{1, \dots, R\}$, що визначатиме, яка модель справедлива на кроці k . В нормальній ситуації m_k змінюється рідко і лише тоді, коли система переходить в інший режим роботи.

Можливі зловмисні впливи чи відмови сенсорів моделюємо як додатковий аномальний сигнал f_k , що може додаватись до вимірів, імітуючи, наприклад, підробку показів датчика, або до стану – імітуючи, наприклад непередбачене зовнішнє збурення. Без втрати загальності розглянемо адитивну підробку вимірів:

$$z_k^{obs} = z_k + Df_k, \quad (3.23)$$

де z_k^{obs} – дійсно спостережуване значення датчика; D – матриця, що визначає, які компоненти сигналу піддалися атаці.

Для спрощення обчислень можна прийняти $D = I$, тоді атака безпосередньо додається до виміряного виходу. Вектор f_k невідомий, його статистика наперед не визначена і може бути довільною послідовністю, навіть негаусовою і корельованою, оскільки нападник не обмежений стандартними умовами шуму.

Щодо завдання фільтрації та виявлення, то на кожному кроці k необхідно оцінити стан $\hat{x}_{k|k}$, а також прийняти рішення щодо m_k , тобто визначити активний режим, і f_k – визначити, чи відбулась атака, тобто $f_k \neq 0$. Це типова задача одночасної оцінки стану і виявлення змін у моделі. Тому, здійснимо розділення процесу на два рівні.

Рівень 1. Оцінка стану (неперервний рівень). Виконується через ККФ, який в кожен момент працює з деякою моделлю $M_{curr} = (A_{curr}, B_{curr}, C_{curr}, Q_{curr}, R_{curr})$. Модифікації фільтра включають механізм ковзного збільшення помітливості, та зменшення впливу підозрілих вимірювань:

1. Механізм ковзного збільшення помітності. У разі виявлення невідповідності, можна збільшити апіорну коваріацію помилки $P_{k|k-1}$ шляхом множення на фактор $\alpha > 1$ (аналог коефіцієнта згасання ФСС). Цей прийом робить підсилення Калмана K_k більшим, отже фільтр більше довірятиме новим даним, швидше наздожене раптову зміну. Фактично, замість класичного оновлення P використаємо $P_{k|k-1} := \alpha P_{k|k-1}$, якщо інновація перевищує поріг. Таке просте коригування випливає з принципу ортогоналізації інновацій, запропонованого для фільтрів сильного стеження [140].

Для більш гнучкого та аналітично обґрунтованого керування чутливістю фільтра можна ввести адаптивне коригування коваріації прогнозу. У цьому випадку апріорна матриця $P_{k|k-1}$ масштабуються множителем β_k , що визначається ступенем відхилення статистики інновацій d_k від порогового значення τ_k так:

$$P_{k|k-1}^* = \beta_k P_{k|k-1}, \beta_k = 1 + \gamma \frac{d_k - \tau_\alpha}{\tau_\alpha} I(d_k > \tau_\alpha), \quad (3.24)$$

де β_k – адаптивний коефіцієнт масштабування апріорної коваріації, який визначає рівень збільшення невизначеності оцінки; d_k – нормалізована нев'язка (3.20), яка відображає ступінь розбіжності між прогнозом та реальним виміром; $\tau_\alpha = \chi_{m,1-\alpha}^2$ – порогове значення статистики, визначене за критерієм χ^2 для заданого рівня значущості α і розмірності вектора вимірів m ; γ – коефіцієнт адаптації, що регулює швидкість зростання невизначеності; $I(\cdot)$ – індикаторна функція, яка дорівнює 1 у разі перевищення порогу і 0 – інакше; $P_{k|k-1}^*$ – оновлена апріорна коваріація, скоригована відповідно до рівня виявленої аномалії.

Такий підхід дозволяє плавно змінювати невизначеність прогнозу залежно від поточного рівня аномальності вимірів, зберігаючи стабільність оцінювання навіть у нестійких або зашумлених умовах. Це є природним розширенням механізму ковзного збільшення помітливості й надає моделі формальну адаптивність без введення додаткових станів.

2. Зменшення впливу підозрілих вимірювань. Якщо виявлено можливу аномалію, замість звичайного Kalman-оновлення застосовується згаданий вище режим жорсткого пропуску або зменшення ваги виміру. Математично це можна записати як збільшення матриці $R_{eff} = \beta R_{curr}$ з великим β на даний крок, або навіть $K_k := 0$, що еквівалентно ігноруванню виміру. У результаті $\hat{x}_{k|k} = \hat{x}_{k|k-1}$, $P_{k|k} = P_{k|k-1}$, або трохи зростає. Такий підхід схожий на робастний фільтр Калмана з перевіркою достовірності вимірів [132].

Рівень 2. Дискретне керування моделлю (логіка подій). Цей рівень приймає рішення про зміну режиму M_k або фіксацію атаки f_k . Математично це можна задати гіпотезою так:

- 1) H_0 - режим не змінився, атаки немає (нормальний стан);
- 2) $H_1(j)$ - відбувся перехід на режим j (для деякого $j \neq curr$);
- 3) H_2 - була аномалія (атака).

Для прийняття рішення між гіпотезами використовуються дві статистики на основі інновацій, які порівнюються з відповідними пороговими значеннями. Формально, ведемо дві статистики: $\Lambda_k^{(m)}$ для перевірки зміни режиму (гіпотеза $H_1(j)$) і $\Lambda_k^{(f)}$ для перевірки атаки (гіпотеза H_2). Якщо обидві статистики нижче своїх порогів, то приймається гіпотеза H_0 . При цьому перевірка гіпотези H_2 має пріоритет над $H_1(j)$. Статистику для перевірки зміни режиму визначимо так:

$$\Lambda_k^{(m)} = \max_{j \neq curr} \left\{ - \left(z_k - C_j \hat{x}_{k|k-1}^{(j)} \right)^T \left(C_j P_{k|k-1} C_j^T + R_j \right)^{-1} \left(z_k - C_j \hat{x}_{k|k-1}^{(j)} \right) \right\}, \quad (3.25)$$

де $C_j \hat{x}_{k|k-1}^{(j)}$ – прогноз стану, отриманий екстраполяцією минулої оцінки $\hat{x}_{k|k-1}$ за моделлю M_j ; $\max_{j \neq curr}$ – оператор максимізації, статистика обчислюється для всіх можливих альтернативних моделей $j \neq curr$, а береться найвище значення – тобто найвірогідніша альтернатива поточному режиму; z_k – вектор фактичних вимірювань, отриманий від сенсорів у момент часу k ; C_j – матриця вимірювань, яка відповідна моделі M_j .

Вона визначає, які компоненти стану x_k впливають на вимірювання у цьому режимі. Ця статистика відображає, наскільки краще пояснює новий вимір модель M_j у порівнянні з поточною M_{curr} . Якщо для деякого j це значення перевищує певний поріг, вирішуємо $H_1(j)$ – тобто перемикаємось на режим j .

Водночас статистика для атаки $\Lambda_k^{(f)}$ ототожнюється з нормалізованою нев'язкою $d_k(3.20)$, перевищення порогового значення сигналізує про аномалію чи фальсифікацію. Якщо $\Lambda_k^{(f)}$ дуже велика навіть для найкращої моделі включно з потенційними j , тобто вимір не вписується ні в одну нормальну динаміку, то приймаємо H_2 – наявна фальсифікація.

У практичному плані, реалізація спрощується, немає потреби перевіряти всі режими щоразу. Можна, наприклад, тримати припущення, в який режим може перейти система (якщо є скінченний автомат режимів, відомий з дизайну системи). Тоді j перевіряється лише з набору допустимих переходів із поточного режиму. Також оцінку $\hat{x}_{k|k-1}^{(j)}$ можна отримувати без запуску повноцінного фільтра, а одним кроком прогнозу для моделі j . Це суттєво скорочує обчислення порівняно з повним паралельним фільтруванням.

Таким чином, математична модель методу враховує стандартні рівняння Калмана для оцінювання стану в рамках активного режиму, а також додаткові правила прийняття рішень (умови на статистики інновацій), які неявно змінюють параметри фільтра, наприклад, перемикають модель або змінюють K_k , R_k , Q_k . Ці правила можна розглядати як алгоритм високого рівня, накладений на систему рівнянь Калмана. Формально він вносить нелінійність і нестационарність у процес оцінювання, але за рахунок цього досягається гнучкість і стійкість до аномалій.

Алгоритм роботи методу виявлення фальсифікацій і аномалій у сигналах КФС у циклі реального часу включає такі кроки:

1. Прогноз стану. Використовуючи поточну активну модель M_{curr} , обчислити прогноз $\hat{x}_{k|k-1} = A_{curr}\hat{x}_{k-1|k-1} + B_{curr}U_k$ і коваріацію прогнозу $P_{k|k-1} = A_{curr}P_{k-1|k-1}A_{curr}^T + Q_{curr}$. Таким чином, на цьому етапі формується апіорна оцінка стану $\hat{x}_{k|k-1}$ та відповідна їй апіорна оцінка похибки $P_{k|k-1}$, які використовуються на наступному кроці для обчислення інновації та перевірки узгодженості вимірювань із прогнозом. Використання режимно-залежних матриць A_{curr} , B_{curr} та Q_{curr} дозволяє

враховувати зміну динаміки системи без запуску паралельних фільтрів і з мінімальними обчислювальними витратами, що є критичним для реалізації алгоритму в ОСРЧ.

2. Обчислення інновації. Зібрати нове вимірювання z_k , обчислити нев'язку $r_k = z_k - Q_{curr}\hat{x}_{k|k-1}$ та її нормалізовану величину $d_k = r_k^T S_k^{-1} r_k$, де S_k обчислюється як:

$$S_k = C_{curr}P_{k|k-1}C_{curr}^T + R_{curr}, \quad (3.26)$$

де S_k - коваріаційна матриця інновації (нев'язки) визначає дисперсію помилки між реальними вимірюваннями z_k і прогнозом вимірювання; C_{curr} – матриця вимірювань (спостереження) для поточного активного режиму M_{curr} визначає, які елементи стану x_k безпосередньо впливають на вимірювані змінні z_k ; C_{curr}^T – транспонована матриця C_{curr} забезпечує симетричність матричних операцій при обчисленні коваріації; R_{curr} – коваріаційна матриця шуму вимірювань для поточного режиму визначає рівень довіри до сенсорних вимірювань так, що якщо шум великий, фільтр покладається більше на модель, а не на вимірювання.

3. Перевірка аномалії/перемикання. Якщо d_k не перевищує поріг (ситуація нормальна), перейти до кроку 4 (звичайне оновлення). Якщо ж d_k великий або має незвичну структуру, активувати модуль розпізнавання. Виконати правила визначення чи відповідає r_k можливому новому режиму. Наприклад, перевірити схожість r_k з очікуваним стрибком при переході на режим M_j , для всіх $j \neq curr$. Якщо так, то встановити прапор перемикання режиму на M_j . Якщо ні, то встановити прапор аномалії (атаки).

4. Оновлення стану (корекція). Якщо встановлено прапор перемикання режиму, то негайно перемкнути активну модель $M_{curr} := M_j$, що відповідає новому режиму функціонування системи. З метою врахування стрибкоподібної зміни динаміки та зростання невизначеності оцінки стану здійснюється коригування апіорної коваріаційної матриці похибки $P_{k|k-1}$ шляхом її масштабування або переініціалізації

значеннями, характерними для нового режиму. За наявності апіорної інформації про систематичний зсув стану при переході між режимами додатково виконується корекція апіорної оцінки стану $\hat{x}_{k|k-1}$. Після цього здійснюється стандартна процедура корекції фільтра Калмана з використанням матриць спостереження C_{curr} та шумів вимірювань R_{curr} , що відповідають поточному режиму так:

$$K_k = P_{k|k-1} C_{curr}^T S_k^{-1}, \quad (3.27)$$

де, K_k – коефіцієнт (підсилювач) Калмана, який визначає, наскільки сильно нові вимірювання впливають на оновлення оціненого стану; $P_{k|k-1}$ – коваріаційна матриця похибки прогнозу стану перед корекцією (апіорна оцінка); C_{curr}^T – транспонована матриця вимірювань для поточного режиму M_{curr} . S_k – коваріація інновації (формула (3.1)).

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k r_k, \quad (3.28)$$

де $\hat{x}_{k|k}$ – оновлена (апостеріорна) оцінка стану після врахування нових вимірювань. $\hat{x}_{k|k-1}$ – попередня (апіорна) оцінка стану, отримана на основі рівняння прогнозу; K_k – коефіцієнт Калмана, розрахований за попередньою формулою (3.27); r_k – інновація, що визначає різницю між фактичним і прогнозованим вимірюванням так:

$$r_k = z_k - C_{curr} \hat{x}_{k|k-1}, \quad (3.29)$$

де z_k – фактичне вимірювання, отримане від сенсорів у момент часу k , що є реальним сигналом, який спостерігається в системі; C_{curr} – матриця вимірювань (спостереження) для поточного режиму M_{curr} визначає, які елементи внутрішнього стану x_k відображаються у вимірюванні z_k .

Після перемикання r_k буде переобчислено за оновленою моделлю або можна використати попереднє r_k якщо різниця моделей мінімальна, що залежить від реалізації.

Після корекції оцінки стану уточнюється коваріаційна матриця похибки оцінювання, що визначається співвідношенням так:

$$P_{k|k} = (I - K_k C_{curr}) P_{k|k-1} \quad (3.30)$$

де $P_{k|k}$ – оновлена коваріаційна матриця похибки стану, що показує нову невизначеність після корекції; I – одинична матриця відповідної розмірності; K_k – коефіцієнт Калмана; C_{curr} – матриця вимірювань для поточного режиму; $P_{k|k-1}$ – апостеріорна коваріація стану з попереднього кроку прогнозу.

Якщо встановлено прапор аномалії, тоді необхідно виконати одну з політик обробки аномалії. Найпростіше це пропустити оновлення, тобто не змінювати $\hat{x}_{k|k-1}$ (або змінити мінімально) і встановити $P_{k|k} = P_{k|k-1}$ збільшене, щоб врахувати, що є невпевненість через втрату достовірного вимірювання. Інший варіант полягає в тому, щоб виконати оновлення зі зниженою довірою до виміру, наприклад, масштабувати K_k на малий фактор або збільшити R_{curr} на цей крок, щоб зменшити вплив r_k . У будь-якому разі метою є заборона аномальному виміру вносити значну похибку в оцінку. Після цього можна тимчасово увімкнути режим підвищеної чутливості, тобто надалі зменшити пороги для виявлення, збільшити частоту перевірок чи активувати резервний механізм перехресної перевірки з іншими датчиками, якщо такі наявні.

Якщо прапорів немає (звичайний випадок) – виконати стандартне оновлення Калмана за наведеними формулами з поточною моделлю.

5. Вихідні дані і повторення циклу. Після завершення процедури корекції отримуємо оновлену оцінку стану $\hat{x}_{k|k}$, яку можна використовувати як вхідні дані для задач керування, моніторингу або прийняття рішень у складі ОСРЧ, зокрема для коригування керуючих впливів, зміни пріоритетів задач або оцінювання стабільності функціонування системи. У випадку виявлення аномалії або ознак атаки формується сигнал тривоги, який може бути використаний для ініціалізації додаткових перевірок або захисних дій. Це дозволяє системі своєчасно реагувати на нестандартні ситуації без негайного порушення основного циклу роботи. Далі цикл переходить на

наступний такт $k + 1$, повторюючи кроки 1-5. Таким чином забезпечується безперервна робота алгоритму в реальному часі.

Реалізація кроків методу компактно реалізується в умовах вбудованої системи реального часу. З погляду програмної структури на FreeRTOS, алгоритм може бути виконаний як окрема задача, що запускається з періодом надходження нових даних від сенсорів. Він не блокує інші задачі, оскільки між вимірюваннями виконує лише короткі перевірки і обчислення. Всі необхідні моделі та параметри зберігаються у пам'яті, наприклад, у вигляді структур або масивів, що уникає динамічного виділення пам'яті. У випадку, якщо система переходить в інший режим, наприклад, зміна конфігурації пристрою, вищий рівень ПЗ може сигналізувати алгоритму про цю зміну, але наш метод здатен і самостійно ідентифікувати її за аномальними інноваціями.

Таким чином, розроблено метод виявлення фальсифікацій та аномалій у комп'ютерних системах для сигналів КФС на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів. Запропонований підхід, на відміну від поширених багатомодельних і безперервно адаптивних фільтрів, не потребує виконання набору моделей, забезпечує селективне пригнічення підозрілих вимірювань на основі аналізу інновацій та дозволяє відрізнити нормальні режимні зміни від зловмисних відхилень. Це забезпечує низьку обчислювальну складність, швидке реагування на аномалії та зменшення кількості хибних спрацювань, що робить метод придатним для ресурсообмежених кіберфізичних систем реального часу.

3.4 Висновки до третього розділу

У роботі представлено метод превентивного відновлення компонентів ОСРЧ, який поєднує АСТ у ролі останнього рівня захисту від фатальних збоїв і програмний предиктивний шар з низькорозмірною марковською моделлю. Такий підхід дає змогу проактивно виявляти ймовірний збій чи аномалію, що може бути спричинена як технічною відмовою, так і навмисним впливом, та виконувати локальний перезапуск проблемних компонентів. Математична модель опирається на дискретний

марковський ланцюг невеликої розмірності, що дає змогу P оновлювати на льоту без великих обчислювальних витрат. Основною особливістю є комбінування цього компактного стохастичного аналізу з двоступеневим підходом до відмовостійкості. Це дозволяє уникнути переважної більшості некритичних збоїв та економити час порівняно з постійними глобальними перезапусками, водночас зберігаючи гарантію глобального скидання у разі найгірших сценаріїв, підвищити здатність систем протистояти сучасним кіберзагрозам.

Також розроблено метод виявлення фальсифікацій та аномалій у комп'ютерних системах для сигналів КФС на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів. Метод базовано на принципах фільтра Калмана, але усунуто головні недоліки класичних адаптивних підходів, які полягають в надмірному навантаженні та неточності у нестабільних умовах. Суть методу полягає в інтеграції процесу фільтрації стану з механізмом детектування змін. Алгоритм сам розпізнає момент, коли модель системи стала неактуальною або дані скомпрометовані, і відповідно підлаштовується, перемикає модель чи ігнорує аномальні виміри. Метод складається з одного фільтра Калмана зі структурою перемикання та модуля моніторингу інновацій, що приймає рішення про перебудову фільтра. Математичну модель подано як систему з прихованими режимами та можливими атаками, для якої фільтр виконує одночасно оцінювання стану і гіпотезу про режим/атаку. Ключова ідея полягає в подієвій адаптації тільки при значних відхиленнях, що забезпечує миттєву реакцію і високу точність оцінки при різких змінах, уникаючи непотрібних витрат обчислювальних ресурсів у звичайних умовах. Метод розроблено з урахуванням обмежених ресурсів вбудованих систем. Він вимогливий до ресурсів не більше за звичайний фільтр Калмана, не потребує паралельних обчислень та може ефективно бути реалізованим на малопотужних процесорах реального часу.

Основні результати розділу опубліковані у працях [2; 4; 7; 8; 9; 10; 137].

РОЗДІЛ 4

ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА АРХІТЕКТУРИ ОПЕРАЦІЙНИХ СИСТЕМ РЕАЛЬНОГО ЧАСУ ТА РОЗРОБЛЕНИХ МЕТОДІВ В УМОВАХ ІНТЕНСИВНИХ ЗМІН СТАНІВ

4.1 Експериментальне дослідження ефективності динамічного розподілу задач та ресурсів на основі тензорної декомпозиції в ОСРЧ

Для реалізації прототипу за основу візьмемо операційну систему FreeRTOS на базі мікроконтролерної платформи STM32F407G-DISC1 (рис 4.1), де використовується стандартний файл `tasks.c` – основний файл планувальника FreeRTOS, у якому визначаються функції керування задачами. Для розширення планування використаємо власний метод розподілу задач. Також візьмемо `main.c`, де додамо код для створення задач збору даних та виклику розширеного планувальника, який буде в файлі `custom_scheduler.c`. У цьому файлі буде визначено функції збору даних, класифікації задач та керування розподілом ресурсів.

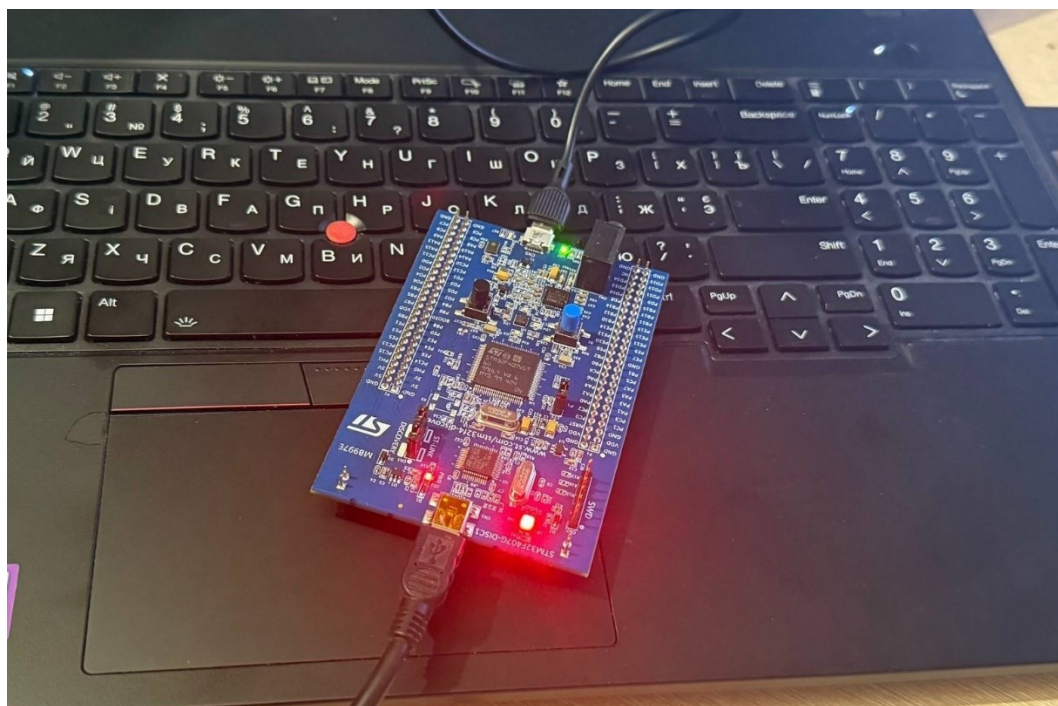


Рисунок 4.1 – Експериментальний стенд на базі плати STM32F407G-DISC1, підключеної до персонального комп'ютера.

Отже поділимо на кроки виконання методу:

1. Збір даних у FreeRTOS. Збираємо дані шляхом запису параметрів під час роботи системи або за допомогою моделювання. Визначаємо основні параметри які будуть використовуватись в моделі, наприклад:

- тип задачі;
- використання ресурсів (CPU);
- використання ресурсів (пам'ять);
- час виконання задачі.

Реалізуємо програмну реалізацію методу. Будемо імітувати виконання задач у системі, записуючи параметри під час виконання. Це можна зробити за допомогою циклів та таймерів для моделювання реального часу. Збір телеметрії здійснювався з періодом 72 секунди (рис 4.2). Аналізуємо і мапимо у потрібний формат моделі зібрані дані для подальшого використання в тензорній декомпозиції та навчанні моделей машинного навчання. Передавання даних з мікроконтролера здійснюється через USB-клас комунікаційних пристроїв (CDC) на мережний сервіс, реалізований на C# (.NET 8), який виконує приймання та обробку даних. На рис 4.3 відображено отримання телеметрії із мікроконтролера.

На основі зібраних даних будуємо багатовимірний тензор. У цьому випадку буде 3-х вимірний тензор $T \in R^{3 \times 3 \times 100}$, де перший вимір – це задачі, другий – характеристики (використання ресурсів (CPU), використання ресурсів (Пам'ять), час виконання задачі), третій – час, коли задачі виконувались. Дискретний момент часу t відповідає фіксованому інтервалу збору телеметрії, у межах якого агрегуються значення використання процесора, пам'яті та часу виконання задач. Це забезпечує порівнюваність вимірювань. Наприклад, для Task 1 перші три спостереження такі:

$$T_{1,j,t} = \begin{bmatrix} 18,7275 & 16,6106 & 13,6275 \\ 8,48 & 8,499 & 5,565 \\ 2508,7768 & 1900,6186 & 1388,5103 \end{bmatrix},$$

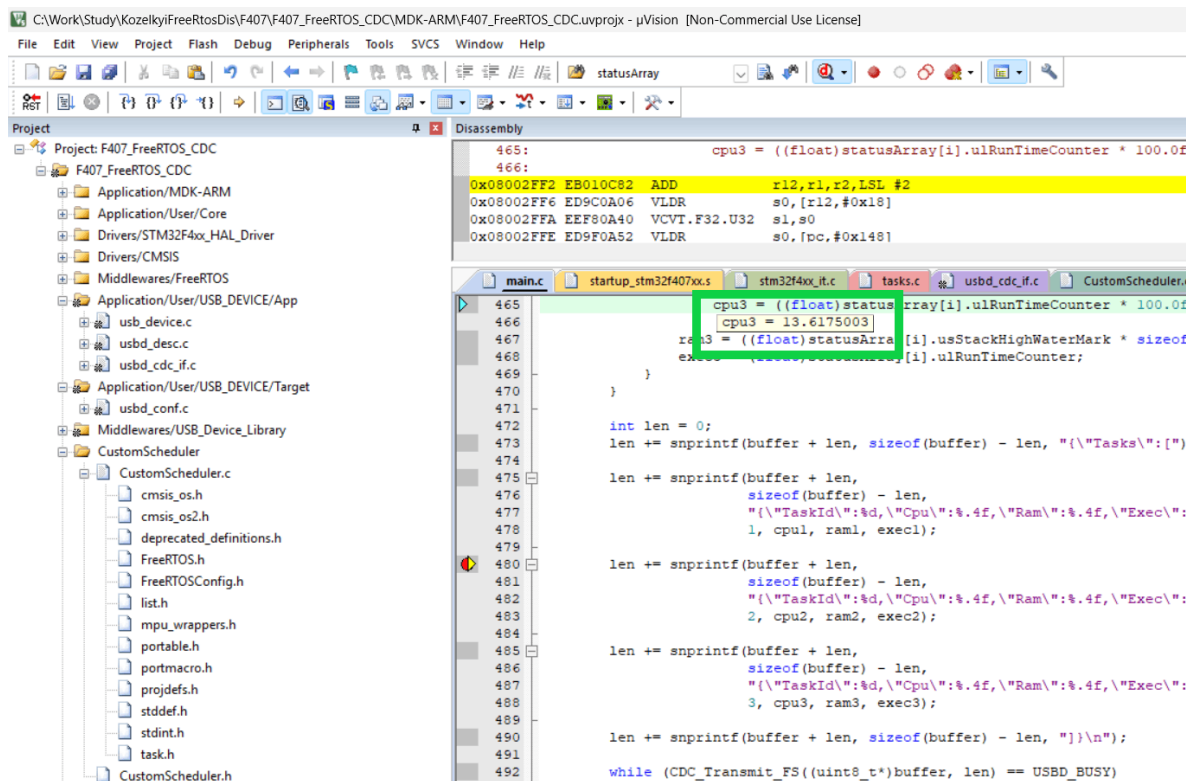


Рисунок 4.2 – Скріншот роботи збору телеметрії з модифікованої FreeRTOS

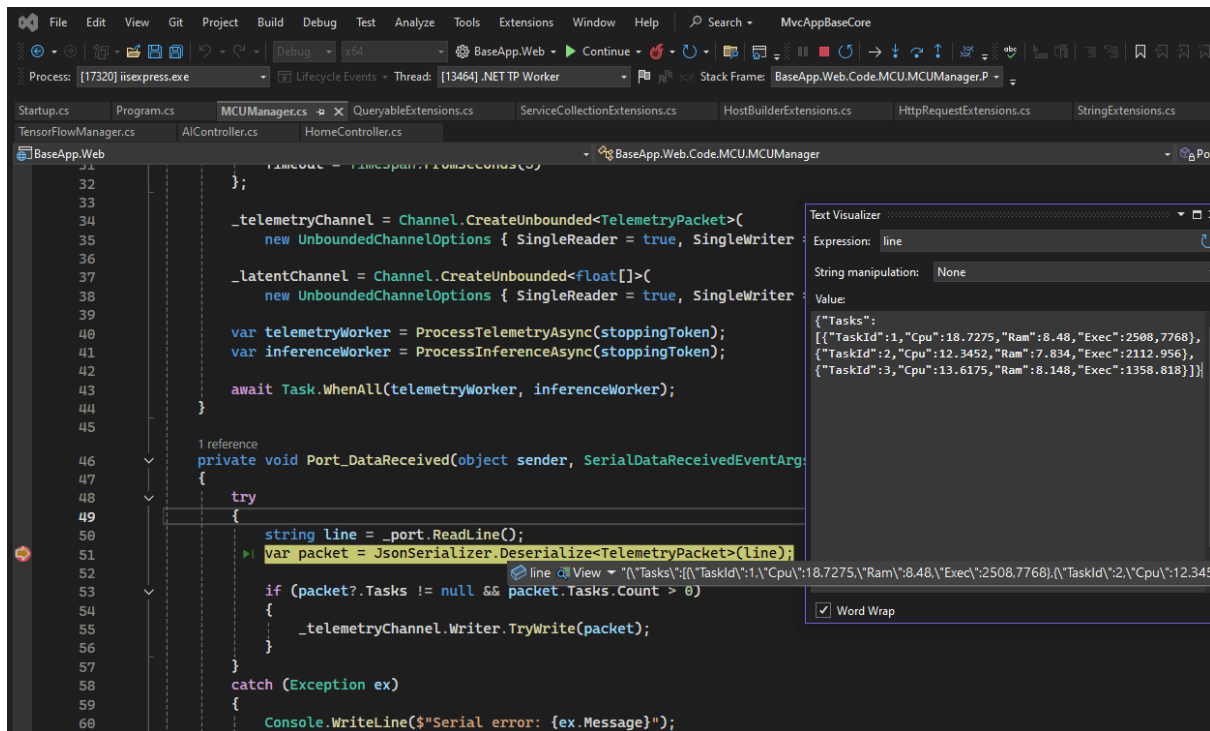


Рисунок 4.3 – Скріншот отримання телеметрії з модифікованої FreeRTOS на сервіс .NET

де $T_{1,j,t}$ – елемент тензора, що відповідає першій задачі, j -й характеристиці та t -му дискретному моменту часу, перший рядок матриці – значення показника CPU у відсотках, другий – використання оперативної пам’яті задачею у кілобайтах, третій – середня тривалість виконання задачі.

Показник CPU визначався як відносна частка процесорного часу, протягом якого відповідна задача перебувала у стані виконання в межах фіксованого інтервалу вимірювання. Значення обчислювалося як відношення сумарного часу активного виконання задачі до тривалості одного вимірювання у відсотках. Середня тривалість виконання задачі обчислювалася як середнє значення часу її перебування у стані виконання за один цикл протягом експериментального періоду.

Далі всі значення нормалізуємо за Z -нормалізацією окремо для кожної метрики, по всьому тензору середні значення становлять $\mu_{CPU} = 16,322$, $\mu_{Mem} = 7,515$, $\mu_{Exec} = 1932,64$, стандартні відхилення (квадратний корінь з дисперсії) – $\sigma_{CPU} = 2,092$, $\sigma_{Mem} = 1,382$, $\sigma_{Exec} = 458,947$.

Нормалізований тензор визначаємо так:

$$T'_{i,j,t} = \frac{T_{i,j,t} - \mu_j}{\sigma_j}, \quad (4.1)$$

де $T_{i,j,t}$ – початкове ненормалізоване значення відповідної метрики, μ_j – середнє значення j -ї характеристики, обчислене за всіма задачами та моментами часу; σ_j – стандартне відхилення j -ї характеристики, обчислене за всіма задачами та моментами часу; $i = 1, \dots, N$ – індекс задачі; $j = 1, \dots, M$ – індекс характеристики; $t = 1, \dots, T$ – індекс дискретного моменту часу. Для того самого фрагмента Task 1 отримуємо:

$$T'_{1,j,t} \approx \begin{bmatrix} 1,1499 & 0,138 & -1,288 \\ 0,6983 & 0,712 & -1,411 \\ 1,2553 & -0,0698 & -1,1856 \end{bmatrix},$$

тобто, наприклад, нормалізоване завантаження CPU для першого вимірювання обчислено як $(18,7275 - 16,322)/2,092 \approx 1,1499$, нормалізоване значення

використання оперативної пам'яті обчислено як $(8,48 - 7,515)/1,382 \approx 0,6983$, а час виконання задачі обчислено як $(2508,7768 - 1932,64)/458,947 \approx 1,2553$.

Отриманий нормалізований тензор T' , сформований із використанням зазначених параметрів μ_j та σ_j , використовувався для подальшої тензорної декомпозиції. Для реалізації тензорної декомпозиції застосовано метод HOSVD [153], який у Math.NET Numerics реалізується через послідовний сингулярний розклад матриць-розгортки. Тензор T' по черзі розгортається за вимірами задач, характеристик і часу, утворюючи матриці розмірів 3×300 , 3×300 та 100×9 . Для кожної з них виконується сингулярний розклад (SVD):

$$A = U\Sigma V^T, \quad (4.2)$$

де U – ортогональна матриця лівих сингулярних векторів; Σ – діагональна матриця сингулярних чисел, впорядкованих у порядку спадання, причому всі вони є невід'ємними; V – ортогональна матриця правих сингулярних векторів; V^T – транспонована матриця V .

У результаті сингулярного розкладу отримано матриці U , Σ та V^T . Матриця V^T у роботі не наводиться через значну розмірність і відсутність необхідності її використання в подальшому аналізі, оскільки інтерпретація результатів виконується за матрицею сингулярних чисел Σ та матрицею U .

Сингулярні значення Σ_{task} отримано в результаті застосування цього розкладу до матриці розгортки $T'_{(1)}$, сформованої з нормалізованого тензора T' . Вони відповідають трьом сингулярним числам цієї матриці. Наприклад, для розгортки за задачами $T'_{(1)} = U_{task}\Sigma_{task}V_{task}^T$ зі сингулярними значеннями $\Sigma_{task} \approx [17.95, 17.47, 16.51]$ та матрицею

$$U_{task} = \begin{bmatrix} -0,2846 & 0,5810 & -0,7625 \\ -0,2568 & 0,7201 & 0,6446 \\ 0,9236 & 0,3793 & -0,0558 \end{bmatrix}.$$

Тут Σ_{task} показує, яка частка структурної варіації нормалізованої телеметрії припадає на кожну незалежну латентну компоненту в просторі задач. Тобто, вони

показують, наскільки суттєво задачі відрізняються одна від одної з урахуванням усіх метрик (CPU, пам'ять, час виконання) та всіх часових проміжках експерименту, дисперсію кожної латентної компоненти. Більше значення елементу масиву Σ_{task} означає, що відповідний напрямок описує більшу частину змін у поведінці задач. U_{task} – показує що саме означають ці напрямки в просторі задач, як задачі розташовані вздовж цього напрямку, ортонормовані вектори цього нового базису та визначає, як саме кожна задача проєктується на виділені латентні компоненти, тобто показує внесок конкретної задачі у відповідний структурний напрям варіації телеметричних даних.

Аналогічно для розгортки за характеристиками $T'_{(2)} = U_{metric}\Sigma_{metric}V_{metric}^T$ з $\Sigma_{metric} \approx [18.33, 17.04, 16.55]$ та

$$U_{metric} = \begin{bmatrix} -0,5217 & -0,7518 & -0,4032 \\ 0,6486 & -0,0426 & -0,7599 \\ 0,5541 & -0,6580 & 0,5099 \end{bmatrix}.$$

Третя розгортка виконується за часовим виміром: $T'_{(3)} = U_{time}\Sigma_{time}V_{time}^T$, де Σ_{time} містить три сингулярні числа $\Sigma_{time} \approx [18,12, 17,68, 16,29]$, аналогічні за масштабом до попередніх, а матриця U_{time} має розмірність 100×3 і містить латентні часові патерни, що відображають динаміку змін системи у часі. Наведемо перші три рядки для ілюстрації структури результату, а разом їх 100 рядків.

$$U_{time} = \begin{bmatrix} 0,2912 & 0,6418 & -0,7074 \\ 0,5140 & -0,7512 & -0,4151 \\ 0,083 & 0,1524 & 0,5689 \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{bmatrix}.$$

Оскільки кількість часових зрізів значно більша, ніж кількість латентних компонент, то для подальшого використання формуємо агрегований часовий латентний вектор який компактно представляє часову структуру багатовимірної телеметрії.

Отримані ортонормовані матриці U_{task} , U_{metric} , U_{time} використовуються для побудови латентних представлень телеметрії.

Для подальшого формування ознак обчислюються усереднені та нормалізовані латентні компоненти. Наприклад, для матриці U_{task} , отриманої після SVD-розкладу тензора, середні значення перших трьох латентних напрямків становлять відповідно:

$$\bar{U}_{task} = \left[\frac{-0,2846-0,2568+0,9236}{3}, \frac{0,5810+0,7201+0,3793}{3}, \frac{-0,7625+0,6446-0,0558}{3} \right] \approx [0.1274, 0.5601, -0.0579].$$

Аналогічно, для характеристики ресурсів (CPU, пам'ять, час виконання), що задається матрицею U_{metric} , усереднений вектор латентних впливів набуває вигляду:

$$\bar{U}_{metric} = \left[\frac{-0,5217+0,6486+0,5541}{3}, \frac{-0,7518-0,0426-0,6580}{3}, \frac{-0,4032-0,7599+0,5099}{3} \right] \approx [0.2270, -0.4841, -0.2177].$$

Аналогічно, для характеристики часових патернів, що задається матрицею U_{time} , усереднений вектор латентних впливів набуває вигляду:

$$\bar{U}_{time} = \left[\frac{1}{100} \sum_{t=1}^{100} u_{t,1}, \frac{1}{100} \sum_{t=1}^{100} u_{t,2}, \frac{1}{100} \sum_{t=1}^{100} u_{t,3} \right] \approx [0.537, 0.014, -0.184].$$

Отримані значення $\bar{U}_{task}$, $\bar{U}_{metric}$ та $\bar{U}_{time}$ разом із сингулярними числами Σ_{task} , Σ_{metric} та Σ_{time} формують вхідний багатовимірний вектор ознак.

Для формування коректних керуючих коефіцієнтів використовується процедура попереднього визначення оптимальних значень α для кожного сформованого латентного вектора стану. Оскільки виконання повного експериментального циклу для кожного набору коефіцієнтів є ресурсоємним, то застосовано аналітичну модель впливу α на основні показники системи. В межах цієї моделі коефіцієнти інтерпретуються як вагові множники, що змінюють інтенсивність використання процесора, пам'яті та середній час виконання задач відповідно до поточного латентного стану. Для кожного вікна телеметрії здійснюється стохастичний перебір допустимих значень коефіцієнтів у заданому діапазоні. Для кожного кандидата α на основі аналітичної моделі оцінюються скориговані значення метрик ефективності,

після чого обчислюється інтегральний показник, що враховує завантаження процесора, використання пам'яті та час виконання задач. Оптимальним вважається той набір коефіцієнтів, який забезпечує мінімальне значення інтегрального показника в межах моделі. У результаті формується навчальна вибірка у вигляді пар «латентний стан - оптимальні коефіцієнти», яка використовується для навчання моделі TensorFlow Lite. Таким чином, модель апроксимує попередньо визначений закон керування, отриманий на основі аналітичної оцінки впливу коефіцієнтів на поведінку планувальника.

2. Надсилання даних на навчання. Цей багатовимірний вектор подається до моделі TensorFlow Lite:

$$X = [\Sigma_{task}, \Sigma_{metric}, \Sigma_{time}, \bar{U}_{task}, \bar{U}_{metric}, \bar{U}_{time}] =$$

$$[17.95, 17.47, 16.51, 18.33, 17.04, 16.55, 18.12, 17.68, 16.29,$$

$$0.1274, 0.5601, -0.0579, 0.2270, -0.4841, -0.2177, 0.537, 0.014, -0.184].$$

Вектор X містить компактне латентне представлення багатовимірної телеметрії, отриманої з ОСРЧ. У форматі JSON: {"latent_features": [17.95, 17.47, 16.51, 18.33, 17.04, 16.55, 18.12, 17.68, 16.29, 0.1274, 0.5601, -0.0579, 0.2270, -0.4841, -0.2177, 0.537, 0.014, -0.184]}.

3. Аналіз навченої моделі і виконання прогнозування. Після того як багатовимірний вектор успішно прийнятий, наступним етапом є виконання прогнозування за допомогою навченої моделі TensorFlow Lite. Отримавши розширений вектор ознак, модель виконує оцінювання та коригує значення вагових коефіцієнтів α_{cpu} , α_{ram} , α_{exec} з урахуванням структурних зв'язків між задачами, їхніми ресурсними характеристиками, а також часових патернів навантаження, виявлених матрицею. Це забезпечує адаптивне оновлення ваг залежно від динаміки системи. Модель зменшує значення α_{cpu} у разі коротких пікових стрибків навантаження, підсилює α_{ram} при тривалих періодах високого споживання пам'яті та коригує α_{exec} при стабільному зростанні часу виконання задач. Після цього повертає

ваги назад у зовнішній модуль у форматі JSON: {"alpha_cpu": -0.71, "alpha_ram": -0.05, "alpha_exec": 0.19}.

4. Відправка результатів у мікроконтролер. Після отримання скоригованих вагових коефіцієнтів із Tensorflow Lite (рис 4.4) відправляємо результати із зовнішнього модуля на мікроконтролер для модифікації прийняття рішень у файлі custom_scheduler.c. Там зберігаємо політику. А також ваги розбираються модулем планувальника задач. Для демонстрації використаємо значення, повернуті моделлю $\alpha_{cpu} = -0,71$, $\alpha_{ram} = -0,05$, $\alpha_{exec} = 0,19$.

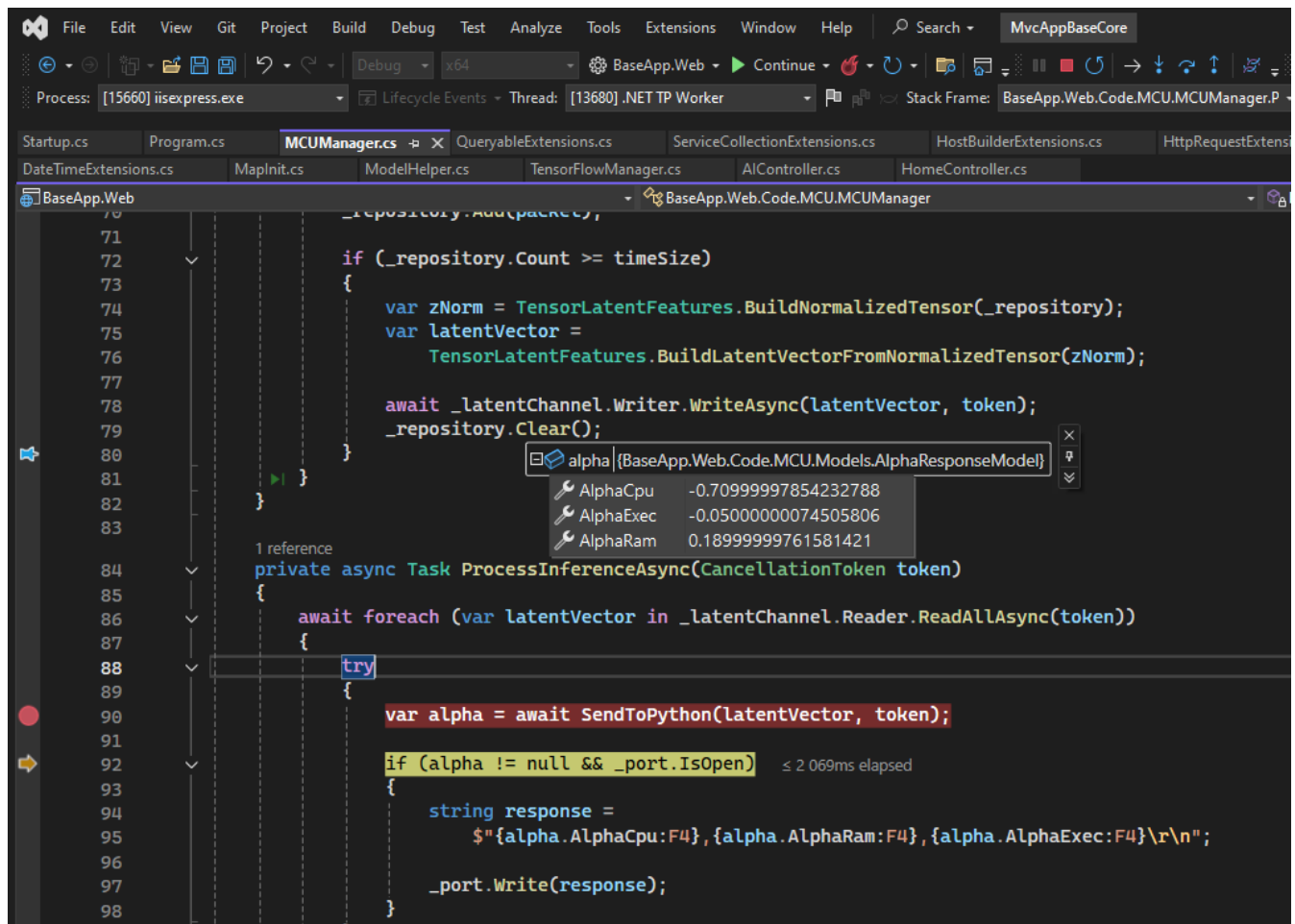


Рисунок 4.4 – Скріншот отримання коефіцієнтів із Tensorflow Lite на сервіс .NET з подальшою відправкою до мікроконтролера.

Отримане співвідношення коефіцієнтів свідчить про те, що у поточному режимі найбільший вплив на ефективність системи має час виконання задач, тоді як

навантаження на процесор і пам'ять не є домінуючими факторами. У такій ситуації планувальник застосовує політики, спрямовані на зменшення тривалості виконання окремих задач або запобігання накопиченню затримок, зокрема шляхом коригування періодів виконання фонових задач або коригування їх пріоритетів.

Отримані коефіцієнти не використовуються ізольовано, а інтерпретуються як узагальнена оцінка поточного стану системи, на основі якої обирається відповідний режим роботи планувальника. Це дозволяє системі адаптивно змінювати поведінку залежно від характеру навантаження, забезпечуючи більш стабільну роботу в умовах динамічних змін параметрів задач. Робота системи логувалась на сервері (рис. 4.5).

Адаптація режимів роботи планувальника у запропонованій архітектурі здійснюється без модифікації базового витісняючого алгоритму FreeRTOS, шляхом динамічного керування параметрами задач через системні механізми ядра. Після отримання від зовнішнього аналітичного модуля коефіцієнтів α_{cpu} , α_{ram} та α_{exec} у модулі `custom_scheduler.c` визначається поточний режим функціонування системи. Аналіз стану може виконуватись у системних хуках, тоді як зміна параметрів реалізується в контексті окремого керуючого компонента. За відсутності домінування будь-якого ресурсу система працює у штатному режимі з базовими пріоритетами та періодами задач. Домінування α_{cpu} інтерпретується як перевантаження процесора, що призводить до зменшення активності фонових задач і забезпечення пріоритетного доступу до обчислювальних ресурсів критичним процесам. Переважання α_{ram} ініціює режим обмеження використання пам'яті. При цьому зменшується динамічне виділення ресурсів та виконується очищення тимчасових структур. Якщо зростає α_{exec} , система знижує безперервну обчислювальну інтенсивність задач із надмірною тривалістю виконання, що зменшує накопичення затримок. Одночасне підвищення α_{cpu} та α_{ram} переводить систему в режим деградації, за якого виконуються лише критичні процеси. Повернення до штатного режиму відбувається поступово, щоб уникнути частих перемикань. У разі нетипового співвідношення коефіцієнтів

активується режим підвищеного моніторингу. Таким чином, адаптація реалізується через керовану зміну параметрів задач і режимів функціонування системи на основі машинного аналізу латентних характеристик навантаження без втручання у внутрішню логіку планувальника.

Спочатку запускається програма з використанням стандартного планувальника задач і зберемо дані про час виконання цих задач, використання ресурсів, пропускну здатність і затримку за допомогою логування.



Рисунок 4.5 – Схема роботи прототипу покращення роботи ОСРЧ на основі класифікації задач у планувальнику за допомогою тензорної декомпозиції і машинного навчання

Процес оновлення вланувальника відбувається один раз в 2 години і автоматично повторюється покращуючи роботу планувальника. Загальна схема роботи алгоритму зображена на рис. 4.5.

Всього було проведено 2 серії експериментів. Формування обчислювальних фаз задач здійснювалося за стохастичною моделлю Бернуллі, у якій поява інтенсивної обчислювальної фази визначається випадковою подією з певною ймовірністю.

У першій серії експериментів ймовірність виникнення ресурсоємної обчислювальної фази становила $p = 0,2$, що відповідає активації інтенсивної обчислювальної ділянки приблизно у 20 % ітерацій виконання задачі, тоді як у більшості випадків виконувалася коротка фаза. За таких умов навантаження на процесор залишалось помірним, а значення коефіцієнтів α_{cpu} , α_{ram} та α_{exec} демонстрували помірне домінування компоненти часу виконання задач. Це інтерпретувалося адаптивним механізмом керування планувальником як ознака перевантаження часових характеристик виконання задач. Порівняння виконувалося на основі часових зрізів телеметрії до застосування політик планування задач та після їх активації. Фрагменти логування роботи зовнішнього аналітичного модуля наведено на рис. 4.6.

Результати збору в першій серії даних класичною FreeRTOS у форматі CSV, відкриті в табличному редакторі, наведено на рис. 4.7. Після цього аналогічно проведеться збір даних у FreeRTOS з оновленим планувальником. Результати збору даних з модифікованої FreeRTOS можна побачити на рис. 4.8. Для статистики кількість задач становила 3 і на кожну задачу по 100 вимірів. Такий обсяг вибірки дозволяє отримати стабільні середні значення метрик і зменшити вплив випадкових коливань навантаження під час роботи планувальника. Результати зберігатимемо у файлах csv. Під час експерименту в системі виконувалася також додаткова задача, що використовувалась для формування фонового навантаження. Вона відображається на часових діаграмах, однак не включалася до складу вибірки експериментальних

вимірювань. Після проведення збору даних результати збережені у 2 файлах `classic_freertos_results` і `tensor_decomposition_results.csv`.

```
-----
2025-06-02 13:31:04.2463 [User: ] [ContentType: ] [: ] [Referer URL: ] [User agent: ] [Server IP: ]
[Build Date: ]
BaseApp.Web.Code.MCU.MCUManager.Port_DataReceived
Receiving Telemetry slice from MCU.
Data:{"Tasks":[{"TaskId":1,"Cpu":18.7275,"Ram":8.48,"Exec":2508.777},{ "TaskId":2,"Cpu":12.3452,"Ram":7.834,
"Exec":2112.956},{ "TaskId":3,"Cpu":13.6175,"Ram":8.148,"Exec":1358.818}]}
-----
2025-06-02 13:32:16.3712 [User: ] [ContentType: ] [: ] [Referer URL: ] [User agent: ] [Server IP: ]
[Build Date: ]
BaseApp.Web.Code.MCU.MCUManager.Port_DataReceived
Receiving Telemetry slice from MCU.
Data:{"Tasks":[{"TaskId":1,"Cpu":16.6106,"Ram":8.499,"Exec":1900.619},{ "TaskId":2,"Cpu":15.5456,"Ram":5.039
,"Exec":2385.525},{ "TaskId":3,"Cpu":17.2475,"Ram":9.274,"Exec":1944.248}]}
-----
2025-06-02 13:33:28.3712 [User: ] [ContentType: ] [: ] [Referer URL: ] [User agent: ] [Server IP: ]
[Build Date: ]
BaseApp.Web.Code.MCU.MCUManager.Port_DataReceived
Receiving Telemetry slice from MCU.
Data:{"Tasks":[{"TaskId":1,"Cpu":13.6275,"Ram":5.565,"Exec":1388.510},{ "TaskId":2,"Cpu":10.6525,"Ram":7.087
,"Exec":2404.810},{ "TaskId":3,"Cpu":17.2375,"Ram":7.991,"Exec":1769.511}]}
.....
-----
2025-06-02 15:29:52.4463 [User: ] [ContentType: ] [: ] [Referer URL: ] [User agent: ] [Server IP: ]
[Build Date: ]
BaseApp.Web.Code.MCU.MCUManager.ProcessTelemetryAsync
Z Normalization processing.....
-----
2025-06-02 15:29:52.4767 [User: ] [ContentType: ] [: ] [Referer URL: ] [User agent: ] [Server IP: ]
[Build Date: ]
BaseApp.Web.Code.MCU.MCUManager.ProcessTelemetryAsync
Tensor decomposition porcessing...
-----
2025-06-02 15:30:02.6832 [User: ] [ContentType: ] [: ] [Referer URL: ] [User agent: ] [Server IP: ]
[Build Date: ]
BaseApp.Web.Code.MCU.MCUManager.ProcessInferenceAsync
Sendind koefecients to MCU: -0.709999997854232788;-0.050000000074505806;0.189999999761581421
-----
2025-06-02 15:31:04.7579 [User: ] [ContentType: ] [: ] [Referer URL: ] [User agent: ] [Server IP: ]
[Build Date: ]
BaseApp.Web.Code.MCU.MCUManager.ProcessInferenceAsync
Sending to TFLite latentVector. Latent Vector:17.951841, 17.466564, 16.513556, 18.331841, 17.036564,
16.552456, 18.120833, 17.681304, 16.291191, 0.1274, 0.5601, -0.0579, 0.2270, -0.4841, -0.2177, 0.537,
0.014, -0.184
-----
2025-06-02 15:32:16.3689 [User: ] [ContentType: ] [: ] [Referer URL: ] [User agent: ] [Server IP: ]
[Build Date: ]
BaseApp.Web.Code.MCU.MCUManager.Port_DataReceived
Receiving Telemetry slice from MCU.
Data:{"Tasks":[{"TaskId":1,"Cpu":12.8625,"Ram":9.107,"Exec":1836.428},{ "TaskId":2,"Cpu":19.0515,"Ram":7.370
,"Exec":2394.411},{ "TaskId":3,"Cpu":16.0257,"Ram":5.938,"Exec":2179.707}]}
-----
2025-06-02 15:31:28.3581 [User: ] [ContentType: ] [: ] [Referer URL: ] [User agent: ] [Server IP: ]
[Build Date: ]
```

Рисунок 4.6 – Фрагмент логів сервера: прийом телеметрії з мікроконтрлера, тензорна декомпозиція та результат інференсу

1	Task Name	CPU Usage (%)	Memory Usage (KB)	Execution Time (ms)	Time (s)			
2	Task 1	18.7275	8.480	2508.777	12			
3	Task 1	16.6106	8.499	1900.619	24			
4	Task 1	13.6275	5.565	1388.510	36			
5	Task 1	19.5728	6.949	1919.808	48			
6	Task 1	12.1325	6.700	2480.797	60			
7	Task 1	14.9058	5.639	2360.202	72			
8	Task 1	18.7349	9.311	1781.745	84			
9	Task 1	14.4725	8.970	1949.500	96			
10	Task 1	15.3317	5.519	2069.230	108			
11	Task 1	12.7753	6.017	1345.737	120			
12	Task 1	12.7675	7.724	1956.517	132			
13	Task 1	15.5479	6.109	1748.514	144			
14	Task 1	18.0925	6.091	2147.873	156			
15	Task 1	18.9375	8.278	2331.349	168			
16	Task 1	15.535	7.161	2380.352	180			
17	Task 1	11.0675	9.256	2121.642	192			
18	Task 1	15.1125	8.222	1712.740	204			
19	Task 1	10.8575	5.851	1949.561	216			
20	Task 1	15.5412	6.405	1967.029	228			

Рисунок 4.7 – Фрагмент результатів збору телеметричних даних під час роботи класичного планувальника FreeRTOS.

1	Task Name	CPU Usage (%)	Memory Usage (KB)	Execution Time (ms)	Time (s)			
2	Task 1	19.7727	8.228	1940.8174	12			
3	Task 1	12.194	6.876	2278.0923	24			
4	Task 1	11.9146	7.965	2018.5328	36			
5	Task 1	15.431	5.512	2335.5049	48			
6	Task 1	14.0737	6.116	1465.5006	60			
7	Task 1	14.5152	8.049	1662.3083	72			
8	Task 1	15.3711	6.61	2271.7192	84			
9	Task 1	13.0207	6.183	2104.1252	96			
10	Task 1	17.2714	7.699	1941.8702	108			
11	Task 1	14.9907	7.755	1406.2644	120			
12	Task 1	17.3334	8.698	1349.4348	132			
13	Task 1	13.6866	5.698	1683.8522	144			
14	Task 1	12.9592	5.945	1502.6476	156			
15	Task 1	10.3391	8.189	1875.7159	168			
16	Task 1	15.1452	6.746	1542.4642	180			
17	Task 1	12.8974	7.599	1286.439	192			
18	Task 1	12.2311	7.324	1668.6062	204			
19	Task 1	14.7795	7.044	1896.395	216			
20	Task 1	13.6505	6.773	1137.5313	228			

Рисунок 4.8 – Фрагмент результатів збору телеметричних даних під час роботи модифікованого планувальника FreeRTOS на основі методу тензорної декомпозиції.

За результатами експериментальних вимірювань в першій серії середнє значення часу виконання задач було приблизно 1943,62 мс після оновлення роботи

планувальника згідно оновлених коефіцієнтів ($\alpha_{cpu} = -0,71$, $\alpha_{ram} = -0,05$, $\alpha_{exec} = 0,19$) середнє значення часу виконання задач стало 1732,5 мс.

У другій серії експериментів імовірність виникнення інтенсивної обчислювальної фази за моделлю Бернуллі було збільшено до $p = 0,6$, а також підвищено тривалість самої обчислювальної ділянки коду. Унаслідок цього значно зросла частота появи ресурсоемних обчислювальних фрагментів задач. Така зміна параметрів стохастичної моделі призвела до помітного збільшення середнього навантаження на процесор, оскільки задачі частіше виконували тривалі обчислювальні цикли без переходу до коротких фаз очікування.

За результатами експериментальних вимірювань в другій серії середнє значення використання процесора збільшилося приблизно з 19 % до 24–25 %, що свідчить про суттєве зростання інтенсивності використання процесорних ресурсів. При цьому використання оперативної пам'яті залишалося практично незмінним, а середній час виконання задач зріс незначно. Така структура телеметрії свідчить про формування домінування процесорного навантаження у системі.

Відповідно до реалізованого алгоритму керування планувальником, домінування коефіцієнта α_{cpu} при перевищенні порогового рівня інтерпретувалося як ознака процесорного перевантаження системи. У цьому випадку активується режим CPU-перевантаження, у межах якого для кожної задачі обчислюється інтегральний показник, що враховує її використання системних ресурсів. На основі цього показника для некритичних задач із високим рівнем споживання процесорного часу знижується пріоритет виконання, тоді як критичним задачам встановлюється підвищений пріоритет. Такий підхід дозволяє зменшити конкуренцію між задачами за процесорний час, обмежити вплив ресурсоемних фонових обчислень та забезпечити стабільне виконання критичних функцій системи. У результаті зменшується ймовірність одночасного виконання інтенсивних обчислювальних фаз різних задач і досягається більш рівномірне використання процесорних ресурсів.

Отже згідно схеми роботи прототипу FreeRTOS здійснює управління процесорним часом за допомогою планувальника задач із фіксованими пріоритетами, де кожній задачі призначається статичний пріоритет $uxPriority$. У кожен момент часу до виконання допускається готова задача з найвищим пріоритетом, а перемикання контексту відбувається у разі появи задачі з вищим пріоритетом або під час обробки системного таймера. У роботі виконано порівняльний аналіз стандартного механізму планування FreeRTOS та модифікованого планувальника задач, побудованого на основі тензорної декомпозиції та методів машинного навчання.

Для порівняння ефективності стандартного планувальника FreeRTOS і FreeRTOS з модифікованою роботою планувальника на основі тензорної декомпозиції використаємо кілька ключових метрик:

1. Тип задачі. Це буде процес обробки даних, введення/виведення, обчислювальна задача або проста фоновіа задача.

2. Використання ресурсів. Вимірювання використання CPU під час виконання задачі.

3. Час виконання задачі. Вимірювання часу, який потрібен для виконання задачі. При першій серії експериментів маємо такі показники. Обчисливши середнє значення отримуємо такі результати.

1. Використання CPU (%). Середнє значення завантаженості процесора зменшилося з 15,75 % до 15,13 %, що на 3,94 % менше у відносному значенні.

2. Використання пам'яті (КБ). Виділення пам'яті збільшилось з 7,00 КВ до 7,07 КВ, що у відносних показниках становить 0,87 %;

3. Виконання часу задач (мс). Середнє значення часу виконання задач було 1943,62 мс до оновлення роботи планувальника, середнє значення часу виконання задач стало 1732,5 мс. Це на 10,86 % менше порівняно із значеннями до оновлення роботи планувальника; скорочення середнього часу виконання на 10,86 % досягнуто за рахунок зменшення затримок, пов'язаних із піковими перекриттями навантаження

та частими перемиканнями контексту, що забезпечило більш стабільне та прогнозоване виконання задач у часі.

При другій серії експериментів маємо такі показники. Обчисливши середнє значення отримаємо такі результати.

1. Використання CPU (%). Середнє значення завантаженості процесора зменшилося з 24,67 % до 21,44 %, що на 13,09 % менше; зменшення завантаження процесора на 13,08 % досягнуто в умовах стохастичного формування навантаження задач за моделлю Бернуллі, яка є дискретним спрощенням класичної пуассонівської моделі потоків подій, що створювало пікові накладання інтенсивних фаз обчислень, при цьому модифікований планувальник забезпечив більш рівномірний розподіл цих фаз у часі, зниження ймовірності одночасного виконання ресурсоємних фрагментів та зменшення накладних витрат перемикання контексту при незмінному наборі та алгоритмічній складності задач.

2. Використання пам'яті (КБ). Виділення пам'яті збільшилось з 7,77 КВ до 7,81 КВ, що у відносних показниках становить близько 0,51 %.

3. Виконання часу задач (мс). Середнє значення часу виконання задач було 2203,36 мс до оновлення роботи планувальника, середнє значення часу виконання задач стало 2123,90 мс. Це приблизно на 3,61 % менше порівняно із значеннями до оновлення роботи планувальника.

Тепер знайдемо інтегральний показник ефективності запропонованого методу. Він обчислюється як середньозважене покращення за трьома ключовими метриками (навантаження CPU, використання пам'яті та час виконання задач).

Перша серія експериментів: $I_1 = w_{CPU}E_{CPU} + w_{RAM}E_{RAM} + w_{TIME}E_{TIME}$, де w_{CPU} – ваговий коефіцієнт використання процесора, E_{CPU} – середнє значення використання процесора, w_{RAM} – ваговий коефіцієнт використання процесора, E_{RAM} – середнє значення використання пам'яті, w_{TIME} – ваговий коефіцієнт виконання часу задач, E_{TIME} – середнє значення виконання часу задач. $w_{CPU} + w_{RAM} + w_{TIME} = 1$. З огляду на відсутність апіорної інформації щодо домінуючої важливості окремих

метрик та з метою отримання узагальненої оцінки ефективності, у роботі прийнято рівнозначні вагові коефіцієнти для всіх показників. Для рівнозначної важливості беремо $w_{CPU} = w_{RAM} = w_{TIME} = \frac{1}{3}$. $I_1 = \frac{1}{3}(E_{CPU} + E_{RAM} + E_{TIME}) = \frac{1}{3}(3,94 - 0,87 + 10,86) = 4,64 \%$.

Друга серія експериментів: $I_2 = \frac{1}{3}(E_{CPU} + E_{RAM} + E_{TIME}) = \frac{1}{3}(13,09 - 0,51 + 3,61) = 5,74 \%$.

Загальний інтегральний показник двох серій: $I = \frac{I_1 + I_2}{2} = \frac{4,64 + 5,74}{2} = 5,19 \%$.

Також підключимо програму DSView, стандартно працююча з логічним аналізатором від DS Logic. За допомогою логічного аналізатора DSLogic Pro у програмі DSView v1.1.2 побудуємо графік, який відображає часові діаграми виконання задач у системі реального часу, де кожен канал відповідає окремому процесу або сигналу. Отримані часові діаграми відображають періоди активності кожної задачі, дозволяючи візуалізувати розподіл процесорного часу та оцінити ефективність роботи планувальника. Виходячи із результатів роботи звичайного планувальника задач FreeRTOS і модифікованого (рис. 4.9) встановлено наступні результати. У звичайному планувальнику задачі 1, 3, і 4 мають рівнозначний пріоритет, що видно з їх рівномірного розподілу на графіку. Вони виконуються по чергово з рівномірним розподілом процесорного часу, що проявляється у вигляді коротких імпульсів активності, які відповідають наданню задачі кванта процесорного часу з подальшим перемиканням контексту. Задача 2 має високий пріоритет і виконується першою після кожного переривання, яке генерується системним таймером, що відповідає стандартній витісняючій політиці планування. Процес простою (Idle) активується нечасто, оскільки процесор майже постійно зайнятий виконанням задач, що свідчить про високу щільність завантаження процесора та значну кількість перемикань між задачами.

Модифікований планувальник має пріоритетний розподіл задач. На графіку (рис. 4.10) зображено розподіл задач ним з урахуванням їхніх пріоритетів і тривалості

виконання. При цьому активні фази задач стають більш згрупованими та менш роздібненими у часі. Тривалість безперервного виконання окремих задач збільшується, кількість коротких імпульсів активності зменшується, а процес простою активується частіше, що свідчить про більш рівномірний розподіл навантаження у часі. Зменшення роздібненості виконання призводить до скорочення частоти перемикань між задачами та зниження пікових накладань інтенсивних фаз обчислень, що забезпечує більш стабільний режим функціонування системи.

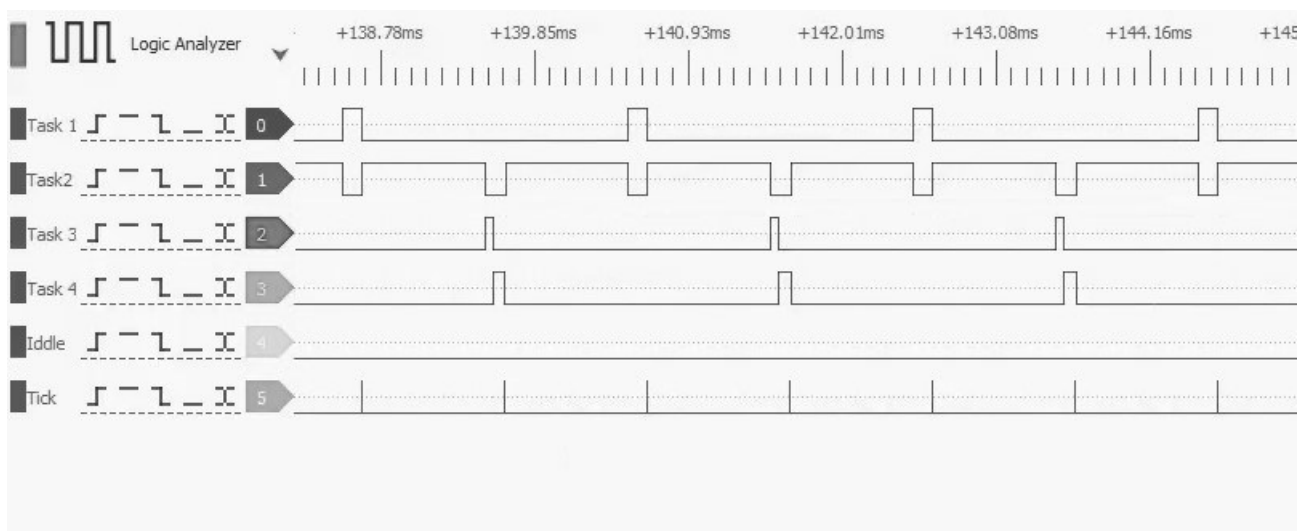


Рисунок 4.9 – Графічне представлення фрагменту часу виконання задач стандартним планувальником FreeRTOS

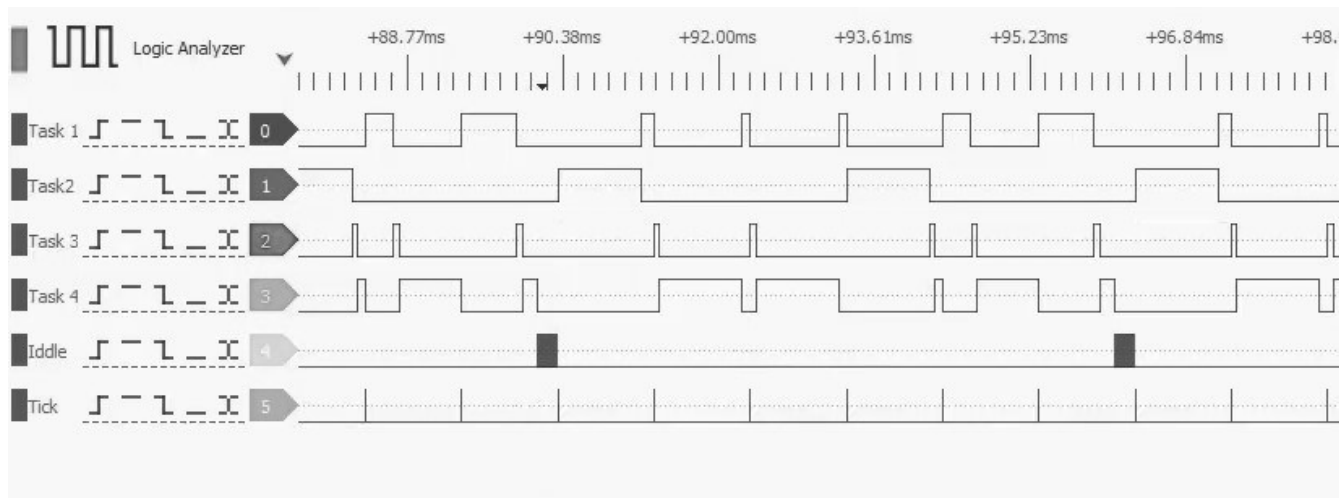


Рисунок 4.10 – Графічне представлення фрагменту часу виконання задач модифікованим планувальником FreeRTOS

Ці результати підтверджують, що система з тензорною декомпозицією досягає заданих покращень у використанні процесора, пам'яті та часу виконання задач у порівнянні з класичною FreeRTOS. Також встановлено, що очікування результатів від впровадження кастомного планувальника задач у FreeRTOS з використанням TensorFlow Lite залежать від інших кількох факторів, включаючи складність задач, характер навантаження та ефективності реалізації моделі машинного навчання.

4.2 Прототип системи автономного моніторингу та результати експериментальних досліджень

Для перевірки дієвості двоетапного підходу автономного моніторингу (АСТ і програмний шар із марковським аналізом) було проведено розгорнуті експерименти на базі мікроконтролерної платформи STM32F407G-DISC1 (ядро Cortex-M4, тактова частота 180 МГц), де використовується ОСРЧ FreeRTOS із п'ятьма задачами прикладного рівня. Додатково, у складі експериментального середовища використовувався зовнішній програмний модуль збору та аналізу телеметрії, який функціонував як незалежна від мікроконтролера підсистема моніторингу. Дані передавалися з STM32F407G-DISC1 до персонального комп'ютера через інтерфейс USB CDC, що дозволяло фіксувати часові параметри перемикавання задач, затримки, навантаження процесора, події сервісу логування та стани програмного предиктивного сторожового таймера. Використання зовнішнього модуля виключало вплив логування на часові характеристики FreeRTOS і забезпечувало можливість побудови емпіричної матриці переходів між станами Normal, Warning, Critical і Failure для подальшого аналізу.

Серед цих задач найголовнішими були:

- один модуль (TaskA), що інтенсивно працює з периферією (SPI/UART) і може періодично навантажувати систему;
- допоміжні задачі (TaskB, TaskC), які виконували обробку сигналів і взаємодію з датчиками;

- сервіс логування, призначений для запису діагностичних повідомлень і часу подій;

- ПЗСК, який реалізував програмний алгоритм прогнозного сторожового контролю на основі низькорозмірної марковської моделі з періодом опитування 100 мс.

АСТ мав таймаут 2 секунди. Програмна задача ПЗСК у кожному циклі перевіряє, чи вчасно надходять сигнали про активність від інших задач, а також заміряє рівень завантаженості черг. На основі цих параметрів (включно з історією переходів за останні 50 спостережень) будувалася актуальна матриця ймовірностей P марковського ланцюга зі станами N (Normal), W (Warning), C (Critical) та F (Fail). Якщо модель вказувала на високу ймовірність переходу в Fail (понад заданий поріг), ПЗСК виконувало превентивний перезапуск проблемного компонента (видалення й повторне створення певної задачі, перевантаження драйвера тощо). Водночас, якщо локальні спроби відновлення не давали результату, програмна задача свідомо не передавав керуючий сигнал апаратному сторожовому таймеру, що призводило до його спрацювання та повного перезапуску системи через 2,8 с.

Задля уніфікації експериментів були визначені спеціальні сценарії збоїв, коли TaskA штучно блокувався, спричиняв зависання, або генерував надмірні обсяги даних, що спричиняло зростання черг понад 80 %, або затримував сигнали про активність. Перевіряли, як швидко ПЗСК виявить критичний стан, чи встигне виконати програмне скидання, та чи буде взагалі потрібно апаратне скидання. Часові мітки фіксувалися через логування, а також через механізми відстеження подій у FreeRTOS+Trace. Для кожного сценарію повторювали серію запусків більше 10 разів і вираховували середні значення основних показників.

Першим ключовим показником був час виявлення початку збоїв. Здійснено вимірювання кількості часу, який проходить від моменту, коли система розпочинала поводитися аномально. Наприклад, переставав надходити сигнал про активність від TaskA, і до моменту, коли ПЗСК переходив у стан C (Critical) і підвищував ймовірність

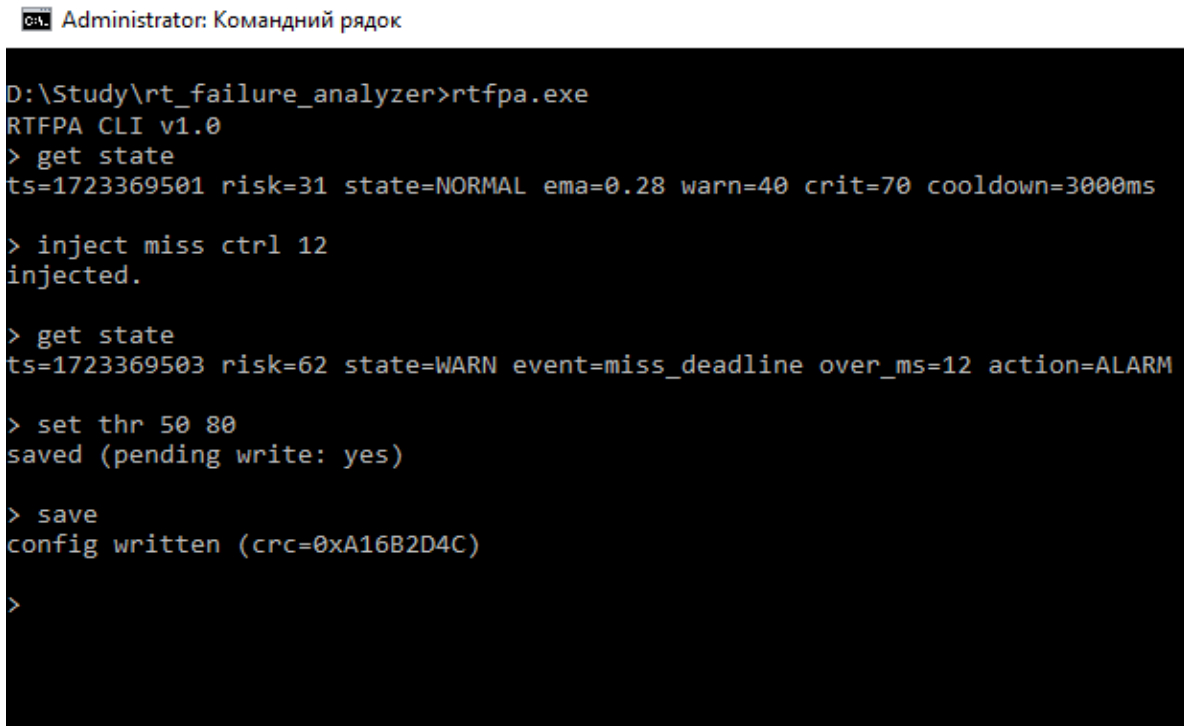
F (Fail), запускаючи програмне скидання. Паралельно було встановлено час фактичного відновлення, тобто скільки спливає від підтвердження зависання чи критичної ситуації до повернення системи у N (Normal). Здійснено порівняння часу, який потрібен на програмний перезапуск одної задачі з часом повного перезавантаження при спрацюванні АСТ.

Окремо було встановлено значення коефіцієнту глобальних перезапусків, тобто відсоток випадків, де глобальне скидання усе-таки виявлялося необхідним. Було важливо встановити, чи не стає ПЗСК надто оптимістичним і не пропускає глибоких збоїв, або, навпаки, не породжує безліч зайвих програмних скидань там, де система могла б відновитися сама.

Крім того, за допомогою профілювання FreeRTOS через інтегровані засоби трасування відстежувалось середнє додаткове обчислювальне навантаження на процесор через актуалізацію марковської моделі, аби переконатися, що ПЗСК не впливає негативно на решту задач.

Для забезпечення кількісної оцінки роботи системи під час проведення експериментів використовувався зовнішній програмний модуль збору та аналізу телеметрії, реалізований у вигляді аналізатора ймовірності відмов у реальному часі. Модуль здійснював безперервний збір метрик виконання задач, реєстрацію подій і передачу даних на зовнішній вузол для подальшого аналізу. Приклад роботи аналізатора в консолі зображено на рис. 4.11 і рис. 4.12.

На рис. 4.11 зображено сесію CLI програмного модуля, а на рис. 4.12 це приклад виводу телеметричних метрик, що містять числові показники ризику відмови та лічильники подій, які використовуються для кількісної оцінки стану системи і подальшого статистичного аналізу.



```

Administrator: Командний рядок

D:\Study\rt_failure_analyzer>rtfpa.exe
RTFPA CLI v1.0
> get state
ts=1723369501 risk=31 state=NORMAL ema=0.28 warn=40 crit=70 cooldown=3000ms

> inject miss ctrl 12
injected.

> get state
ts=1723369503 risk=62 state=WARN event=miss_deadline over_ms=12 action=ALARM

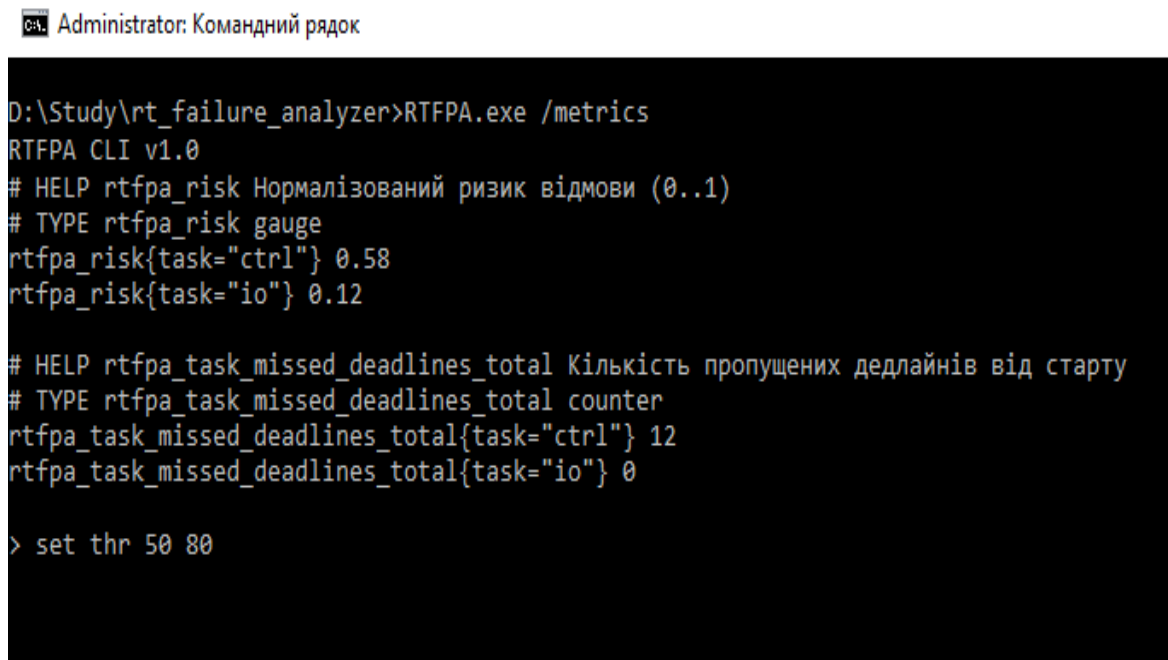
> set thr 50 80
saved (pending write: yes)

> save
config written (crc=0xA16B2D4C)

>

```

Рисунок 4.11 – Приклад роботи аналізатора ймовірності відмов у реальному часі (АІВРЧ). Сесія CLI



```

Administrator: Командний рядок

D:\Study\rt_failure_analyzer>RTFPA.exe /metrics
RTFPA CLI v1.0
# HELP rtfpa_risk Нормалізований ризик відмови (0..1)
# TYPE rtfpa_risk gauge
rtfpa_risk{task="ctrl"} 0.58
rtfpa_risk{task="io"} 0.12

# HELP rtfpa_task_missed_deadlines_total Кількість пропущених дедлайнів від старту
# TYPE rtfpa_task_missed_deadlines_total counter
rtfpa_task_missed_deadlines_total{task="ctrl"} 12
rtfpa_task_missed_deadlines_total{task="io"} 0

> set thr 50 80

```

Рисунок 4.12 - Приклад роботи аналізатора ймовірності відмов у реальному часі (АІВРЧ). Вивід метрик у форматі Prometheus

Дані передавались із мікроконтролера STM32F407G-DISC1 через інтерфейс USB CDC у реальному часі, що дозволило контролювати часові характеристики задач, події перемикання контексту, навантаження процесора та зміни індексу стану програмного предиктивного сторожового таймера. Повний вихідний код програмного модуля наведено у Додатку Е. Модуль автоматично реєстрував переходи між станами Normal, Warning, Critical та Failure, на основі чого обчислювалися емпіричні частоти переходів та формувалась марковська матриця, необхідна для оцінювання ймовірностей критичних ситуацій і виявлення аномальних сценаріїв. Отримані дані використовувалися для порівняльного аналізу поведінки системи при застосуванні традиційного АСТ та удосконаленого двоетапного підходу з предиктивним контролем.

У першій серії випробувань (рис. 4.13, логуювання зліва) з імітацією взаємного блокування у TaskA, коли у 5 спробах ПЗСК свідомо не передавав керуючий сигнал апаратному сторожовому таймеру, що призводило до його спрацювання та повного перезапуску системи. Таким чином коефіцієнт глобальних перезапусків становив $S_{\text{ПЗСК}} = \frac{100}{5} \times 1 = 0,2$. За чисто реактивного підходу, коли програмний сторожовий таймер був відсутній, система реагувала лише після повного зависання – у 100% випадків скидання відбувалося виключно через АСТ.

Сумарний простій:

$$T_{\text{АСТ}} = 5 \times 2,7 = 13,5 \text{ с}$$

Відсоток простою за 5 хвилин (300 с):

$$S_{\text{АСТ}} = \frac{13,5}{300} \times 100\% = 4,5\%$$

Обчислимо коефіцієнт готовності:

$$R_{\text{АСТ}} = 100\% - 4,5\% = 95,5\%$$

Знайдемо середній час відновлення. У першій серії експериментів було зафіксовано 5 глобальних відновлень із часом 2,7 с. Тоді:

$$MTTR_{ACT} = \frac{5 \cdot 2,7}{5} = 2,7 \text{ с.}$$

simple_watchdog.log	predictive_watchdog.log
<pre> 1 [0.000] [LOGGER] System boot (HW WDT only) 2 [0.010] [TaskA] Init complete 3 [0.020] [TaskB] Init complete 4 [0.030] [TaskC] Init complete 5 [0.050] [LOGGER] State: NORMAL, Queue usage=0% 6 [0.096] [TaskA] Heart-beat OK 7 [0.152] [TaskB] Sensor reading: 310 8 ... 9 [36.237] [LOGGER] << CRITICAL >> No response, queue usage=95% 10 [36.307] [HW WDT] Triggering HARD RESET 11 --- DOWNTIME from 36.307s to 38.996s --- 12 [38.996] [LOGGER] Reboot after WDT reset 13 [39.268] [TaskA] Init complete 14 [39.686] [TaskB] Init complete 15 [40.453] [TaskC] Init complete 16 [40.280] [LOGGER] State: NORMAL, Queue usage=0% 17 [41.555] [TaskA] Heart-beat OK 18 ... 19 [59.529] [LOGGER] << CRITICAL >> No response, queue usage=88% 20 [59.641] [HW WDT] Triggering HARD RESET 21 --- DOWNTIME from 59.529s to 62.307s --- 22 [62.307] [LOGGER] Reboot after WDT reset 23 [62.936] [TaskA] Init done 24 [62.946] [TaskB] Init done 25 [62.949] [TaskC] Init done 26 [63.030] [LOGGER] State: NORMAL, Queue usage=0% 27 ... 28 [102.732] [LOGGER] << CRITICAL >> No response, queue usage=93% 29 [102.854] [HW WDT] Triggering HARD RESET 30 --- DOWNTIME from 102.854s to 104.805s --- 31 [104.805] [LOGGER] System reboot 32 [105.520] [TaskA] Init done 33 [105.847] [TaskB] Init done 34 [105.534] [TaskC] Init done 35 [107.428] [LOGGER] State: NORMAL, Queue usage=0% 36 ... 37 [198.940] [LOGGER] << CRITICAL >> No response, queue usage=90% 38 [198.951] [HW WDT] Triggering HARD RESET 39 --- DOWNTIME from 198.951s to 201.931s --- 40 [201.931] [LOGGER] System reboot 41 [201.985] [TaskA] Init done 42 [196.930] [TaskB] Init done 43 [196.960] [TaskC] Init done 44 [200.302] [LOGGER] State: NORMAL, Queue usage=0% 45 ... 46 [220.170] [LOGGER] << CRITICAL >> No response, queue usage=85% 47 [222.675] [HW WDT] Triggering HARD RESET 48 --- DOWNTIME from 222.675s to 225.383s (~2.7s) --- 49 [225.383] [LOGGER] System reboot 50 [225.680] [TaskA] Init done 51 [225.708] [TaskB] Init done 52 [225.722] [TaskC] Init done 53 [226.558] [LOGGER] State: NORMAL, Queue usage=0% 54 ... 55 [LOGGER] *** End of 5-min run *** 56 [LOGGER] Hard Resets=5(36,59,102,198,227,222) => 2.7s 57 [LOGGER] Total downtime=5*2.7=13.5s (~4.5%) </pre>	<pre> 1 [0.000] [LOGGER] System boot (HW WDT + PredictiveWDG) 2 [0.010] [TaskA] Init done 3 [0.020] [TaskB] Init done 4 [0.029] [TaskC] Init done 5 [0.051] [LOGGER] Markov state = N, P(N)=1.0, Q usage=0% 6 [0.095] [TaskA] Heart-beat OK 7 [0.202] [TaskC] Processing cycle, Q usage=5% 8 [0.507] [TaskC] Done, CPU=25% 9 ... 10 [53.617] [LOGGER] Queue usage=80%, State update => CRITICAL (P(C)>0.7) 11 [58.259] [PredictiveWDG] SoftReset(TaskA) 12 --- partial downtime from 58.259s to 58.611s (~0.35s) --- 13 [58.259] [TaskA] Re-initialized 14 [58.612] [LOGGER] State => N, queue usage=10% 15 ... 16 [66.366] [LOGGER] Queue usage=85%, State update => CRITICAL (P(C)>0.72) 17 [66.466] [PredictiveWDG] SoftReset(TaskA) 18 --- partial downtime 66.466s to 66.817s (~0.35s) --- 19 [67.477] [TaskA] Re-initialized, drivers reinit 20 [67.570] [LOGGER] State => NORMAL, queue usage=5% 21 ... 22 [90.448] [TaskB] Sensor reading=999 23 [90.510] [LOGGER] Queue usage=88%, State update => CRITICAL (P(C)>0.72) 24 [90.609] [PredictiveWDG] SoftReset(TaskB) // this time we reset TaskB 25 --- partial downtime 90.609 to 90.951 (~0.35s) --- 26 [90.951] [TaskB] Re-initialized 27 [90.961] [LOGGER] State => N, queue usage=10% 28 ... 29 [91.448] [TaskA] Sensor reading=1002 30 [93.758] [LOGGER] Queue usage=82%, State update => CRITICAL (P(C)>0.71) 31 [95.075] [PredictiveWDG] SoftReset(TaskA) 32 --- downtime 95.075 to 95.421 (~0.35s) --- 33 [95.175] [TaskA] Re-initialized 34 [95.270] [LOGGER] Markov => N, queue usage=8% 35 ... 36 [210.882] [LOGGER] Queue usage=95, State update => CRITICAL (P(C)>0.78) 37 [215.999] [PredictiveWDG] CRITICAL lock-up, SoftReset fails 38 [227.121] [HW WDT] Triggering HARD RESET 39 --- downtime from 227.121 to 229.833 (~2.7s) --- 40 [227.241] [LOGGER] System reboot 41 [229.833] [TaskA] Init done 42 [229.859] [TaskB] Init done 43 [229.861] [TaskC] Init done 44 [230.476] [LOGGER] State => N, queue usage=0% 45 [230.671] [TaskA] Heart-beat OK 46 ... 47 [LOGGER] *** End of 5-min run *** 48 [LOGGER] Soft Resets=4 (58.259,66.466,90.609,95.075) ~0.35s each => 1.4s total 49 [LOGGER] Hard Resets=1 (227.121) => 2.7s 50 [LOGGER] Total downtime=4.2s (~1.40%) </pre>
Normal text file	Normal text file

Рисунок 4.13 - Логування системи з (зліва) та без (справа) ПЗСК, 5 хвилин з випадковими збоями

У другій серії випробувань (рис. 4.13, логування справа) відбувалося, де TaskA зависає з імітацією перевантаження периферії SPI, система з ПЗСК виявляла критичний стан у середньому за 0,11–0,13 с, бо період опитування становив 100 мс, і зазвичай вистачало 1–2 циклів, аби змінити оцінку стану від Normal/Warning до Critical. Після переходу в Critical з високою ймовірністю переходу в F (Fail), зокрема, $PC \rightarrow F > 0,7$ ПЗСК виконував програмне скидання TaskA, перевантажуючи пов'язані драйвери. Така операція займала близько 0,3–0,4 с, тож загальний час повернення в Normal не перевищував 0,5 с. Для порівняння, повне спрацювання АСТ

зумовлювало очікування в 2 секунди (через таймаут) плюс 0,6–0,8 с на ініціалізацію всієї системи, тобто в середньому майже 2.7 секунди простою.

Сумарний простій:

$$T_{\text{ПЗСК}} = (4 \times 0,35) + (1 \times 3) = 1,4 + 2,7 = 4,1 \text{ с.}$$

Відсоток простою за 5 хвилин (300 с):

$$S_{\text{ПЗСК}} = \frac{4,1}{300} \times 100\% \approx 1,36666\%.$$

Обчислимо коефіцієнт готовності:

$$R_{\text{ПЗСК}} = 100\% - 1,36666\% = 98,63333\%$$

Знайдемо середній час відновлення. У другій серії експериментів було зафіксовано 4 локальних відновлення із середнім часом 0,35 с та 1 глобальне відновлення із часом 2,7 с. Тоді:

$$MTTR_{\text{ПЗСК}} = \frac{4 \cdot 0,35 + 1 \cdot 2,7}{5} = 0,82 \text{ с.}$$

Тепер знайдемо зменшення часу простою:

$$t_{\text{abs}} = 4,5\% - 1,36666\% = 4,5\% - 1,36666\% = 3,13334\%.$$

Розрахуємо відносний показник скорочення простою:

$$K_{\text{red}} = \frac{4,5 - 1,36666}{4,5} \times 100\% = 69,62978\%.$$

Щодо навантаження на процесор, найгірший вимір у FreeRTOS+Trace зафіксував додатково 1,6% навантаження, коли TaskA генерував велику кількість подій і ПЗСК одночасно обробляв лічильники переходів у матриці P . У середньому, з урахуванням, що ядро працює на 180 МГц, це навантаження залишалося нижче 1,4%, тобто не впливало помітно на виконання решти задач.

Також додатково розрахуємо коефіцієнт загального скорочення простою:

$$K_{\text{sd}} = \frac{4,5}{1,36666} = 3,2927.$$

Отже розроблений метод забезпечує скорочення простою системи приблизно на 69,63% та у 3,29 рази прискорює відновлення після збоїв порівняно з традиційним

апаратним сторожовим таймером. Загальні висновки з експерименту свідчать, що запропонована стратегія завдяки регулярному м'якому перезапуску проблемних компонентів дає суттєве зменшення середнього часу простою, до 5–8 разів менше, ніж за повного перевантаження. При цьому зберігається гарантований жорсткий метод захисту від фатальних збоїв. Сам час роботи не сильно збільшився в відсотковому співвідношенні. Отже, поєднання легкої стохастичної оцінки ризику через марковську модель та двоетапного програмно-апаратного механізму сприяє помітному підвищенню надійності і швидкості відновлення у вбудованих і КФС, при цьому не вимагаючи значних обчислювальних ресурсів. Результати роботи АСТ разом з і без ПЗСК зображено на рис. 4.14.

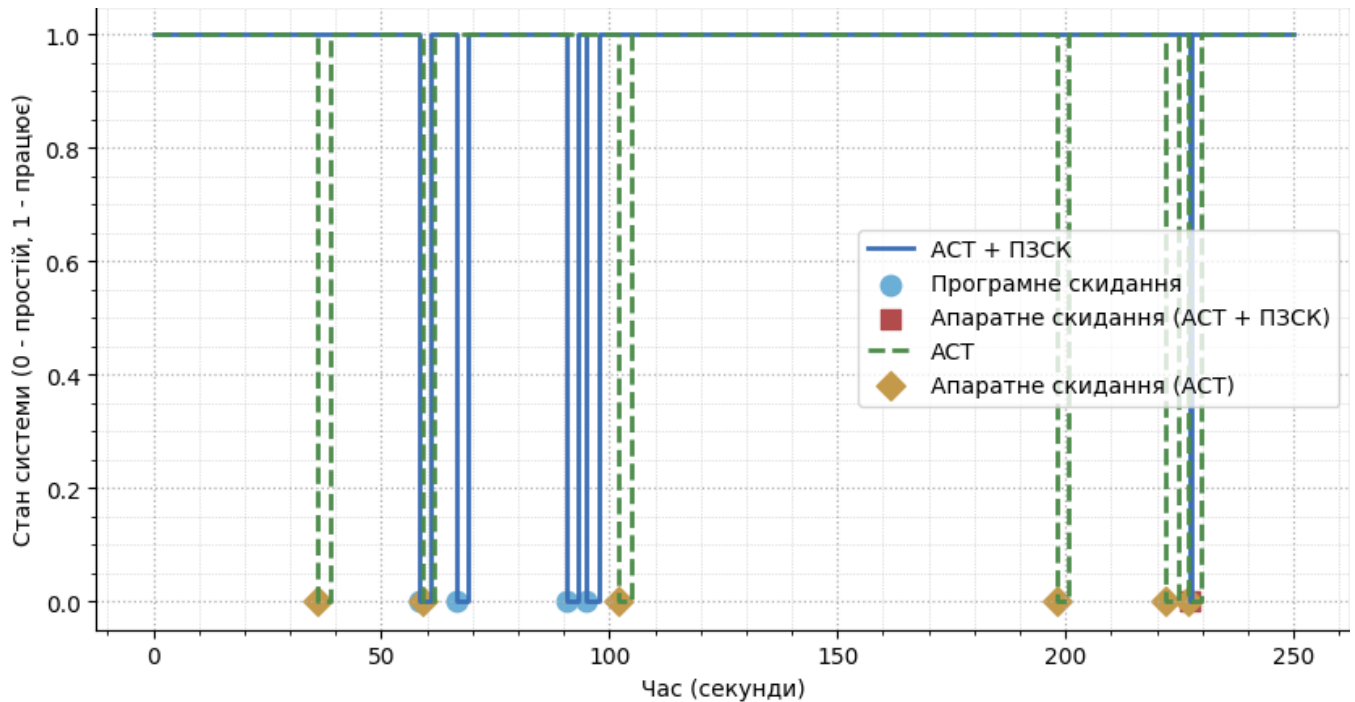


Рисунок 4.14 - Порівняння системи з ПЗСК та без нього (5 хвилин) з випадковими збоями

Спостережуваний позитивний ефект особливо важливий у застосуваннях, де простій системи призводить до критичних наслідків, де є обмеження апаратних ресурсів і можливості швидкого розгортання складних ML або резервних схем. Наприклад, промислова автоматика, робототехніка, транспорт, медичні прилади.

Експериментальні результати підтверджують доцільність впровадження такого методу в будь-яких проєктах, де, з одного боку, необхідна висока надійність, а з іншого – бажано мінімізувати час глобальних перезавантажень.

Проведені експериментальні дослідження підтвердили працездатність запропонованого методу превентивного відновлення компонентів ОСРЧ на основі низькорозмірної марковської моделі прогнозування станів у поєднанні з багаторівневим сторожовим контролем. У результаті його застосування середній час відновлення зменшився з 2,7 с до 0,82 с, а частка простою – з 4,5 % до приблизно 1,37 %, що відповідає відносному зменшенню простою близько 69,6 % при додатковому обчислювальному навантаженні не більше 1,6 %, що не впливає на виконання задач реального часу. Отримані результати підтверджують доцільність використання методу для підвищення надійності та скорочення часу відновлення у вбудованих і кіберфізичних системах

4.3 Експериментальна порівняльна оцінка ефективності комбінованого фільтра Калмана

Для перевірки описаного методу розроблено прототип адаптивного фільтра Калмана, вбудованого в систему на мікроконтролері STM32 під керуванням FreeRTOS. Фільтр призначений для оцінювання стану об'єкта за даними цифрового сенсора, з автоматичним виявленням і обробкою аномальних, фальшивих вимірювань. Реалізація базується на подієвому перемиканні моделей динаміки та контролі інновації, нев'язки вимірювання у режимі реального часу. При перевищенні порогів нев'язки прототип або відсікає некоректні вимірювання, або ініціює зміну моделі стану, що відповідає іншому режиму роботи системи. Нижче описано архітектуру задач FreeRTOS, апаратну та програмну інтеграцію сенсора (I^2C), формат та обробку вхідного сигналу, алгоритми фільтра Калмана й виявлення аномалій, а також реакцію системи на аномалії та перемикання режимів. Наведено деталі реалізації, як-от пріоритети задач, синхронізація, збереження результатів

експерименту через UART/SD та оцінку навантаження на мікроконтролер. Архітектура програмного забезпечення побудована на багатозадачності FreeRTOS, що забезпечує детерміноване планування та швидкий відгук на події в системах реального часу. Задачі розподілено за функціональними ролями і пріоритетами, аби гарантувати своєчасний збір даних, фільтрацію та передачу результатів. Ключові задачі прототипу визначено так.

1. Задача зчитування сенсора. Виконується з високим пріоритетом. Наприклад, пріоритет 3 із 1–3, де 3 найвищий. Її мета полягає в періодичному отриманні даних з цифрового датчика через I^2C . Ця задача запускається при старті системи і циклічно виконується з фіксованим інтервалом, наприклад, 100 Гц. Вона формує подію, що нові дані готові для фільтраційної задачі. Для реалізації періодичності використовується або виклик `vTaskDelayUntil()` на основі системного тика ОСРЧ, або апаратний таймер, що генерує переривання/нотифікацію задачі;

2. Задача фільтрації через фільтр Калмана. Має дещо нижчий пріоритет (пріоритет 2), щоб не перешкоджати критичному збору даних, але достатньо високий для оперативної обробки щойно отриманих вимірювань. Ця задача очікує сигнал (через семафор, чергу повідомлень або пряме сповіщення задачі) від задачі сенсора про надходження нового вимірювання. Отримавши сигнал, задача-фільтр виконує цикл алгоритму Калмана: прогноз стану, корекцію на основі нового вимірювання, якщо воно не відкинута, оновлення оцінок і коваріацій, та аналіз нев'язки вимірювання для можливого виявлення аномалії. У разі виявлення події, наприклад, аномалії або зміни режиму, задача може змінювати внутрішню модель фільтра, перемикає параметри, або ініціювати інші дії, наприклад, сигналізувати про подію;

3. Задача логування результатів. Це низькопріоритетна задача (пріоритет 1), що отримує оброблені дані, а саме оцінені стани, індикатори аномалій тощо, через чергу повідомлень від задачі фільтрації. Її призначення полягає в зберіганні результатів експерименту, зокрема у передаванні даних через інтерфейс UART на комп'ютер або

у записі їх на SD-карту. Низький пріоритет гарантує, що ця задача виконується у фоновому режимі і не затримує критичні цикли збору даних та оцінки стану.

При старті системи в функції `main()` після ініціалізації апаратури створюються згадані задачі з заданими пріоритетами та розмірами стеків. FreeRTOS планувальник гарантує, що при надходженні нового вимірювання високопріоритетна задача збору даних активується негайно, випереджаючи задачі нижчого пріоритету. Отримавши дані, вона сигналізує задачі фільтра. Завдяки механізму пріоритетів, задача фільтрації продовжує виконання відразу після збору даних, якщо була заблокована в очікуванні, або ж виконується після завершення поточного такту сенсора. Під час очікування нових даних блокування на семафорі/нотифікації задача фільтра не споживає CPU. Аналогічно, задача логування працює тільки коли є нові повідомлення в черзі, і може тимчасово поступатися CPU, якщо більш пріоритетні задачі готові до виконання. Така архітектура відповідає рекомендованому дизайну ОСРЧ-систем: збір даних – з найвищим пріоритетом, обробка даних – середнім, передача/відображення – нижчим. Це мінімізує затримку між фізичним вимірюванням і його обробкою, що особливо важливо при інтенсивних змінах стану об'єкта. Для синхронізації задач використано комбінацію примітивів FreeRTOS. Задача сенсора після кожного зчитування виставляє бінарний семафор або здійснює `xTaskNotifyGive` для задачі фільтрації. Остання, відповідно, чекає `xTaskNotifyTake` або `xSemaphoreTake` на цей сигнал кожен цикл. Такий механізм є ефективним і зводить до мінімуму час простою – фільтр запускається негайно після готовності нового вимірювання. Передача даних від фільтра до задачі логування виконується через чергу `xQueueSend` в задачі фільтра та `xQueueReceive` в задачі логування. Черга буферизує результати, дозволяючи задачі фільтра не чекати завершення операцій вводу-виводу. Розмір черги підібрано так, щоб уникнути переповнення навіть при короткочасному призупиненні задачі логування, наприклад, якщо запис на SD-карту зайняв більше часу. Варто зазначити, що доступ до апаратних ресурсів, зокрема, I^2C -інтерфейсу в багатозадачному середовищі потребує узгодження. У нашій архітектурі шина I^2C використовується виключно

задачею сенсора, тому конфліктів не виникає. Якщо ж інші задачі могли б звертатися до цього інтерфейсу, слід було б використовувати взаємовиключення (м'ютекс) для послідовного доступу, аби уникнути одночасних запитів до I^2C з різних задач. Завдяки даній структурі, система є подієво-орієнтованою та ефективною: критичні задачі реагують на події негайно, а фонові обробка не впливає на точність таймінгів збору даних.

Вихідний сигнал сенсора – це цифрові дані, що представляють вимірювану фізичну величину, наприклад, прискорення, кутову швидкість, відстань або іншу змінну. Формат даних зазвичай цілочисельний, типово 16-бітне зі знаком, або у вигляді кількох регістрів. Перед використанням у фільтрі Калмана ці сирі дані потрібно перетворити у фізично осмислену форму і за потреби відфільтрувати артефакти вимірювання. Конвертація та масштабування. Перший етап попередньої обробки – застосувати до сирового значення масштабний коефіцієнт та зсув. Наприклад, для акселерометра 1 один молодший розряд цифрового значення може відповідати 0.001 g . Якщо сенсор має відомий коефіцієнт перерахунку k і нульове зміщення b , то отримане ціле d_{raw} перетворюється на фізичну величину: $x_{\text{means}} = k * d_{raw} + b$. У нашому прикладі b може бути використаний для компенсації статичного зсуву датчика, наприклад, гравітації або нульового дрейфу. Таким чином, значення приводиться до реальних одиниць, наприклад, м/с^2 , градусів/с тощо. Фільтрація шуму сенсора. Іноді перед подачею даних у Калман-фільтр застосовують елементарну фільтрацію, наприклад, ковзне середнє або медіанний фільтр, щоб зменшити високочастотний шум чи поодинокі сплески. У нашому прототипі головним фільтром є Калмана, тому додаткове згладжування не обов'язкове. Однак, якщо сенсор має відоме високе шумове забруднення, можна усереднювати декілька швидких вимірів у задачі сенсора, щоб подати фільтру більш стабільне значення. Це еквівалентно зменшенню дисперсії вимірювань R . Попереднє згладжування слід використовувати обережно, щоб не ввести зайву затримку; в нашому випадку алгоритм Калмана сам оцінює шум і пригнічує його. Перевірка діапазону та маскування помилок. На рівні

попередньої обробки варто також перевірити адекватність отриманого значення. Якщо сенсор видав дані поза фізично можливим діапазоном, наприклад, код переповнення, чи, наприклад, передав повторювано одне й те саме значення, можлива ознака збою, задача сенсора може позначити такі дані як невалідні.

У прототипі реалізовано простий контроль. Якщо значення перевищує очікувані межі, наприклад, акселерометр раптово показав прискорення $100g$, що нереально для даної системи, то вимірювання маркується як підозріле ще до фільтра. Такі грубі відхилення можуть бути відразу відкинуті або оброблені особливим чином, наприклад, ініціювати перехід фільтра у режим високої невизначеності. В нашому випадку при роботі у штатному режимі системи зміни вимірюваної величини очікуються відносно плавні. Інтенсивні ж стрибки сигналу можуть бути або наслідком реальної різкої зміни стану, або артефактом/помилкою. Подальший алгоритм фільтра Калмана з моніторингом інновації визначатиме природу таких стрибків. Проте базова перевірка на рівні сенсора на предмет явного збою підвищує надійність. Завідомо помилкові дані чи фальсифікації не будуть навіть намагатися коригувати оцінку стану. Отримавши та підготувавши вимір z^k у фізичних одиницях, задача сенсора упакує його, разом з міткою часу, якщо потрібно, у структуру даних і передає задачі фільтрації через повідомлення. Формат вимірювання узгоджений з моделлю фільтра. Наприклад, якщо фільтр оцінює стан об'єкта у термінах положення і швидкості, а сенсор дає положення, то z^k – скаляр положення в метрах. Додатково може передаватися прапор, чи вимір валідний. У разі невалідності, якщо присутня сенсорна помилка, то фільтр пропустить крок оновлення.

Експериментальну перевірку запропонованого комбінованого фільтра Калмана виконували на відладочній платі зі мікроконтролером STM32F407G-DISC1, що працювала під керуванням FreeRTOS. Один із апаратних таймерів налаштували на період 10 мс, тобто навігаційна задача виконувалася зі звичною для безпілотної частотою 100 Гц. Оскільки фактичний політ не вимагався, траєкторію висоти задавали скриптом, додаючи до неї нормальний атмосферний шум із дисперсією приблизно

один метр квадратний, а самі пакети надсилалися у чергу FreeRTOS так, ніби їх прислав барометр. Передбачалося два режими роботи безпілотного апарата. У стабільному режимі модель тримала висоту з незначними коливаннями, а у маневровому режимі імітувалася швидка зміна рівня приблизно на п'ять метрів за два-три такти, що відтворювало типову команду на зниження чи набір висоти. Атаки реалізовувалися як навмисна підміна барометричних пакетів. Спочатку вводили одноразові значення зі зсувом $\pm 300\text{--}500$ м і тривалістю один цикл, а далі перевіряли короткі серії з тих самих амплітуд, але вже протягом тридцяти-п'ятдесяти мілісекунд.

На STM32F407G-DISC1 було реалізовано чотири фільтри для порівняння мовою С у вигляді окремих програмних модулів у складі FreeRTOS на основі стандартних рівнянь фільтра Калмана та їх відомих робастних і адаптивних модифікацій.

Перший – класичний фільтр Калмана без адаптації, з фіксованими Q і R . Для нього окремо реалізовано детектор аномалій на резидуалі з фіксованим порогом за правилом 3σ [133]. Тобто, якщо $r^k > T$, генерується тривога. Це відображає типову схему – фільтр для згладжування і пороговий детектор. ККФ реалізовано згідно з описаним вище алгоритмом, з інтегрованим модулем подієво-резидуальної адаптації та дискретною логікою перемикавання режимів. Параметри обрано наступні: поріг $S_1 = 3\sigma_{\text{шум}} (\approx 5 \text{ м})$, $T_2 = 4\sigma_{\text{шум}} (\approx 10 \text{ м})$, $N = 1$. Q_0 відповідає повільній динаміці рівня, варіація 0.25 м^2 за цикл, Q_{max} встановлено 25 м^2 , що дозволяє миттєвий стрибок рівня до $\sim 5 \text{ м}$ за кілька циклів, R_0 – дисперсія шуму датчика $\sim 1 \text{ м}^2$, $R_{\text{max}} = 100 \text{ м}^2$, що практично ігнорує сигнал. Реалізація H_∞ -фільтра передбачає застосування робастного фільтра сильного стеження, у якому оцінка стану на кожному кроці визначається за мінімакс-критерієм, що мінімізує найгірший можливий випадок похибки. За номінальні значення коваріацій вибрано $R_0 = 1 \text{ м}$, $Q_0 = 0.25 \text{ м}^2$, виходячи з припущення про повільний дрейф висоти, а параметр робастності встановлено $\gamma = 10$, що визначає співвідношення між апостеріорною похибкою та процесним шумом.

У ПКФК корекція оцінки виконується лише тоді, коли інновація перевищує адаптивний поріг. В інший час система виконує тільки прогноз. Формально, якщо $||r^k|| > \delta$, де δ – динамічний поріг, залежний від очікуваного рівня шуму, тоді виконується стандартне оновлення Калмана. Інакше фільтр пропускає крок корекції. $R_0 = 1\text{м}$, $Q_0 = 0,25\text{м}^2$, $\delta = 2.5\text{м}$. Також розглядалося онлайн-адаптивний фільтр Калмана і тривіальне рішення без фільтра – порівняння сирого сигналу з порогом.

У ході експериментів зібрано дані для двох сценаріїв. Хоча обробка даних усередині контролера здійснювалася з періодом 10 мс (100 Гц) графіки в статті наведено з кроком дискретизації одна секунда. Під час запуску експерименту кожен цикл навігаційної задачі формував повний вектор оцінок, проте в журнал записувалися лише кожні соті результати. Таке зменшення частоти логування не впливає на репрезентативність, стрибки висоти на п'ять метрів і разові імпульсні атаки $\pm 300\text{-}500$ м залишаються добре помітними навіть за дискретизації 1 Гц, а при цьому розмір файлів і візуальна читабельність кривих значно поліпшуються. Висока внутрішня частота забезпечує коректну роботу фільтрів у реальному часі, тоді як раз на секунду збережених даних достатньо для цілей аналізу й публікації.

Для кількісної оцінки якості фільтрації використовувалися два показники: середньо-квадратичне відхилення та кількість хибних спрацювань. Середньо-квадратичне відхилення використовувалося для оцінювання точності відновлення сигналу, тоді як кількість хибних спрацювань характеризувала стійкість алгоритмів до імпульсних збурень і фальсифікованих вимірів.

Середньо-квадратичне відхилення використовується як міра похибки оцінювання та обчислюється відносно еталонного сигналу (вимірювання сенсора), що відповідає середньоквадратичному відхиленню похибки, а не статистичному розкиду вибірки відносно її середнього та визначимо його так:

$$\sigma = \sqrt{\frac{1}{N} \sum_{k=1}^N (x_k^{(f)} - z_k)^2}, \quad (4.2)$$

де $x_k^{(f)}$ - оцінка стану, отримана відповідним фільтром; z_k - вимірювання барометричного сенсора; N - кількість відліків поза інтервалами цілеспрямованих атак.

Хибні спрацювання визначалися бінарно за правилом для кожної секунди фіксувалося значення «1», якщо модуль різниці між виходом фільтра та сенсорним сигналом перевищував 3 м, і «0» в іншому випадку:

$$FA_k = \begin{cases} 1, & |x_k^{(f)} - z_k| > 3 \text{ м} \\ 0, & |x_k^{(f)} - z_k| \leq 3 \text{ м} \end{cases}, \quad (4.3)$$

де $x_k^{(f)}$ - оцінка стану, отримана відповідним фільтром; z_k - вимірювання барометричного сенсора.

Загальна кількість хибних спрацювань визначалася як сума таких подій за інтервал спостереження.

На рис. 4.15 показано графічно, а на рис. 4.16 і табл. 4.1 – таблично типовий приклад фільтрів у ситуації, коли на 20-й секунді і на 38-й секунді відбувається заплановане зниження висоти БПЛА, законний стрибок рівня з переходом на новий режим. Після переходів на новий режим фільтри кожний по своєму підлаштовуються до нового рівня, повільно виходячи на рівень, накопичуючи похибку, криві класичного КФ і ККФ рухаються однаково.

Як видно з табл. 4.1, за відсутності атак комбінований фільтр Калмана має ті самі показники точності та хибних спрацювань, що й класичний фільтр Калмана, тобто працює у штатному режимі без погіршення характеристик. Затримка виявлення в ККФ визначається параметром N та періодом дискретизації і за $N = 3$ та періоді 10 мс не перевищує 30 мс, що більше за мінімальну затримку порогового детектора (10 мс), але суттєво менше, ніж у класичного фільтра Калмана з фіксованим порогом,

оскільки в останньому момент спрацювання визначається інерційною динамікою збіжності оцінки, тоді як у ККФ затримка жорстко обмежена параметром N і періодом дискретизації.

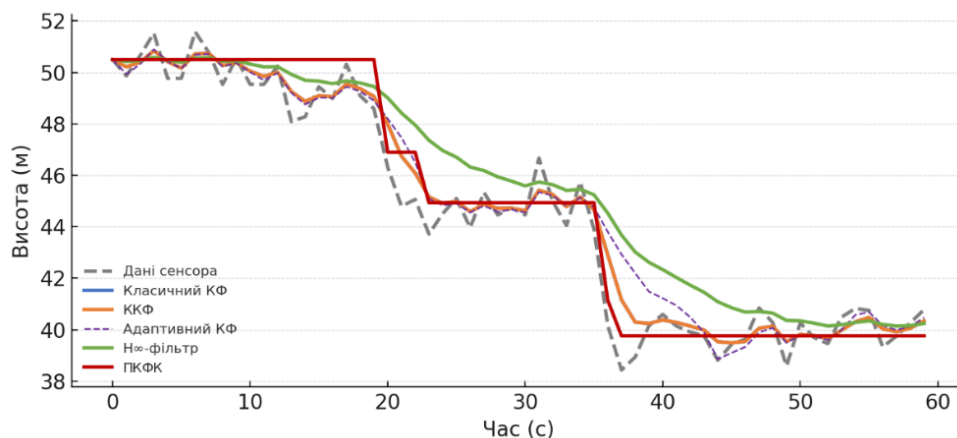


Рисунок 4.15 - Графік порівняння поведінки фільтрів при режимних змінах

	A	B	C	D	E	F	G	H
1	Time_sec	Sensor_Data	Kalman_Filter	KKF	Adaptive_KF	Hinf_Filter	ETKF	
2	1	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123	
3	2	49.86173569882882	50.213416073452926	50.213416073452926	49.931138660056774	50.4262773677844	50.49671415301123	
4	3	50.647688538100695	50.39165488547389	50.39165488547389	50.31801487516154	50.45238119215656	50.49671415301123	
5	4	51.52302985640802	50.84165551406326	50.84165551406326	50.892105866226615	50.584378789070485	50.49671415301123	
6	5	49.765846625276666	50.41874364371871	50.41874364371871	50.42700920545653	50.48008887019351	50.49671415301123	
7	6	49.76586304305082	50.16320732614738	50.16320732614738	50.15388583472261	50.386854367988015	50.49671415301123	
8	7	51.57921281550739	50.71653034426986	50.71653034426986	50.684180529658256	50.54531611744295	50.49671415301123	
9	8	50.76743472915291	50.73640991039906	50.73640991039906	50.716579852083996	50.5752282536835	50.49671415301123	
10	9	49.53052561406505	50.26558445283847	50.26558445283847	50.224015041977154	50.43315862506478	50.49671415301123	
11	10	50.542560043585965	50.37371778678803	50.37371778678803	50.367149234216456	50.44814411106918	50.49671415301123	
12	11	49.53658230718754	50.04690398359193	50.04690398359193	50.01300420546548	50.32261178777273	50.49671415301123	
13	12	49.53427024642974	49.84677645746603	49.84677645746603	49.69391413281421	50.213618199868606	50.49671415301123	
14	13	50.24196227156603	50.001052727719106	50.001052727719106	49.977545837740216	50.21754843843798	50.49671415301123	
15	14	48.0867197553422	49.25371901558106	49.25371901558106	49.20615379280139	49.92144407451889	50.49671415301123	
16	15	48.275082167486964	48.87167060136232	48.87167060136232	48.754104205684904	49.69229574339176	50.49671415301123	
17	16	49.437712470759024	49.092646698389714	49.092646698389714	49.01551184698322	49.656819671815164	50.49671415301123	
18	17	48.987168879665575	49.051469400273085	49.051469400273085	49.00443646130458	49.56342232779153	50.49671415301123	
19	18	50.31424733259527	49.54444301715711	49.54444301715711	49.441541320512634	49.66820907828034	50.49671415301123	
20	19	49.09197592447879	49.367805200660996	49.367805200660996	49.25668295910986	49.587750164091524	50.49671415301123	
21	20	48.58769629866471	49.06325988735429	49.06325988735429	48.89876805756984	49.448063990169935	50.49671415301123	
22	21	46.3190838920294	47.99196595028267	47.99196595028267	48.195314511143735	49.01089751476696	46.8953087556131	
23	22	44.796801329562115	46.74461137463762	46.74461137463762	47.46765000343535	48.42200930862734	46.8953087556131	
24	23	45.060775384219134	46.08726166776737	46.08726166776737	46.509293235912395	47.95223466555188	46.8953087556131	
25	24	43.71772663240789	45.162223142845	45.162223142845	45.08649340302008	47.36034660624453	44.93422671917952	
26	25	44.51005554792734	44.90762460727392	44.90762460727392	44.836226857349935	46.96190845795561	44.93422671917952	
27	26	45.09983033073888	44.982659454303075	44.982659454303075	44.902352371217184	46.70159634702432	44.93422671917952	
28	27	43.964105780319926	44.58502811565082	44.58502811565082	44.55836691717483	46.3188881055103	44.93422671917952	
29	28	45.3381282165111	44.879029510857336	44.879029510857336	44.81697930330772	46.18177097325353	44.93422671917952	
30	29	44.45942517907307	44.71522092971616	44.71522092971616	44.57779519053464	45.940969235482854	44.93422671917952	
31	30	44.73747562518605	44.723908900293466	44.723908900293466	44.67305739342869	45.77270545785726	44.93422671917952	
32	31	44.45846404899354	44.620282361745204	44.620282361745204	44.54730525043011	45.588955351148556	44.93422671917952	
33	32	46.6670536605804	45.419316445320675	45.419316445320675	45.34780447564432	45.739690189626266	44.93422671917952	
34	33	44.98785249773586	45.250878010076505	45.250878010076505	45.208067329241835	45.634570573520364	44.93422671917952	
35	34	44.04806016393969	44.7813121123436	44.7813121123436	44.80986790215446	45.412748444598414	44.93422671917952	
36	35	45.74029042089287	45.15568593112804	45.15568593112804	45.12949820158352	45.458544765385426	44.93422671917952	
37	36	43.90124071502608	44.66596531719839	44.66596531719839	44.75523839706327	45.24080473443954	44.93422671917952	
38	37	40.167090876003805	42.909657807668104	42.909657807668104	43.80505218501202	44.53140379713976	41.14653970494582	
39	38	43.27763490089618	41.1617360453749	41.1617360453749	42.92770803430569	43.67867613447436	39.75883581541964	

Рисунок 4.16 - Фрагмент таблиці даних порівняння поведінки фільтрів при режимних змінах

Таблиця 4.1 – Порівняння характеристик методів виявлення аномалій в стандартних умовах

Метод	Хибні спрацювання (за 1 хв)	Середньо-квадратичне Відхилення(м)	Навантаження CPU (%)	Опис навантаження
Класичний фільтр Калмана	0	0,817	~3	Постійне оновлення, але формули прості
Адаптивний фільтр Калмана	3	1,04	~6	Постійно змінює Q/R , постійні обчислення навантажують CPU
ККФ (пропонований фільтр)	0	0,817	~3	Змінює Q/R , оцінює нев'язку, перемикає моделі – трохи складніше за простий ФК
H_{∞} фільтр	5	1,707	~8	Мінімакс-оптимізація, складна матрична алгебра кожного кроку
ПКФК	0	0,966	~2	оновлення лише при перевищенні порогу, економія

На рис. 4.17 і рис. 4.18 було аналогічно зроблене графічне і табличне представлення порівняння поведінки різних фільтрів при режимних змінах, але також спостерігаються точкові атаки. В графіках сіра штрихова крива – фактичний сигнал датчика з шумом і впливом атаки в заштрихованому інтервалі. Оранжева крива – оцінка рівня комбінованим фільтром Калмана. Фіолетова штрихова, червона, зелена, синя – оцінка іншими фільтрами. Як видно на графіку (рис. 4.17) при точкових атаках ККФ і адаптивний фільтр Калмана демонструють майже однакові нульові відхилення. Але ККФ не потребує великих обчислень тому що ігнорує сильні відхилення як ненормальні, а адаптивний калман фільтр весь час вираховує ковзну дисперсію останніх резидуалів і динамічно оновлює параметри Q та R . H_{∞} найменш чутливий,

практично не повірив аномальним вимірам. ПКФК пропускає кроки з відхиленнями, оновлює лише коли інновація перевищує поріг.

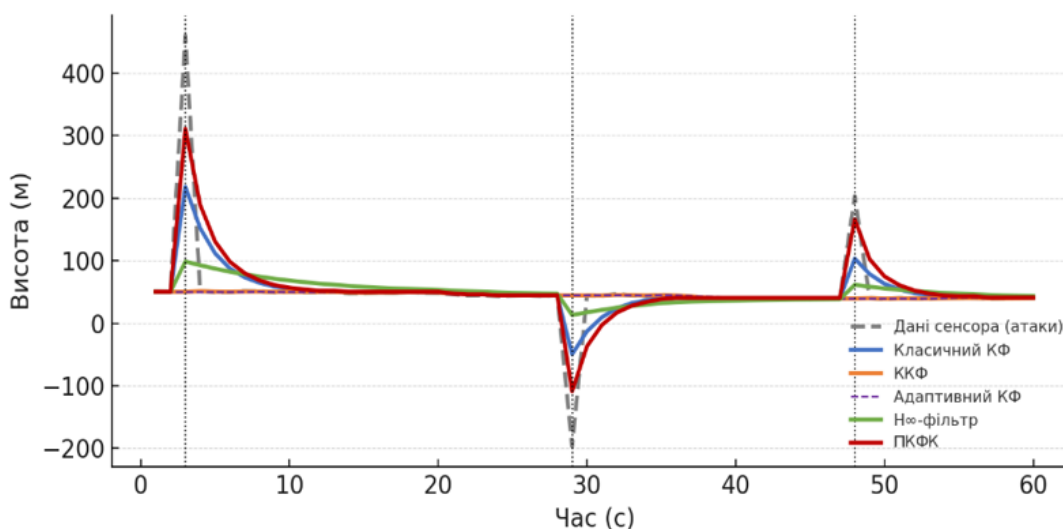


Рисунок 4.17 - Графік порівняння поведінки фільтрів при режимних змінах і атаках

	A	B	C	D	E	F	G	H	I
1	Time_sec	Sensor_Data_Attacked	Kalman_Filter	KKF	Adaptive_KF	Hinf_Filter	ETKF		
2	1	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123		
3	2	49.86173569882882	50.214461538461535	50.214461538461535	49.93092981865409	50.42900618556701	50.49671415301123		
4	3	459.97040207878206	218.39093600058854	50.214461538461535	50.046958427272747	98.71291460921711	311.072074050134		
5	4	51.52302985640802	152.01857668318135	51.47109638380637	50.04746712124784	92.89461816167423	189.1138705807939		
6	5	49.765846625276666	111.82357805854198	50.538842729004216	50.047352304721365	87.40007179121004	130.78955347254566		
7	6	49.76586304305082	87.53591686446838	50.197602584850785	50.04723497445092	82.4878705264556	98.32645765712263		
8	7	51.57921281550739	73.48566311312113	50.763865882759326	50.04790129685602	78.38028421653503	79.89947703076649		
9	8	50.76743472915291	64.6145702187972	50.76630391168801	50.733436939314956	74.66207898930436	68.48661372582623		
10	9	49.53052561406505	58.72495071138559	50.28043307641945	50.1319505915048	71.24435513849569	61.07625399301519		
11	10	50.542560043585965	55.52948498010466	50.382016744148295	50.315794563331025	68.4083331896757	56.960976091574764		
12	11	49.53658230718754	53.19121786177867	50.052999963832136	50.00843917555611	65.80995127860228	54.06337840846969		
13	12	49.53427024642974	51.76191186771391	49.848756935397546	49.74191207785655	63.559137122930146	52.293479242314625		
14	13	50.24196227156603	51.16777392493498	50.00151307453868	50.01878415846773	61.71228226770275	52.293479242314625		
15	14	48.0867197553422	49.96624616271532	49.25524558419286	49.22958426072924	59.8193017301289	50.31357478497664		
16	15	48.275082167486964	49.307955323061705	48.874514513231844	48.76129191582664	58.213207880084454	50.31357478497664		
17	16	49.437712470759024	49.35950401392828	49.09527482001044	49.02024019232337	56.99066506534418	50.31357478497664		
18	17	48.987168879665575	49.21525399889329	49.05417667230501	49.00847730238329	55.87479859242147	50.31357478497664		
19	18	50.31424733259527	49.64262993089302	49.54443572752043	49.43798069918396	55.09816334733267	50.31357478497664		
20	19	49.09197592447879	49.42688972369802	49.36702932083915	49.253921851110285	54.25924893221047	50.31357478497664		
21	20	48.58769629866471	49.100177847375925	49.06368620260513	48.89619663286072	53.46737589444008	50.31357478497664		
22	21	46.3190838920294	48.01482921195721	47.992583426170356	48.192058464907035	52.46877793914807	47.55984954863415		
23	22	44.796801329562115	46.75979781186995	46.74623649738488	47.46413037641435	51.397123974598486	46.22294105210524		
24	23	45.060775384219134	46.09621679820264	46.087949656782676	46.507161248366806	50.511431333073666	46.22294105210524		
25	24	43.71772663240789	45.16856979188958	45.16353004279878	45.087240795891475	49.56214258729485	44.98130012648171		
26	25	44.51005554792734	44.91147191414604	44.90839962342634	44.83599082651698	48.85591156620955	44.98130012648171		
27	26	45.09983033073888	44.9850710548362	44.983198150194376	44.90231893230698	48.33084806109624	44.98130012648171		
28	27	43.964105780319926	44.58489540758313	44.5837536627714	44.556344379454586	47.71979222925791	44.98130012648171		
29	28	45.3381282165111	44.87967933264676	44.878983311553725	44.81616305236598	47.38708053345665	44.98130012648171		
30	29	-196.76001839998307	-49.45360808755072	44.878983311553725	44.64045771241802	13.252793508465714	-108.72836814780847		
31	30	44.73747562518605	-12.681534673114285	44.745217015251185	44.64053264546111	17.655107127593254	-36.63916039789649		
32	31	44.45846404899354	9.625846376140476	44.588913800606754	44.640391806458496	21.40282188715078	-2.6979860337392054		
33	32	46.66705036605804	24.087446948505793	45.5125646334607	44.642029098034804	24.93557545613704	17.082012537634352		
34	33	44.98785249773586	32.24755707661749	45.29851046789213	44.642319411159401	27.739522946419207	28.08287698383347		
35	34	44.04806016393969	36.85509156287341	44.80228587399102	44.10392590883825	30.020015228038808	34.33872123215029		
36	35	45.74029042089287	40.3236550032594	45.170809886192906	44.91708510576096	32.217954782316056	38.79560288051523		
37	36	43.90124071502608	41.71981790056817	44.67346800912935	44.55434431845888	33.85132168110679	40.78929008852951		
38	37	40.167090876003805	41.11478727507474	42.913744779793284	43.77764475969954	34.73479214219043	40.78929008852951		
39	38	43.7376349089618	40.0667799477816	41.16274354038064	43.01512423887271	35.25145273836798	40.78929008852951		

Рисунок 4.18 - Фрагмент таблиці даних порівняння поведінки фільтрів при режимних змінах і атаках

Для кількісного порівняння, табл. 4.2 наводить зведення показників п'яти підходів при точкових атаках: класичний фільтр Калмана, адаптивний фільтр Калмана, пропонувані ККФ, N_{∞} фільтр, ПКФК. Дані усереднені за кількома прогонами в різних режимах.

Таблиця 4.2 – Порівняння характеристик методів виявлення аномалій при режимних змінах і атаках

Метод	Хибні спрацювання (за 1 хв)	Середньоквадратичне відхилення	Навантаження CPU (%)	Опис навантаження
Класичний фільтр Калмана	19	20,46	~3	Постійне оновлення, але формули прості.
адаптивний фільтр Калмана	3	1,19	~6	Постійно змінює Q/R, постійні обчислення навантажують CPU
ККФ (пропонувані фільтр)	0	0,82	~3.5	Змінює Q/R, оцінює нев'язку, перемикає моделі – трохи складніше за простий ФК
N_{∞} фільтр	47	14,13	~8	Мінімакс-оптимізація, складна матрична алгебра кожного кроку.
ПКФК	21	28,19	~2	оновлення лише при перевищенні порогу, економія.

ККФ за рахунок адаптивності практично не реагує на допустимі стрибки. Під час експериментів одна хибна тривога могла статися лише в разі, коли перехід був надто різким і тривав довше N циклів, що траплялось рідко. Класичний же метод подавав кілька помилкових тривог щоразу, коли відбувався режимний перехід (в середньому 6-7 хибних спрацювання на кожен перехід, що сумарно дало близько 19 за хвилину при наших умовах, враховуючи періодичні переключення).

H_∞ -фільтр показав найвищу робастність до шумів і атак – завдяки мінімакс-критерію він практично не піддається коливанням сенсора. Однак ціною такої стійкості є помітна інерційність. Фільтр повільніше реагує навіть на легальні зміни режиму, а також часто продовжує згладжувати вже завершений стрибок. Через це H_∞ може викликати підвищену кількість хибних тривог у динамічних системах з частими легальними переходами. Тим не менш, для стабільних режимів він надає високу якість згладження з низькою чутливістю до атак.

ПКФК, навпаки, показав мінімальне навантаження на CPU завдяки евент-тригерованій схемі – корекція виконується лише тоді, коли інновація перевищує поріг. Такий підхід дозволяє зменшити кількість обчислень, однак у тестах проявив підвищену чутливість до шумів – поріг спрацьовує навіть при незначних флуктуаціях сенсора, що збільшує кількість хибних спрацювань.

Щодо навантаження CPU ККФ з додатковою логікою зайняв близько 3.5% часу процесора на 100 Гц, що є дуже малим навантаженням, залишає ресурс для інших задач. H_∞ – трохи вищий (~8%) через складнішу матричну обробку. ПКФК найменш вимогливий (~2%), оскільки більшість циклів пропускає оновлення. Класичний фільтр Калмана – аналогічно ККФ, близько 3%, адаптивний фільтр Калмана ~6%.

Для кількісного порівняння ефективності методів оцінювання стану було використано інтегральний показник, який узагальнює три ключові критерії: точність оцінювання, обчислювальне навантаження та частоту хибних спрацювань. Перед обчисленням інтегрального показника всі часткові критерії було нормалізовано, що дозволило забезпечити їх порівнюваність. За умови рівнозначної ваги критеріїв визначено інтегральну оцінку ефективності кожного з розглянутих методів. Оскільки середньоквадратична похибка є мірою відхилення оцінки від еталонного сигналу, менші її значення відповідають вищій точності, тому для узгодження з іншими критеріями застосовано інверсну нормалізацію за схемою *min – max*.

Для комбінованого фільтра Калмана середньоквадратична похибка (м): $E_{СКП} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{28,19 - 0,82}{28,19 - 0,82} = 1$. Середнє значення процесора(%): $E_{CPU} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{8 - 3,5}{8 - 2} = 0,75$. Хибні спрацювання: $E_{Er} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{47 - 0}{47 - 0} = 1$. Інтегральний показник: $I = w_{CPU}E_{CPU} + w_{СКП}E_{СКП} + w_{Er}E_{Er}$, де $w_{CPU} + w_{СКП} + w_{Er} = 1$. Для рівнозначної важливості беремо $w_{CPU} = w_{СКП} = w_{Er} = \frac{1}{3}$. $I = \frac{1}{3}(0,75 + 1 + 1) = 0,9166666667 \approx 9,17$.

Для класичного фільтра Калмана середньоквадратична похибка: $E_{СКП} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{28,19 - 20,46}{28,19 - 0,82} = 0,2824324453$. Середнє значення процесора (%): $E_{CPU} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{8 - 3}{8 - 2} = 0,8333333333$. Хибні спрацювання: $E_{Er} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{47 - 19}{47 - 0} = 0,5957446809$. Інтегральний показник: $I = w_{CPU}E_{CPU} + w_{СКП}E_{СКП} + w_{Er}E_{Er}$, де $w_{CPU} + w_{СКП} + w_{Er} = 1$. Для рівнозначної важливості беремо $w_{CPU} = w_{СКП} = w_{Er} = \frac{1}{3}$. $I = \frac{1}{3}(E_{CPU} + E_{СКП} + E_{Er}) = \frac{1}{3}(0,8333333333 + 0,2824324453 + 0,5957446809) = 0,5705034865 \approx 5,71$.

Для адаптивного фільтра Калмана середньоквадратична похибка: $E_{СКП} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{28,19 - 1,19}{28,19 - 0,82} = 0,9864815491$. Середнє значення процесора (%): $E_{CPU} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{8 - 6}{8 - 2} = 0,3333333333$. Хибні спрацювання: $E_{Er} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{47 - 3}{47 - 0} = 0,9361702128$. Інтегральний показник: $I = w_{CPU}E_{CPU} + w_{СКП}E_{СКП} + w_{Er}E_{Er}$, де $w_{CPU} + w_{СКП} + w_{Er} = 1$. Для рівнозначної важливості беремо $w_{CPU} = w_{СКП} = w_{Er} = \frac{1}{3}$. $I = \frac{1}{3}(E_{CPU} + E_{СКП} + E_{Er}) = \frac{1}{3}(0,3333333333 + 0,9864815491 + 0,9361702128) = 0,7519950317 \approx 7,52$.

Для N_{∞} фільтра середньоквадратична похибка: $E_{СКП} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{28,19 - 14,13}{28,19 - 0,82} = 0,5137084399$. Середнє значення процесора (%): $E_{CPU} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{8 - 8}{8 - 2} = 0$. Хибні спрацювання: $E_{Er} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{47 - 47}{47 - 0} = 0$. Інтегральний показник:

$I = w_{CPU}E_{CPU} + w_{СКП}E_{СКП} + w_{Er}E_{Er}$, де $w_{CPU} + w_{СКП} + w_{Er} = 1$. Для рівнозначної важливості беремо $w_{CPU} = w_{СКП} = w_{Er} = \frac{1}{3}$. $I = \frac{1}{3}(E_{CPU} + E_{СКП} + E_{Er}) = \frac{1}{3}(0+0.5137084399+0) = 0.1712361466 \approx 1,71$.

Для ПКФК фільтра середньоквадратична похибка: $E_{СКП} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{28,19 - 28,19}{28,19 - 0,82} = 0$. Середнє значення процесора (%): $E_{CPU} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{8 - 2}{8 - 2} = 1$. Хибні спрацювання: $E_{Er} = \frac{X_{max} - X_{curr}}{X_{max} - X_{min}} = \frac{47 - 21}{47 - 0} = 0,5531914894$. Інтегральний показник: $I = w_{CPU}E_{CPU} + w_{СКП}E_{СКП} + w_{Er}E_{Er}$, де $w_{CPU} + w_{СКП} + w_{Er} = 1$. Для рівнозначної важливості беремо $w_{CPU} = w_{СКП} = w_{Er} = \frac{1}{3}$. $I = \frac{1}{3}(E_{CPU} + E_{СКП} + E_{Er}) = \frac{1}{3}(1+0+0,5531914894) = 0,5177304965 \approx 5,18$.

Результати порівняння ефективності наведені у табл. 4.3.

Таблиця 4.3 – Підсумкова порівняльна таблиця ефективності фільтрів

Метод	Ефективність
Класичний фільтр Калмана	5,71
Адаптивний фільтр Калмана	7,52
ККФ (пропонований)	9,17
H_{∞} фільтр	1,71
ПКФК	5,18

За результатами експериментів інтегральний показник ефективності для ККФ становить $I_{ККФ} = 9.17$. Порівняємо його з адаптивним фільтром Калмана, який має друге місце по ефективності. Відносний приріст ефективності комбінованого фільтра Калмана порівняно з адаптивним визначимо так:

$$\Delta I = \frac{I_{ККФ} - I_{АФК}}{I_{АФК}} \cdot 100\%, \quad (4.4)$$

де $I_{ККФ}$ – показник ефективності для ККФ; $I_{АФК}$ – показник ефективності для адаптивного фільтра Калмана.

Тепер маємо: $\Delta I = \frac{9,17-7,52}{7,52} \cdot 100\% = 21,941489361\% \approx 21,94\%$. Отже

проведені експерименти підтвердили, що запропонований ККФ здатен працювати на реальному мікроконтролері, виявляючи атаки на сенсори навіть за умов частих режимних змін. Він перевершує традиційні підходи в швидкості та точності спрацювання при порівнянному рівні ресурсоспоживання. Особливо помітна різниця у зменшенні хибних тривог. Система з ККФ може стабільно функціонувати, не реагуючи на власні перемикання режимів як на щось підозріле, що важливо для експлуатації, тому що уникає зайвих переривань роботи чи помилкових аварійних зупинок.

Отже, виконано експериментальну перевірку запропонованого методу виявлення фальсифікацій та аномалій у сигналах кіберфізичних систем на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів. Результати порівняльних досліджень показали, що застосування комбінованого фільтра Калмана з подієвим перемиканням забезпечує найвищу інтегральну ефективність серед розглянутих методів ($I = 9,17$), перевищуючи класичний фільтр Калмана ($I = 5,71$), адаптивний фільтр Калмана ($I = 7,52$), H_∞ -фільтр ($I = 1,71$) та ПКФК ($I = 5,18$), при збереженні низьких обчислювальних витрат і стабільної роботи в реальному часі. Отримані результати підтверджують практичну придатність запропонованого методу для використання у вбудованих кіберфізичних системах з обмеженими ресурсами.

4.4 Висновки до четвертого розділу

Виконано експериментальну перевірку методу динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції, реалізованого у вигляді модифікованого планувальника FreeRTOS із зовнішнім аналітичним модулем. За результатами першої серії експериментів встановлено зниження використання процесора на 3,94 %, зменшення середнього часу виконання задач на 10,87 %, при

незначному збільшенні використання пам'яті на 0,87 %. У другій серії експериментів зниження використання процесора становило 13,08 % за рахунок більш рівномірного розподілу обчислювальних фаз задач у часі та ефективнішого використання задіяних системних ресурсів, зменшення часу виконання задач на 3,61 %, при збільшенні використання пам'яті на 0,51 %. Інтегральний показник ефективності методу за результатами експериментальної перевірки становить 5,19 %.

Проведено експериментальне дослідження ефективності методу превентивного відновлення компонентів ОСРЧ на основі низькорозмірної марковської моделі та програмного предиктивного сторожового контролю у поєднанні з апаратним сторожовим таймером. Встановлено, що застосування ПЗСК забезпечує скорочення середнього часу відновлення з 2,7 с до 0,82 с (приблизно у 3,29 раза) та зменшення частки простою системи з 4,5 % до близько 1,37 %, що відповідає відносному скороченню простою приблизно на 69,6 %. Додаткове обчислювальне навантаження на процесор не перевищувало 1,6 % і не впливало на виконання задач реального часу.

Проведено експериментальну перевірку методу виявлення фальсифікацій та аномалій у сигналах КФС на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів. За інтегральним показником ефективності, що враховує середньоквадратичну похибку оцінювання, кількість хибних спрацювань та обчислювальне навантаження, комбінований фільтр Калмана продемонстрував найвище значення ефективності ($I = 9,17$), перевищивши другий за ефективністю адаптивний фільтр Калмана приблизно на 21,9 %.

Основні результати розділу опубліковані у працях [2; 4; 5; 6; 7; 8; 9; 10; 136].

ВИСНОВКИ

У результаті виконання дисертаційного дослідження було розв’язано актуальну науково-прикладну задачу розроблення методів організації роботи ОСРЧ для КФС в умовах інтенсивних змін станів, нестационарних навантажень і обмежених обчислювальних ресурсів, що забезпечують врахування багатовимірної динаміки системного навантаження, своєчасне виявлення відхилень режимів роботи та виявлення аномалій і спотворень вимірювань, а також розроблено відповідні засоби.

У роботі отримано наступні наукові та практичні результати:

1. Проведено аналіз існуючих архітектур ОСРЧ для КФС з інтенсивними змінами станів та визначено їхні обмеження щодо адаптивності, відмовостійкості та інтеграції інтелектуальних методів обробки даних.

2. Удосконалено архітектуру планувальника задач шляхом залучення зовнішнього модуля машинного навчання та зворотного каналу самоадаптації, що забезпечує збирання багатовимірної телеметрії, її тензорне подання та адаптивне оновлення параметрів планування без втручання користувача.

3. Розроблено метод динамічного розподілу задач і ресурсів в ОСРЧ на основі тензорної декомпозиції, який враховує взаємозв’язки між параметрами задач і ресурсів та забезпечує підвищення ефективності використання обчислювальних ресурсів. Експериментальна перевірка методу була проведена у двох серіях експериментів, що моделювали різні режими функціонування системи. За результатами першої серії експериментів середнє значення завантаженості процесора зменшилося на 3,94 %, середній час виконання задач зменшився на 10,87 %, тоді як використання пам’яті збільшилося на 0,87 %. У другій серії експериментів зниження використання процесора становило 13,08 %, зменшення часу виконання задач – 3,61 %, при збільшенні використання пам’яті на 0,51 %. Для узагальненої оцінки ефективності запропонованого підходу було введено інтегральний показник, визначений як середнє відносне покращення показників використання процесора,

використання пам'яті та часу виконання задач. За результатами двох серій експериментів його значення становило $I = 5,19$.

4. Розроблено метод превентивного відновлення компонентів ОСРЧ на основі низькорозмірної марковської моделі прогнозування станів у поєднанні з багаторівневим сторожовим контролем, що дозволяє завчасно виявляти деградаційні процеси та ініціювати локальні відновлювальні дії. Програмний рівень виконує ймовірнісний моніторинг станів задач і черг повідомлень, обчислює ризики деградації та, у разі потреби, ініціює локальне відновлення окремих компонентів без перезапуску всієї системи. Такий підхід дозволяє зменшити сумарний час простою системи з 4,5% до 1,37%, що відповідає скороченню на 69,63%, а також прискорити відновлення після збоїв у 3,29 рази порівняно з використанням лише традиційного апаратного сторожового таймера.

5. Розроблено метод виявлення фальсифікацій та аномалій у комп'ютерних системах для сигналів КФС на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів, придатний для реалізації в умовах обмежених ресурсів мікроконтролерів. На відміну від класичної реалізації, запропонований варіант працює у комбінованому режимі з перемиканнями станів, що дозволяє коригувати оцінку параметрів при виникненні різких змін чи аномалій. За результатами узагальненої оцінки, яка базується на інтегральному критерії з урахуванням середньоквадратичної похибки оцінювання, кількості хибних спрацювань та обчислювальних витрат, комбінований фільтр Калмана показав найкращий рівень ефективності з інтегральним показником $I = 9,17$, перевищивши найближчий за характеристиками адаптивний фільтр Калмана приблизно на 21,9%.

6. Проведено експериментальну перевірку розроблених методів і архітектурних рішень на вбудованій платформі під керуванням ОСРЧ та було оцінено їх ефективність за показниками використання процесора й пам'яті, часу виконання задач, тривалості простою системи, кількості хибних спрацювань, швидкодії відновлення та обчислювальної складності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Козельський О.В. Методи та засоби створення мультикомп'ютерних систем з подвійною автентифікацією потоків даних в корпоративних мережах. *Актуальні проблеми комп'ютерних наук: Збірник наукових праць. Матеріали XV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2023»*, Україна, Хмельницький, ХНУ, листопад 17-18, 2023. С. 142–145. URL: <https://kn.khmnua.edu.ua/wp-content/uploads/sites/18/apkn-2023-corpuspaper.pdf>
2. Козельський О., Савенко Б. Виявлення фальсифікацій та аномалій у цифрових сигналах від кіберфізичних систем на основі комбінованого фільтра Калмана. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 2025. №2. С. 23–35. <https://doi.org/10.32782/IT/2025-2-4>
3. Козельський О., Савенко Б. Зовнішня адаптивна система кластеризації для підвищення гнучкості операційних систем реального часу на основі динамічного планування завдань. Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 5. 2025. Ч. 2. С. 124-132 DOI: <https://doi.org/10.32782/2663-5941/2025.5.2/18>
4. Козельський О., Савенко Б., Савенко О. Дворівнева стратегія підвищення відмовостійкості операційних систем реального часу з використанням ймовірнісного аналізу. *Herald of Khmelnytskyi National University. Technical Sciences*. 2025. №353(3.2). С. 438-446. DOI: <https://doi.org/10.31891/2307-5732-2025-353-60>
5. Козельський О.В. Метод тензорної декомпозиції для адаптивного розподілу ресурсів у системах реального часу. *Measuring and computing devices in technological processes*. 2025. №82(2). С. 426-433. DOI: <https://doi.org/10.31891/2219-9365-2025-82-61>
6. Kozelskiy O., Kashtalian A., Stetsyuk V., Martiniuk D., Sachenko A. A model of an intelligent clustering system with an external module for the architecture of RTOS with intensive changes of states regarding their flexibility and balancing. *1st International*

Workshop on Advanced Applied Information Technologies: (AdvAIT-2024), Khmelnytskyi, Ukraine and Zilina, Slovakia, December 5, 2024. Vol. 3899. P. 234-243. URL: <https://ceur-ws.org/Vol-3899/paper21.pdf> (Scopus)

7. Kozelskiy O., Drozd A., Savenko B., Gaj P. A model for probabilistic monitoring and proactive restart of real-time operating systems under intensive state changes in cyber-physical systems. *2nd International Workshop on Intelligent & CyberPhysical Systems: (ICyberPhyS 2025)*, Khmelnytskyi, Ukraine, July 4, 2025. Vol. 4013. P. 198-210. URL: <https://ceur-ws.org/Vol-4013/paper16.pdf> (Scopus)

8. Kozelskiy O., Stetsyuk Y., Stetsyuk V., Stetsyuk M., Gaj P. Architecture of the system with a subsystem of providing fault tolerance, survivability and information protection of specialized information technology. *ICyberPhyS-2024: 1st International Workshop on Intelligent & CyberPhysical Systems*, June 28, 2024, Khmelnytskyi, Ukraine, Vol-3736 pp 293-311. URL: <https://ceur-ws.org/Vol-3736/paper22.pdf> (Scopus)

9. Козельський О., Савенко О. Проактивний механізм інформаційної безпеки в операційних системах реального часу з використанням гібридного сторожового таймера. *Measuring and computing devices in technological processes*. 2025. №83(3). С. 459 – 466. DOI: <https://doi.org/10.31891/2219-9365-2025-83-57>

10. Stetsyuk, M., Cheshun, V., Stetsyuk, Y., Kozelskiy, O., & Salem, B. M. (2024). A model of a DDoS attack scenario on elements of specialized information technology and methods of combating cybercriminals. *CEUR Workshop Proceedings*, Vol-4013 pp 265–273. URL: <https://ceur-ws.org/Vol-3675/paper19.pdf> (Scopus)

11. Xu Z., Zhang Y., Li H. et al. Dynamic resource provisioning for cyber-physical systems in cloud-fog-edge computing. *Journal of Cloud Computing*. 2020. Vol. 9. Article 32. URL: <https://doi.org/10.1186/s13677-020-00181-y>

12. Mohammad A., Das R., Islam M. A., Mahjabeen F. Real-time Operating Systems (RTOS) for Embedded Systems. *Asian Journal of Mechatronics and Electrical*

Engineering. 2024. Vol. 2, No. 2. P. 95–104. URL: <https://doi.org/10.55927/ajmee.v2i2.7761>

13. Fayziev A. X. Real-time monitoring and control in automated industrial environments. *Journal of Applied Science and Social Science*. 2025. Vol. 15, No. 4. P. 55–60. URL: <https://www.internationaljournal.co.in/index.php/jasass/article/view/906>

14. Kumar A., Saini M., Saini D. K. et al. Cyber physical systems reliability modelling: critical perspective and its impact. *International Journal of System Assurance Engineering and Management*. 2021. Vol. 12. P. 1334–1347. URL: <https://doi.org/10.1007/s13198-021-01305-6>

15. Ashraf K., Le Moullec Y., Pardy T., Rang T. Design of cyber bio-analytical physical systems: Formal methods, architectures, and multi-system interaction strategies. *Microprocessors and Microsystems*. 2023. Vol. 97. Article 104780. URL: <https://doi.org/10.1016/j.micpro.2023.104780>

16. Pradhani J. et al. Prototype development for water leakage monitoring system with RTOS implementation. In: *Proceedings of the 2024 International Conference on Distributed Computing and Optimization Techniques (ICDCOT)*. IEEE, 2024. P. 1–7. URL: <https://doi.org/10.1109/ICDCOT61034.2024.10515556>

17. Cheng X., Tang H. Dynamic memory allocation of embedded real-time operating system based on TLSF. In: Chen Q., Wu J., Zhang S., Yuan C., Batten L., Li G. (eds). *Applications and Techniques in Information Security (ATIS 2018)*. Communications in Computer and Information Science. Vol. 950. Singapore: Springer, 2018. P. 25–34. URL: https://doi.org/10.1007/978-981-13-2907-4_3

18. Mehalaine R., Boutekkouk F. Energy consumption reduction in real time multiprocessor embedded systems with uncertain data. In: Silhavy R. (eds). *Artificial Intelligence and Bioinspired Computational Methods (CSOC 2020)*. Advances in Intelligent Systems and Computing. Vol. 1225. Cham: Springer, 2020. P. 37–48. URL: https://doi.org/10.1007/978-3-030-51971-1_4

19. Paredes C. M., Martínez Castro D., González Potes A., Rey Piedrahita A., Ibarra Junquera V. Design procedure for real-time cyber–physical systems tolerant to cyberattacks. *Symmetry*. 2024. Vol. 16, No. 6. Article 684. URL: <https://doi.org/10.3390/sym16060684>
20. Manjunath V., Baunach M. A framework for static analysis and verification of low-level RTOS code. *Journal of Systems Architecture*. 2024. Vol. 154. Article 103220. URL: <https://doi.org/10.1016/j.sysarc.2024.103220>
21. Pascucci F. Fault tolerance in cyber-physical systems. In: Jajodia S., Samarati P., Yung M. (eds). *Encyclopedia of Cryptography, Security and Privacy*. Berlin, Heidelberg: Springer, 2023. URL: https://doi.org/10.1007/978-3-642-27739-9_1724-2
22. Zalke J. B., Pandey S. K. Dynamic partial reconfigurable embedded system to achieve hardware flexibility using 8051-based RTOS on Xilinx FPGA. In: *Proceedings of the 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies (ACT)*. IEEE, 2009. P. 684–686. URL: <https://doi.org/10.1109/ACT.2009.174>
23. Amadera K., Ichimura A., Yokoyama T., Yoo M. A real-time operating system for cyber-physical systems based on physical time and logical time. In: Wehrmeister M. A., Kreutz M., Götz M., Henkler S., Pimentel A. D., Rettberg A. (eds). *Analysis, Estimations, and Applications of Embedded Systems*. IFIP Advances in Information and Communication Technology. Vol. 576. Cham: Springer, 2023. URL: https://doi.org/10.1007/978-3-031-26500-6_1
24. Donga J., Holia M. S. An analysis of scheduling algorithms in real-time operating system. In: Smys S., Bestak R., Rocha Á. (eds). *Inventive Computation Technologies*. Lecture Notes in Networks and Systems. Vol. 98. Cham: Springer, 2020. URL: https://doi.org/10.1007/978-3-030-33846-6_42
25. Sun Z., Guo M., Liu X. A survey of real-time scheduling on multiprocessor systems. In: Cai Z., Li J., Zhang J. (eds). *Theoretical Computer Science*. Communications

in Computer and Information Science. Vol. 1494. Singapore: Springer, 2021. URL: https://doi.org/10.1007/978-981-16-7443-3_7

26. Cittadini E., Marinoni M., Biondi A. et al. Supporting AI-powered real-time cyber-physical systems on heterogeneous platforms via hypervisor technology. *Real-Time Systems*. 2023. Vol. 59. P. 609–635. URL: <https://doi.org/10.1007/s11241-023-09402-4>

27. Gelgor A. et al. Flexible satellite direct-to-home services with layered-division multiplexing. *IEEE Transactions on Broadcasting*. 2021. Vol. 67, No. 1. P. 83–95. URL: <https://doi.org/10.1109/TBC.2020.3031733>

28. Zhang F., Sun G., Zhou Y., Gao B., Pan S. Towards high-resolution imaging with photonics-based time-division-multiplexing MIMO radar. *IEEE Journal of Selected Topics in Quantum Electronics*. 2022. Vol. 28, No. 5. Article 6000310. URL: <https://doi.org/10.1109/JSTQE.2022.3146862>

29. Zhang L. et al. Layered-division-multiplexing: Theory and practice. *IEEE Transactions on Broadcasting*. 2016. Vol. 62, No. 1. P. 216–232. URL: <https://doi.org/10.1109/TBC.2015.2505408>

30. Schwäricke G., Kloda T., Gracioli G., Bertogna M., Caccamo M. Fixed-priority memory-centric scheduler for COTS-based multiprocessors. In: *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020). Leibniz International Proceedings in Informatics (LIPIcs)*. Vol. 165. Dagstuhl: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. P. 1:1–1:24. URL: <https://doi.org/10.4230/LIPIcs.ECRTS.2020.1>

31. Gu X., Liu P., Yang M., Yang J., Li C., Yao Q. An efficient scheduler of RTOS for multi/many-core system. *Computers & Electrical Engineering*. 2012. Vol. 38, No. 3. P. 785–800. URL: <https://doi.org/10.1016/j.compeleceng.2011.09.009>

32. Mañas-Álvarez F. J., Guinaldo M., Dormido R., Dormido-Canto S. Scalability of cyber-physical systems with real and virtual robots in ROS 2. *Sensors*. 2023. Vol. 23, No. 13. Article 6073. URL: <https://doi.org/10.3390/s23136073>

33. Sagar M., Vanmathi C. A comprehensive review on deep learning techniques on cyber attacks on cyber physical systems. *SN Computer Science*. 2024. Vol. 5. Article 891. URL: <https://doi.org/10.1007/s42979-024-03253-x>
34. Haouari B., Mzid R., Mosbahi O. A reinforcement learning-based approach for online optimal control of self-adaptive real-time systems. *Neural Computing & Applications*. 2023. Vol. 35. P. 20375–20401. URL: <https://doi.org/10.1007/s00521-023-08778-5>
35. LeCun Y., Bengio Y., Hinton G. Deep learning. *Nature*. 2015. Vol. 521. P. 436–444. URL: <https://doi.org/10.1038/nature14539>
36. Shivkumar B., Murphy J., Ziarek L. Real-time MLton: A Standard ML runtime for real-time functional programs. *Journal of Functional Programming*. 2021. Vol. 31. Article e19. URL: <https://doi.org/10.1017/S0956796821000174>
37. Heaton J. Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning. *Genetic Programming and Evolvable Machines*. 2018. Vol. 19. P. 305–307. URL: <https://doi.org/10.1007/s10710-017-9314-z>
38. Gahtan B., Cohen R., Bronstein A. M., Kedar G. Using deep reinforcement learning for mmWave real-time scheduling. *arXiv*. 2023. Article arXiv:2210.01423. URL: <https://doi.org/10.48550/arXiv.2210.01423>
39. Delgadillo O., Blieninger B., Kuhn J., Baumgarten U. A generalistic approach to machine-learning-supported task migration on real-time systems. *Journal of Low Power Electronics and Applications*. 2022. Vol. 12, No. 2. Article 26. URL: <https://doi.org/10.3390/jlpea12020026>
40. Goossens J., Richard P. Multiprocessor real-time scheduling. In: Wang X. (eds). *Cyber-Physical Systems: A Reference*. Berlin, Heidelberg: Springer, 2017. URL: https://doi.org/10.1007/978-3-642-54477-4_2-1
41. Premalatha G., Abinaya S. An effective implementation of vehicle document verification using genetic algorithm. In: *Proceedings of the 2021 International Conference on System, Computation, Automation and Networking (ICSCAN)*. IEEE, 2021. P. 1–6. URL: <https://doi.org/10.1109/ICSCAN53069.2021.9526394>

42. Kabilesh S. K., Stephensagayaraj A., Anandkumar A., Dinakaran K., Mani T., Gokulnath S. Resemblance of real-time scheduling algorithms for real-time embedded systems. *Journal of Optoelectronics and Communication*. 2020. Vol. 2, No. 3. P. 1–8. URL: <https://doi.org/10.5281/zenodo.4311108>
43. Haur I., Béchenec J.-L., Roux O. H. Formal verification of the inter-core synchronization of a multi-core RTOS kernel. In: Riesco A., Zhang M. (eds). *Formal Methods and Software Engineering*. Lecture Notes in Computer Science. Vol. 13478. Cham: Springer, 2022. URL: https://doi.org/10.1007/978-3-031-17244-1_9
44. Liew J. Y., Ng K. H., Khor K. C., Tee K. Y. Evaluating path-finding algorithms for real-time route recommendation system built using FreeRTOS. In: Ghazali R., Nawi N. M., Deris M. M., Abawajy J. H., Arbaiy N. (eds). *Recent Advances on Soft Computing and Data Mining*. Lecture Notes in Networks and Systems. Vol. 1078. Cham: Springer, 2024. URL: https://doi.org/10.1007/978-3-031-66965-1_17
45. Park S., Kwon M.-Y., Kim H.-K., Kim H. Execution model to reduce the interference of shared memory in ARINC 653 compliant multicore RTOS. *Applied Sciences*. 2020. Vol. 10, No. 7. Article 2464. URL: <https://doi.org/10.3390/app10072464>
46. Lazarova-Molnar S., Mohamed N. Reliability analysis of cyber-physical systems. In: Risco Martín J. L., Mittal S., Ören T. (eds). *Simulation for Cyber-Physical Systems Engineering*. Cham: Springer, 2020. URL: https://doi.org/10.1007/978-3-030-51909-4_15
47. Merchant S., Dedhia K. Performance comparison of RTOS. Department of Computer Science, Columbia University, 2001. URL: <https://www.cs.columbia.edu/~sedwards/classes/2001/w4995-02/reports/shamil-kalpen.pdf> (дата звернення: 15.09.2025).
48. Bagula A., Ajayi O., Maluleke H. Cyber physical systems dependability using CPS-IoT monitoring. *Sensors*. 2021. Vol. 21, No. 8. Article 2761. URL: <https://doi.org/10.3390/s21082761>

49. Luna R., Islam S. A. Security and reliability of safety-critical RTOS. *SN Computer Science*. 2021. Vol. 2. Article 356. URL: <https://doi.org/10.1007/s42979-021-00753-y>
50. Jurcut A., Niculcea T., Ranaweera P. et al. Security considerations for Internet of Things: A survey. *SN Computer Science*. 2020. Vol. 1. Article 193. URL: <https://doi.org/10.1007/s42979-020-00201-3>
51. Santos D. A. et al. Hybrid hardening approach for a fault-tolerant RISC-V system-on-chip. *IEEE Transactions on Nuclear Science*. 2024. Vol. 71, No. 8. P. 1722–1730. URL: <https://doi.org/10.1109/TNS.2024.3406021>
52. Savenko O., Sachenko A., Lysenko S., Markowsky G., Vasylykiv N. Botnet detection approach based on the distributed systems. *International Journal of Computing*. 2020. Vol. 19, No. 2. P. 190–198. URL: <https://doi.org/10.47839/ijc.19.2.1761>
53. Lysenko S., Savenko O., Bobrovnikova K., Kryshchuk A., Savenko B. Information technology for botnets detection based on their behaviour in the corporate area network. In: Gaj P., Kwiecień A., Sawicki M. (eds). *Computer Networks. Communications in Computer and Information Science*. Vol. 718. Cham: Springer, 2017. URL: https://doi.org/10.1007/978-3-319-59767-6_14
54. Lysenko S., Savenko O., Bobrovnikova K., Kryshchuk A. Self-adaptive system for the corporate area network resilience in the presence of botnet cyberattacks. In: Gaj P., Sawicki M., Suchacka G., Kwiecień A. (eds). *Computer Networks. Communications in Computer and Information Science*. Vol. 860. Cham: Springer, 2018. URL: https://doi.org/10.1007/978-3-319-92459-5_31
55. Saraeian S., Shirazi B. Digital twin-based fault tolerance approach for cyber–physical production system. *ISA Transactions*. 2022. Vol. 130. P. 35–50. URL: <https://doi.org/10.1016/j.isatra.2022.03.007>
56. Alho P. Service-based fault tolerance for cyber-physical systems: A systems engineering approach. Doctoral dissertation. Tampere University, 2015. URL: <https://doi.org/10.13140/RG.2.1.2862.6003>

57. Reghenzani F., Guo Z., Fornaciari W. Software fault tolerance in real-time systems: Identifying the future research questions. *ACM Computing Surveys*. 2023. Vol. 55, No. 14s. Article 306. P. 1–30. URL: <https://doi.org/10.1145/3589950>
58. Kim B., Yang H. Reliability optimization of real-time satellite embedded system under temperature variations. *IEEE Access*. 2020. Vol. 8. P. 224549–224564. URL: <https://doi.org/10.1109/ACCESS.2020.3044044>
59. Shajahan M. A. Fault tolerance and reliability in AUTOSAR stack development: Redundancy and error handling strategies. *Journal of Embedded Systems and Applications*. 2018. Vol. 3, No. 1. URL: https://hal.science/hal-04561763v1/file/2018_4.pdf (дата звернення: 02.07.2025).
60. Al-Salihi N. K. Improvement of the fault tolerance in IoT-based positioning systems by applying redundancy in the controller layer. *Baghdad Science Journal*. 2021. Vol. 18, No. 4. Article 17. URL: <https://doi.org/10.21123/bsj.2021.18.4.1303>
61. Singh A., D'Souza M., Ebrahim A. Conformance testing of ARINC 653 compliance for a safety critical RTOS using UPPAAL model checker. In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC '21)*. New York: Association for Computing Machinery, 2021. P. 1807–1814. URL: <https://doi.org/10.1145/3412841.3442053>
62. Zagan I., Găitan V. G. Designing a custom CPU architecture based on hardware RTOS and dynamic preemptive scheduler. *Mathematics*. 2022. Vol. 10, No. 15. Article 2637. URL: <https://doi.org/10.3390/math10152637>
63. Aydin H., Melhem R., Mosse D., Mejia-Alvarez P. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*. 2004. Vol. 53, No. 5. P. 584–600. URL: <https://doi.org/10.1109/TC.2004.1275298>
64. Chou Y.-L., Yang J.-M., Wu C.-H. An energy-aware scheduling algorithm under maximum power consumption constraints. *Journal of Manufacturing Systems*. 2020. Vol. 57. P. 182–197. URL: <https://doi.org/10.1016/j.jmsy.2020.09.004>

65. Safari S. et al. A survey of fault-tolerance techniques for embedded systems from the perspective of power, energy, and thermal issues. *IEEE Access*. 2022. Vol. 10. P. 12229–12251. URL: <https://doi.org/10.1109/ACCESS.2022.3144217>
66. Rausand M., Barros A., Hoyland A. *System reliability theory: Models, statistical methods, and applications*. 3rd ed. Hoboken: John Wiley & Sons, 2020. URL: <https://doi.org/10.1002/9781119373940>
67. Rahnamanian M., Ashtiani F. A new analytical approach for delay analysis in the presence of correlated arrivals. In: *Proceedings of the 2024 12th Iran Workshop on Communication and Information Theory (IWCIT)*. IEEE, 2024. P. 1–6. URL: <https://doi.org/10.1109/IWCIT62550.2024.10553217>
68. Zagalo K. Stochastic analysis of stationary real-time systems. Doctoral dissertation. Sorbonne Université, 2023. URL: https://theses.hal.science/tel-04378907v1/file/ZAGALO_Kevin_these_2023.pdf (дата звернення: 15.07.2025).
69. Zagalo K., Abdeddaïm Y., Bar-Hen A., Cucu-Grosjean L. Response time stochastic analysis for fixed-priority stable real-time systems. *IEEE Transactions on Computers*. 2023. Vol. 72, No. 1. P. 3–14. URL: <https://doi.org/10.1109/TC.2022.3211421>
70. Ueter N., Günzel M., Chen J.-J. Response-time analysis and optimization for probabilistic conditional parallel DAG tasks. In: *Proceedings of the 2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021. P. 380–392. URL: <https://doi.org/10.1109/RTSS52674.2021.00042>
71. Sharma S., Chmaj G., Selvaraj H. Applying machine learning to minimize the impact of sensor failures to RTOS-based Internet of Things systems. In: Selvaraj H., Chmaj G., Zydek D. (eds). *Advances in Systems Engineering. Lecture Notes in Networks and Systems*. Vol. 761. Cham: Springer, 2023. URL: https://doi.org/10.1007/978-3-031-40579-2_14
72. Ramani S., Jhaveri R. H. ML-based delay attack detection and isolation for fault-tolerant software-defined industrial networks. *Sensors*. 2022. Vol. 22, No. 18. Article 6958. URL: <https://doi.org/10.3390/s22186958>

73. Bouhata D., Moumen H., Mazari J. A., Bounceur A. Byzantine fault tolerance in distributed machine learning: a survey. *Journal of Experimental & Theoretical Artificial Intelligence*. 2024. Vol. 37, No. 8. P. 1331–1389. URL: <https://doi.org/10.1080/0952813X.2024.2391778>
74. Abeni L., Andreoli R., Gustafsson H., Mini R., Cucinotta T. Fault tolerance in real-time cloud computing. In: *Proceedings of the 2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2023. P. 170–175. URL: <https://doi.org/10.1109/ISORC58943.2023.00031>
75. Bosio A., Di Carlo S., Rebaudengo M., Savino A. Toward the hardening of real-time operating systems. In: *Proceedings of the 2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2022. P. 1–6. URL: <https://doi.org/10.1109/DFT56152.2022.9962356>
76. Magliano E., Savino A., Di Carlo S. Real-time embedded system fault injector framework for micro-architectural state-based reliability assessment. *Journal of Electronic Testing*. 2025. Vol. 41. P. 193–208. URL: <https://doi.org/10.1007/s10836-025-06170-w>
77. Aalund R., Paglioni V. P. Enhancing reliability in embedded systems hardware: A literature survey. *IEEE Access*. 2025. Vol. 13. P. 17285–17302. URL: <https://doi.org/10.1109/ACCESS.2025.3534138>
78. Pomorova O., Savenko O., Lysenko S., Kryshchuk A., Bobrovnikova K. A technique for the botnet detection based on DNS-traffic analysis. In: Gaj P., Kwiecień A., Stera P. (eds). *Computer Networks. Communications in Computer and Information Science*. Vol. 522. Cham: Springer, 2015. P. 133–142. URL: https://doi.org/10.1007/978-3-319-19419-6_12
79. Lysenko S., Pomorova O., Savenko O., Kryshchuk A., Bobrovnikova K. DNS-based anti-evasion technique for botnets detection. In: *Proceedings of the 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. IEEE, 2015. P. 453–458. URL: <https://doi.org/10.1109/IDAACS.2015.7340777>

80. Ukhan Y., Zhuravska I. Forming controlled zones in local wireless computer networks. *Computer-Integrated Technologies: Education, Science, Production*. 2025. No. 59. P. 241–246. URL: <https://doi.org/10.36910/6775-2524-0560-2025-59-30>
81. Boccaccio M., Rachiglia P., Malfense Fierro G. P., Pucillo G. P., Meo M. Deep-subwavelength-optimized holey-structured metamaterial lens for nonlinear air-coupled ultrasonic imaging. *Sensors*. 2021. Vol. 21, No. 4. Article 1170. URL: <https://doi.org/10.3390/s21041170>
82. Аль-Амморі А. Елементи теорії надійності та інформаційної безпеки комп'ютеризованих систем: навч. посіб. Київ: Ліра К, 2024. 282 с. ISBN 978-617-520-755-0
83. Zaytseva T., Kravtsova L., Kaminskaya N. Markov processes in researching the probability of cyber attacks on marine vessels. *Journal of Information Technologies in Education (ITE)*. 2023. No. 52. P. 20–32. URL: <https://doi.org/10.14308/ite000763>
84. Marković F., Roux P., Bozhko S., Papadopoulos A. V., Brandenburg B. B. CTA: A correlation-tolerant analysis of the deadline-failure probability of dependent tasks. In: *Proceedings of the 2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023. P. 317–330. URL: <https://doi.org/10.1109/RTSS59052.2023.00035>
85. Cabral E. A., Tofoli F. L., Sampaio R. F., Leão R. P. S. Reliability assessment applied in the design of an industrial substation in the context of Industry 4.0. *Electric Power Systems Research*. 2024. Vol. 231. Article 110365. URL: <https://doi.org/10.1016/j.epsr.2024.110365>
86. Sagar K. V., Jerald J. Real-time automated guided vehicles scheduling with Markov decision process and double Q-learning algorithm. *Materials Today: Proceedings*. 2022. Vol. 64, No. 1. P. 279–284. URL: <https://doi.org/10.1016/j.matpr.2022.04.522>
87. Liao H.-Y., Cade W., Behdad S. Markov chain optimization of repair and replacement decisions of medical equipment. *Resources, Conservation and Recycling*. 2021. Vol. 171. Article 105609. URL: <https://doi.org/10.1016/j.resconrec.2021.105609>

88. Khan Q. W. Exploring Markov decision processes: A comprehensive survey of optimization applications and techniques. *IgMin Research*. 2024. Vol. 2, No. 7. P. 508–517. URL: <https://doi.org/10.61927/igmin210>
89. Saramud M., Losev V., Karaseva M. Implementation of the $t/(n-1)$ decision-making algorithm by RTOS and FPGA tools. In: *Proceedings of the IV International Scientific Conference on Advanced Technologies in Aerospace, Mechanical and Automation Engineering (MIST: Aerospace-IV 2021)*. AIP Conference Proceedings. 2023. Vol. 2700, No. 1. Article 070005. URL: <https://doi.org/10.1063/5.0126813>
90. Marchetto A., Scanniello G. A rapid review on software vulnerabilities and embedded, cyber-physical, and IoT systems. In: Kadgien R., Jedlitschka A., Janes A., Lenarduzzi V., Li X. (eds). *Product-Focused Software Process Improvement (PROFES 2023)*. Lecture Notes in Computer Science. Vol. 14483. Cham: Springer, 2024. P. 456–469. URL: https://doi.org/10.1007/978-3-031-49266-2_32
91. Palleti V., Adepu S., Mishra V. Cascading effects of cyber-attacks on interconnected critical infrastructure. *Cybersecurity*. 2021. Vol. 4. Article 8. URL: <https://doi.org/10.1186/s42400-021-00071-z>
92. Wani J. A., Sing S., Malik A., Singh P. K. An analysis study of security threats, vulnerabilities, and challenges of cyber-physical systems. In: Singh P. K., Wierzchoń S. T., Pawłowski W., Kar A. K., Kumar Y. (eds). *IoT, Big Data and AI for Improving Quality of Everyday Life: Present and Future Challenges*. Studies in Computational Intelligence. Vol. 1104. Cham: Springer, 2023. P. 141–155. URL: https://doi.org/10.1007/978-3-031-35783-1_9
93. Cho C., Seong Y., Won Y. Mandatory access control method for Windows embedded OS security. *Electronics*. 2021. Vol. 10, No. 20. Article 2478. URL: <https://doi.org/10.3390/electronics10202478>
94. Duo W., Zhou M., Abusorrah A. A survey of cyber attacks on cyber-physical systems: Recent advances and challenges. *IEEE/CAA Journal of Automatica Sinica*. 2022. Vol. 9, No. 5. P. 784–800. URL: <https://doi.org/10.1109/JAS.2022.10554>

95. Valade A., Acco P., Grabolosa P., Fourniols J.-Y. A study about Kalman filters applied to embedded sensors. *Sensors*. 2017. Vol. 17, No. 12. Article 2810. URL: <https://doi.org/10.3390/s17122810>
96. Rostami S. M. H., Pourgholi M., Asharioun H. Enhancing resilience of distributed DC microgrids against cyber attacks using a transformer-based Kalman filter estimator. *Scientific Reports*. 2025. Vol. 15. Article 6815. URL: <https://doi.org/10.1038/s41598-025-90959-4>
97. Tummala A. S. L. V., Inapakurthi R. K. A two-stage Kalman filter for cyber-attack detection in automatic generation control system. *Journal of Modern Power Systems and Clean Energy*. 2022. Vol. 10, No. 1. P. 50–59. URL: <https://doi.org/10.35833/MPCE.2019.000119>
98. Wen F., Wang Z. Distributed Kalman filtering for robust state estimation over wireless sensor networks under malicious cyber attacks. *Digital Signal Processing*. 2018. Vol. 78. P. 92–97. URL: <https://doi.org/10.1016/j.dsp.2018.03.002>
99. Karimipour H., Leung H. Relaxation-based anomaly detection in cyber-physical systems using ensemble Kalman filter. *IET Cyber-Physical Systems: Theory & Applications*. 2020. Vol. 5, No. 1. P. 49–58. URL: <https://doi.org/10.1049/iet-cps.2019.0031>
100. Soule A., Salamatian K., Taft N. Combining filtering and statistical methods for anomaly detection. In: *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement (IMC '05)*. New York: Association for Computing Machinery, 2005. P. 31–36. URL: https://www.usenix.org/legacy/event/imc05/tech/full_papers/soule/soule.pdf (дата звернення: 18.09.2025).
101. Ahmad R., Alkhamash E. H. Online adaptive Kalman filtering for real-time anomaly detection in wireless sensor networks. *Sensors*. 2024. Vol. 24, No. 15. Article 5046. URL: <https://doi.org/10.3390/s24155046>
102. Liang C., Wen F., Wang Z. Trust-based distributed Kalman filtering for target tracking under malicious cyber attacks. *Information Fusion*. 2019. Vol. 46. P. 44–50. URL: <https://doi.org/10.1016/j.inffus.2018.04.002>

103. Nkemeni V., Mieyeville F., Tsafack P. A distributed computing solution based on distributed Kalman filter for leak detection in WSN-based water pipeline monitoring. *Sensors*. 2020. Vol. 20, No. 18. Article 5204. URL: <https://doi.org/10.3390/s20185204>
104. Mustafa A., Mazouchi M., Modares H. Secure event-triggered distributed Kalman filters for state estimation over wireless sensor networks. *arXiv*. 2021. Article arXiv:1901.06746. URL: <https://doi.org/10.48550/arXiv.1901.06746>
105. Jihani N., Kabbaj M. N., Benbrahim M. Kalman filter-based sensor fault detection in wireless sensor network for smart irrigation. *Results in Engineering*. 2023. Vol. 20. Article 101395. URL: <https://doi.org/10.1016/j.rineng.2023.101395>
106. Bosman H. H. W. J., Iacca G., Tejada A., Wörtche H. J., Liotta A. Spatial anomaly detection in sensor networks using neighborhood information. *Information Fusion*. 2017. Vol. 33. P. 41–56. URL: <https://doi.org/10.1016/j.inffus.2016.04.007>
107. Lu X., Wang J., Liu G., Du W., Yang D. Station-and-network-coordinated planning of integrated energy system considering integrated demand response. *Global Energy Interconnection*. 2021. Vol. 4, No. 1. P. 39–47. URL: <https://doi.org/10.1016/j.gloi.2021.03.004>
108. Liu N., Xie Y., Su Z., Zhao Z., Wang W. Adaptive Kalman filter-integrated navigation measurement using inertial sensor for vehicle motion state recognition. *Measurement*. 2025. Vol. 248. Article 116907. URL: <https://doi.org/10.1016/j.measurement.2025.116907>
109. Варанкін Д. В., Журавська І. М., Обухова К. О. Система дальнього радіозв'язку для контролю датчиків на основі одноплатних мікрокомп'ютерів з LoRa. *Methods and Devices of Quality Control*. 2024. No. 2(53). P. 121–129. URL: [https://doi.org/10.31471/1993-9981-2024-2\(53\)-121-129](https://doi.org/10.31471/1993-9981-2024-2(53)-121-129)
110. Koller T. L., Frese U. The interacting multiple model filter and smoother on boxplus-manifolds. *Sensors*. 2021. Vol. 21, No. 12. Article 4164. URL: <https://doi.org/10.3390/s21124164>

111. Radhika M. N., Mallick M., Tian X. IMM filtering algorithms for a highly maneuvering fighter aircraft: An overview. *Algorithms*. 2024. Vol. 17, No. 9. Article 399. URL: <https://doi.org/10.3390/a17090399>
112. Chen C., Zhou W., Gao L. A novel robust IMM filtering method for surface-maneuvering target tracking with random measurement delay. *Journal of Marine Science and Engineering*. 2023. Vol. 11, No. 5. Article 1047. URL: <https://doi.org/10.3390/jmse11051047>
113. Barber R., Rosa D. R., Flores-Caballero A., Garrido S. Deployment of model-based-design-adaptive controllers for monitoring and control mechatronic devices. *Applied Sciences*. 2023. Vol. 13, No. 22. Article 12432. URL: <https://doi.org/10.3390/app132212432>
114. Li Y. Adaptive H_∞ Kalman filtering for stochastic systems with nonlinear uncertainties. In: Samui P., Bui D. T., Chakraborty S., Deo R. C. (eds). *Handbook of Probabilistic Models*. Butterworth-Heinemann, 2020. P. 243–264. URL: <https://doi.org/10.1016/B978-0-12-816514-0.00010-2>
115. Elsisì M., Altius M., Su S.-F., Su C.-L. Robust Kalman filter for position estimation of automated guided vehicles under cyberattacks. *IEEE Transactions on Instrumentation and Measurement*. 2023. Vol. 72. Article 1002612. P. 1–12. URL: <https://doi.org/10.1109/TIM.2023.3250285>
116. Emanuele A., Gasparotto F., Guerra G., Zorzi M. Robust distributed Kalman filtering: On the choice of the local tolerance. *Sensors*. 2020. Vol. 20, No. 11. Article 3244. URL: <https://doi.org/10.3390/s20113244>
117. Yang Y., Li Z., Zhao X., Li Y. Worst-case performance of Kalman filtering against cyber-attacks with true and contaminated information. *Automatica*. 2025. Vol. 176. Article 112257. URL: <https://doi.org/10.1016/j.automatica.2025.112257>
118. Slomka F., Sadeghi M. Beyond the limitations of real-time scheduling theory: A unified scheduling theory for the analysis of real-time systems. *Software-Intensive Cyber-Physical Systems*. 2021. Vol. 35. P. 201–236. URL: <https://doi.org/10.1007/s00450-021-00429-1>

119. Gotz M., Dittmann F., Pereira C. E. Deterministic mechanism for run-time reconfiguration activities in an RTOS. In: *Proceedings of the 4th IEEE International Conference on Industrial Informatics (INDIN 2006)*. IEEE, 2006. P. 693–698. URL: <https://doi.org/10.1109/INDIN.2006.275645>
120. Godunov A. N., Khomenkov I. I., Shchepkov V. G. Configurable test system for RTOS. *Programming and Computer Software*. 2023. Vol. 49. P. 286–294. URL: <https://doi.org/10.1134/S0361768823040059>
121. Hahm S.-I. et al. Reliable real-time operating system for IoT devices. *IEEE Internet of Things Journal*. 2021. Vol. 8, No. 5. P. 3705–3716. URL: <https://doi.org/10.1109/JIOT.2020.3025612>
122. Ровінський В. А., Євчук О. В., Ровінський Ю. В. Вибір операційних систем реального часу при розробці пристроїв для технічної діагностики. *Методи та прилади контролю якості*. 2023. No. 1(48). P. 66–77. URL: [https://doi.org/10.31471/1993-9981-2022-1\(48\)-66-77](https://doi.org/10.31471/1993-9981-2022-1(48)-66-77)
123. What are the most popular real-time operating systems in 2025? *Lynx Software Technologies*. URL: <https://www.lynx.com/blog/most-popular-real-time-operating-systems-rtos> (дата звернення: 28.05.2025).
124. Deos RTOS software: Safety critical multicore RTOS | DAL A, DO-178C, FACE. *DDC-I, Inc.* URL: <https://www.ddci.com/solutions/products/deos/> (дата звернення: 11.02.2025)
125. Amin M. R., Hasan M., Arnab S. P., DeGiorgio M. Tensor decomposition-based feature extraction and classification to detect natural selection from genomic data. *Molecular Biology and Evolution*. 2023. Vol. 40, No. 10. Article msad216. URL: <https://doi.org/10.1093/molbev/msad216>
126. Boyd S. P., Vandenberghe L. *Convex optimization*. Version 29. Cambridge: Cambridge University Press, 2023. URL: <https://www.cse.iitd.ac.in/~naveen/courses/CSL866/bvBook.pdf> (дата звернення: 18.05.2025).

127. Bertsekas D. P. *Nonlinear programming*. Belmont: Athena Scientific, 1999. URL: https://web.mit.edu/dimitrib/www/NLP_Appendixes_Refs.pdf (дата звернення: 18.05.2025).
128. Udayakumar K., Ramamoorthy S. Adaptive resource allocation for time-critical applications in industrial edge computing using heuristic-HCMDO approach. *Research Square*. 2023. URL: <https://doi.org/10.21203/rs.3.rs-3150864/v1>
129. Braverman V. Sliding window algorithms. In: Kao M. Y. (eds). *Encyclopedia of Algorithms*. New York: Springer, 2016. URL: https://doi.org/10.1007/978-1-4939-2864-4_797
130. Ding B., Xu Z., Zhang T. Fault estimation and fault tolerant control for linear stochastic uncertain systems. *Asian Journal of Control*. 2023. Vol. 25, No. 1. P. 475–484. URL: <https://doi.org/10.1002/asjc.2787>
131. Gillijns S., De Moor B. Unbiased minimum-variance input and state estimation for linear discrete-time systems with direct feedthrough. *Automatica*. 2007. Vol. 43, No. 5. P. 934–937. URL: <https://doi.org/10.1016/j.automatica.2006.11.016>
132. Fang H., Haile M. A., Wang Y. Robustifying the Kalman filter against measurement outliers: An innovation saturation mechanism. In: *Proceedings of the 2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018. P. 6390–6395. URL: <https://doi.org/10.1109/CDC.2018.8619140>
133. Montgomery D. C., Runger G. C. *Applied statistics and probability for engineers*. 5th ed. Hoboken: John Wiley & Sons, 2010. 784 p. URL: https://books.google.com.ua/books/about/Applied_Statistics_and_Probability_for_E.html?id=_f4KrEcNAfEC (дата звернення: 11.04.2025).
134. Tanenbaum A. S., Bos H. *Modern operating systems*. 4th ed. Boston: Pearson, 2015. URL: <https://os.ecci.ucr.ac.cr/slides/Andrew-S.-Tanenbaum-Modern-Operating-Systems.pdf> (дата звернення: 15.11.2025).
135. Stallings W. *Operating systems: Internals and design principles*. 8th ed. Boston: Pearson, 2014. URL:

<https://dl.ebooksworld.ir/motoman/Pearson.Operating.Systems.Internals.and.Design.Principles.8th.Edition.www.EBooksWorld.ir.pdf> (дата звернення: 15.11.2025).

136. Козельський О., Савенко Б. Виявлення зловмисних атак на сенсори та підробки телеметрії в кіберфізичних системах на основі модифікованого фільтра Калмана. *Measuring and computing devices in technological processes*. 2025. №84(4). С. 228–235. URL: <https://doi.org/10.31891/2219-9365-2025-84-24>

137. Gaj P., Sochor T., Korobchynskiy M., Savenko B., Kozelskyi O. Hybrid Method for Protecting RTOS from Failures and Cyberattacks Using Compact Markov Models. *The International Workshop on Applied Intelligent Security Systems in Law Enforcement (AISSLE-2025), Vinnytsia, Ukraine, October, 30 – November, 02, 2025*. Vol. 4126. pp. 361-376. URL: <https://ceur-ws.org/Vol-4126/paper19.pdf> (Scopus)

138. Свідоцтво про реєстрацію авторського права на твір №141525. Комп'ютерна програма «Програмне забезпечення аналізатора імовірності відмов у реальному часі для операційних систем кіберфізичних систем з інтенсивними змінами станів (АІВРЧ)» / О.В. Козельський, Б.О. Савенко. Дата реєстрації 17.12.2025. URL: <https://sis.nipo.gov.ua/uk/search/detail/1897556/>

139. Козельський О.В. Модель системи адаптивної кластеризації даних із зовнішнім модулем аналізу для архітектури ОС реального часу при динамічних змінах станів. *Актуальні проблеми комп'ютерних наук: Збірник наукових праць, Матеріали XVI Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2024»*, Україна, Хмельницький, ХНУ, листопад 15-16, 2024. С. 272 – 274. URL: <https://kn.khmnu.edu.ua/wp-content/uploads/sites/18/apkn-2024-corpuspaper.pdf>

140. Zhou D. H., Frank P. M. Strong tracking filtering of nonlinear time-varying stochastic systems with coloured noise: application to parameter estimation and empirical robustness analysis. *International Journal of Control*. 1996. Vol. 65, No. 2. P. 295–307. URL: <https://doi.org/10.1080/00207179608921698>

141. Журавська І. М., Тонха О. Л., Нечай І. В., Гуменюк Т. В. Керування групою БПЛА із застосуванням комп'ютерного зору в просторовому коридорі на базі гетерогенної мережі. *Perspective Technologies and Devices*. 2025. Vol. 1, No. 27. P. 58–69. URL: <https://doi.org/10.36910/10.36910/6775-2313-5352-2025-27-8>
142. Lysenko S., Savenko O., Bobrovnikova K., Kryshchuk A. Self-adaptive system for the corporate area network resilience in the presence of botnet cyberattacks. In: Gaj P., Sawicki M., Suchacka G., Kwiecień A. (eds). *Computer Networks. Communications in Computer and Information Science*. Vol. 860. Cham: Springer, 2018. P. 365–376. URL: https://doi.org/10.1007/978-3-319-92459-5_31
143. Huang P., Li H., Wen G., Wang Z. Application of adaptive weighted strong tracking unscented Kalman filter in non-cooperative maneuvering target tracking. *Aerospace*. 2022. Vol. 9, No. 8. Article 468. URL: <https://doi.org/10.3390/aerospace9080468>
144. Kronauer T., Pohlmann J., Matthé M., Smejkal T., Fettweis G. Latency analysis of ROS2 multi-node systems. In: *Proceedings of the 2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. IEEE, 2021. P. 1–7. URL: <https://doi.org/10.1109/MFI52462.2021.9591166>
145. Ranjbar B., Alikhani H., Safaei B., Ejlali A., Kumar A. Learning-oriented QoS- and drop-aware task scheduling for mixed-criticality systems. *Computers*. 2022. Vol. 11, No. 7. Article 101. URL: <https://doi.org/10.3390/computers11070101>
146. Lo D., Ismail M., Chen T., Suh G. E. Slack-aware opportunistic monitoring for real-time systems. In: *Proceedings of the 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2014. P. 203–214. URL: <https://doi.org/10.1109/RTAS.2014.6926003>
147. Wang Y., Li A., Wang J., Baruah S., Zhang N. Opportunistic data flow integrity for real-time cyber-physical systems using worst-case execution time reservation. In: *Proceedings of the 33rd USENIX Security Symposium (SEC '24)*. 2024. P. 6615–6632. URL: <https://doi.org/10.1145/3728364>

148. Loutsikii H., Volokyta A., Yakushev O., Rehida P., Thinh V. D. Development of real time method of detecting attacks based on artificial intelligence. *Technology Audit and Production Reserves*. 2016. Vol. 3, No. 1(29). P. 40–46. URL: <https://doi.org/10.15587/2312-8372.2016.71677>
149. Петришин О., Монастирський Л., Власов А. Використання операційних систем реального часу при проектуванні та розробці системи збору даних акселерометрії. In: *Матеріали IV Міжнародної науково-технічної конференції «Теоретичні та прикладні аспекти радіотехніки, приладобудування і комп'ютерних технологій»*. Тернопіль, 2019. С. 220–221. URL: <https://elartu.tntu.edu.ua/handle/lib/28836> (дата звернення: 09.01.2026)
150. Letichevsky A. A., Letychevskiy O. O., Skobelev V. G., Volkov V. A. Cyber-physical systems. *Cybernetics and Systems Analysis*. 2017. Vol. 53, No. 6. P. 821–834. URL: <https://doi.org/10.1615/JAutomatInfScien.v49.i9.10>
151. Kanarskyi Y., Kharchenko V., Orekhov O., Ponochovnyi Y. Markov modelling of human-machine interaction in an augmented reality environment for UAV/UGV-based hazardous area monitoring systems. *Radioelectronic and Computer Systems*, 2025(4), Pp. 35-54. DOI: <https://doi.org/10.32620/reks.2025.4.03>
152. Moskalenko V., Kharchenko V., Semenov S. Model and Method for Providing Resilience to Resource-Constrained AI-System. *Sensors*, 2024. 24(18), 5951. DOI: <https://doi.org/10.3390/s24185951>
153. De Lathauwer L., De Moor B., Vandewalle J. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 2000, Vol. 21, No. 4, Pp. 1253–1278. DOI: <https://doi.org/10.1137/S0895479896305696>

ДОДАТОК А

Список публікацій здобувача

Наукові праці, в яких опубліковані основні наукові результати дисертації

1. Козельський О., Савенко Б. Виявлення фальсифікацій та аномалій у цифрових сигналах від кіберфізичних систем на основі комбінованого фільтра Калмана. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 2025. №2. С. 23–35. DOI: <https://doi.org/10.32782/IT/2025-2-4>

2. Козельський О.В. Метод тензорної декомпозиції для адаптивного розподілу ресурсів у системах реального часу. *Measuring and computing devices in technological processes*. 2025. №82(2). С. 426-433. DOI: <https://doi.org/10.31891/2219-9365-2025-82-61>

3. Козельський О., Савенко Б., Савенко О. Дворівнева стратегія підвищення відмовостійкості операційних систем реального часу з використанням ймовірнісного аналізу. *Herald of Khmelnytskyi National University. Technical Sciences*. 2025. №353(3.2). С. 438-446. DOI: <https://doi.org/10.31891/2307-5732-2025-353-60>

4. Козельський О., Савенко Б. Зовнішня адаптивна система кластеризації для підвищення гнучкості операційних систем реального часу на основі динамічного планування завдань. *Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки*. Том 5. 2025. Ч. 2. С. 124-132 DOI: <https://doi.org/10.32782/2663-5941/2025.5.2/18>

5. Козельський О., Савенко О. Проактивний механізм інформаційної безпеки в операційних системах реального часу з використанням гібридного сторожового таймера. *Measuring and computing devices in technological processes*. 2025. №83(3). С. 459 – 466. DOI: <https://doi.org/10.31891/2219-9365-2025-83-57>

6. Козельський О., Савенко Б. Виявлення зловмисних атак на сенсори та підробки телеметрії в кіберфізичних системах на основі модифікованого фільтра Калмана. *Measuring and computing devices in technological processes*. 2025. №84(4). С.

228–235. DOI: <https://doi.org/10.31891/2219-9365-2025-84-24>

Праці, які засвідчують апробацію матеріалів дисертації

7. Kozelskiy O., Kashtalian A., Stetsyuk V., Martiniuk D., Sachenko A. A model of an intelligent clustering system with an external module for the architecture of RTOS with intensive changes of states regarding their flexibility and balancing. *1st International Workshop on Advanced Applied Information Technologies: (AdvAIT-2024)*, Khmelnytskyi, Ukraine and Zilina, Slovakia, December 5, 2024. Vol. 3899. P. 234-243. URL: <https://ceur-ws.org/Vol-3899/paper21.pdf> (Scopus)

8. Kozelskiy O., Drozd A., Savenko B., Gaj P. A model for probabilistic monitoring and proactive restart of real-time operating systems under intensive state changes in cyber-physical systems. *2nd International Workshop on Intelligent & CyberPhysical Systems: (ICyberPhyS 2025)*, Khmelnytskyi, Ukraine, July 4, 2025. Vol. 4013. P. 198-210. URL: <https://ceur-ws.org/Vol-4013/paper16.pdf> (Scopus)

9. Gaj P., Sochor T., Korobchynskyi M., Savenko B., Kozelskiy O. Hybrid Method for Protecting RTOS from Failures and Cyberattacks Using Compact Markov Models. *The International Workshop on Applied Intelligent Security Systems in Law Enforcement (AISSLE-2025)*, Vinnytsia, Ukraine, October, 30 – November, 02, 2025. Vol. 4126. pp. 361-376. URL: <https://ceur-ws.org/Vol-4126/paper19.pdf>

Публікації, які додатково відображають наукові результати дисертації

10. Свідоцтво про реєстрацію авторського права на твір №141525. Комп'ютерна програма «Програмне забезпечення аналізатора імовірності відмов у реальному часі для операційних систем кіберфізичних систем з інтенсивними змінами станів (АІВРЧ)» / О.В. Козельський, Б.О. Савенко. Дата реєстрації 17.12.2025. URL: <https://sis.nipo.gov.ua/uk/search/detail/1897556>

ДОДАТОК Б

Акти впровадження

ЗАТВЕРДЖУЮ

Директор ТОВ Ультра ІТ



Гудзь С. В.

16 березня 2026 року

АКТ

Про впровадження результатів дисертаційної роботи

Козельського Олександра Володимировича

«Методи та засоби планування задач і підвищення ефективності операційних систем реального часу»

Результати дисертаційної роботи аспіранта Козельського О.В. кафедри комп'ютерної інженерії та інформаційних систем Хмельницького національного університету впроваджені на ТОВ «Ультра ІТ».

В процесі впровадження методів та засобів забезпечення безпеки спеціалізованих мережних ОС в умовах впливів зловмисного програмного забезпечення були використанні на ТОВ «Ультра ІТ» результати, які одержані Козельським О.В. особисто:

1) дворівневу модель забезпечення відмовостійкості, яка поєднує апаратний контроль працездатності кіберфізичної системи з програмним предиктивним модулем на основі стохастичного аналізу стані;

2) метод на основі комбінованого фільтра Калмана, який забезпечує стійку оцінку станів операційної системи реального часу та її компонентів в умовах стохастичних шумів, невизначеностей і можливих зловмисних впливів на інформаційні канали. Метод поєднує класичну процедуру фільтрації з подієвим перемиканням режимів і механізмом контролю достовірності вимірювань, що дає змогу відокремлювати реальні зміни стану системи від аномалій або фальсифікованих сигналів. Запропонований підхід дозволяє своєчасно виявляти порушення у поведінці сенсорних або функціональних підсистем, ізолювати потенційно небезпечні дані, запобігати їх поширенню в системі керування та зберігати коректність оцінок навіть за наявності зовнішніх або цілеспрямованих збурень.

Отримані результати дозволили покращити захищеність кіберфізичних систем на основі ОС реального часу з інтенсивними змінами станів в умовах впливів зловмисного програмного забезпечення та комп'ютерних атак, використовуваних на ТОВ «Ультра ІТ» в.

Результати роботи використано для організації захисту використовуваних на ТОВ «Ультра ІТ» інформаційних технологій від впливів зловмисного програмного забезпечення та комп'ютерних атак.

Цей акт не є підставою для фінансових розрахунків.

Бухгалтер (фінансист)



ЗАТВЕРДЖУЮ

Директор ТОВ ДЕВІКС ДІДЖИТАЛ

 Гуцал Ю.В.13  2026 р.

Код ЄДРПОУ: 44713336

Сайт: <https://ua.linkedin.com/company/devixdigital>Юридична адреса: Україна, Хмельницька обл., Хмельницький р-н, місто Хмельницький,
вул. Нижня Берегова, будинок 45А**АКТ**

Про впровадження результатів дисертаційної роботи

Козельського Олександра Володимировича

«Методи та засоби планування задач і підвищення ефективності операційних систем
реального часу»

Результати дисертаційної роботи аспіранта кафедри комп'ютерної інженерії та інформаційних систем Хмельницького національного університету впроваджені на ТОВ «ДЕВІКС ДІДЖИТАЛ».

В процесі впровадження методів та засобів забезпечення безпеки кіберфізичних систем на базі ОС реального часу з інтенсивними змінами станів в умовах впливів зловмисного програмного забезпечення були використанні на ТОВ «ДЕВІКС ДІДЖИТАЛ» результати, які одержані Козельським О.В. особисто:

1) модель архітектури планувальника задач з інтелектуальною кластеризацією, що базується на принципі збирання наборів даних під час роботи ОСРЧ, їх передавання до зовнішнього аналітичного модуля машинного навчання, аналізу тензорних представлень і подальшого оновлення параметрів розподілу ресурсів у планувальнику із зворотним каналом самооптимізації, що дозволяє системі покращувати ефективність управління ресурсами у наступних циклах без втручання користувача.

2) дворівневу модель забезпечення відмовостійкості, яка поєднує апаратний контроль працездатності з програмним предиктивним модулем на основі стохастичного аналізу станів. Запропонований підхід реалізує принцип проактивного моніторингу, що дозволяє виявляти тенденції до деградації роботи системи,

оцінювати ймовірність виникнення збоїв і здійснювати локальне відновлення окремих компонентів до переходу у критичний стан, забезпечуючи безперервність функціонування ОСРЧ.

Отримані результати дозволили на основі запропонованих моделей розробити покращені алгоритми захисту кіберфізичних систем з інтенсивними змінами станів з ОС реального часу, використовуваних на ТОВ «ДЕВІКС ДІДЖИТАЛ» в умовах впливів зловмисного програмного забезпечення та комп'ютерних атак.

Цей акт не є підставою для фінансових розрахунків.

Бухгалтер (фінансист)



«Затверджую»

Технічний директор ТОВ «Nolt technologies»

Мельниченко О.В.

13» березня 2026 р.

АКТ



про впровадження результатів дисертаційної роботи
аспіранта кафедри комп'ютерної інженерії та інформаційних систем
Хмельницького національного університету Козельського Олександра
Володимировича

«Методи та засоби планування задач і підвищення ефективності операційних
систем реального часу»

Результати дисертаційної роботи аспіранта кафедри комп'ютерної інженерії та інформаційних систем Хмельницького національного університету Козельського О.В. впроваджені на ТОВ «Nolt technologies».

У процесі впровадження на ТОВ «Nolt technologies» методів та засобів забезпечення безпеки ОСРЧ в умовах впливу зловмисного програмного забезпечення, а також підвищення відмовостійкості та надійності її функціонування, були використані результати, одержані Козельським О.В. особисто, а саме:

1) метод превентивного відновлення компонентів ОСРЧ, який поєднує низькорозмірну марковську модель прогнозування станів із багаторівневим сторожовим контролем, що на відміну від традиційних watchdog-схем, які реагують лише після зупинки або зависання, дозволяє завчасно оцінювати ризик відмови та локально програмно ініціювати апаратний перезапуск з формалізацією індексу стану, перехідних ймовірностей та політик ескалації, зберігаючи при цьому низьку обчислювальну складність, придатну для мікроконтролерів у реальному часі;

2) метод виявлення фальсифікацій та аномалій у сигналах кіберфізичних систем на основі модифікованого комбінованого фільтра Калмана з подієвим перемиканням режимів, який, на відміну від поширених багатомодельних або адаптивних фільтрів, не потребує одночасного запуску набору моделей і забезпечує селективне пригнічення підозрілих вимірів на основі аналізу

інновацій.

Отримані результати забезпечили підвищення стійкості та надійності обробки й використання телеметричних даних у кіберфізичних системах в умовах зловмисних впливів і комп'ютерних атак, а також стабільність функціонування в операційних системах реального часу в умовах інтенсивних змін станів, що використовуються на ТОВ «Nolt technologies». Результати роботи використано для організації захисту використовуваних на ТОВ «Nolt technologies» інформаційних технологій в умовах впливу зловмисного програмного забезпечення та комп'ютерних атак.

Цей акт не є підставою для фінансових розрахунків.

Директор



Мельник Олександр

«Затверджую»



Проректор з науково-педагогічної роботи

Віктор ЛОПАТОВСЬКИЙ

2026 року

АКТ

Про впровадження результатів дисертаційної роботи
Козельського Олександра Володимировича
«Методи та засоби планування задач і підвищення ефективності операційних систем реального часу»

Ми, комісія складі: декана факультету інформаційних технологій д.т.н. професора Говорущенко Т.О., завідувача кафедрою комп'ютерної інженерії та інформаційних систем д.філ. Павлової О.О., професора кафедри, д.т.н професора Лисенко С.М. склала акт про те результати дисертаційної роботи Козельського О.В. впроваджені та використовуються на кафедрі комп'ютерної інженерії та інформаційних систем для спеціальності 123 «Комп'ютерна інженерія» галузі знань 12 «Інформаційні технології», зокрема при викладанні освітніх компонентів «Технічна діагностика і надійність комп'ютерних пристроїв» та «Безпека та захист комп'ютерних систем».

При викладанні цих дисциплін використовуються наступні матеріали досліджень, одержані Козельським О.В. особисто:

а) модель архітектури планувальника задач з інтелектуальною кластеризацією, що базується на принципі збирання наборів даних під час роботи ОСРЧ, їх передавання до зовнішнього аналітичного модуля машинного навчання, аналізу тензорних представлень і подальшого оновлення параметрів розподілу ресурсів у планувальнику.

б) метод інтелектуальної кластеризації на основі тензорної декомпозиції, який дозволяє здійснювати багатовимірний аналіз взаємозв'язків між задачами, ресурсами та часовими характеристиками операційної системи реального часу.

Отримані Козельським О.В. матеріали досліджень надали можливість розробити лабораторні практикуми з використанням розроблених моделей та методів для забезпечення безпеки кіберфізичних систем з інтенсивними змінами станів.

Тетяна ГОВОРУЩЕНКО

Ольга ПАВЛОВА

Сергій ЛИСЕНКО



ГОЛОВНЕ УПРАВЛІННЯ РОЗВІДКИ
МІНІСТЕРСТВА ОБОРОНИ УКРАЇНИ

ВОЄННА АКАДЕМІЯ

ІМЕНІ ЄВГЕНІЯ БЕРЕЗНЯКА

Код 14303342

«18» 12 2025 р.

№ 222/BA/2164

04050, м. Київ

ЗАТВЕРДЖУЮ

Тимчасово виконуючий обов'язки
начальника Воєнної академії
імені Євгенія Березняка
полковник

Вячеслав КОЛОТОВ



впровадження результатів дисертаційної роботи

«Методи та засоби планування задач і підвищення ефективності операційних
систем реального часу»

Козельського Олесандра Володимировича

Комісія в складі: голова комісії – начальник Другого навчально-наукового інституту Воєнної академії імені Євгенія Березняка, кандидат технічних наук, доцент, полковник Чумак Олександр Ілліч; члени комісії: заступник начальника другої кафедри Другого навчально-наукового інституту Воєнної академії імені Євгенія Березняка, кандидат технічних наук, доцент, полковник Руденко Михайло Миколайович; доцент другої кафедри Другого навчально-наукового інституту Воєнної академії імені Євгенія Березняка, кандидат технічних наук, доцент, полковник Гончарук Андрій Вікторович; старший викладач другої кафедри Другого навчально-наукового інституту Воєнної академії імені Євгенія Березняка, кандидат технічних наук, старший науковий співробітник, полковник Волков Олександр Віталійович, склала цей акт про те, що результати дисертаційної роботи здобувача наукового ступеня доктора філософії Козельського О.В. впроваджені в освітній процес у блоці військово-спеціальних дисциплін та використані при удосконаленні навчально-лабораторного комплексу другої кафедри Другого навчально-наукового інституту Воєнної академії імені Євгенія Березняка.

До переліку впроваджених та використаних результатів дисертаційної роботи Козельського О.В. належать:

1) архітектура планувальника задач з інтелектуальною кластеризацією, яка базується на принципі збирання наборів даних під час роботи операційної

системи реального часу, їх передавання до зовнішнього аналітичного модуля машинного навчання, аналізу тензорних представлень і подальшого оновлення параметрів розподілу ресурсів у планувальнику, що дозволяє системі покращувати ефективність управління ресурсами у майбутніх циклах без втручання користувача;

2) метод інтелектуальної кластеризації на основі тензорної декомпозиції, який дозволяє здійснювати багатовимірний аналіз взаємозв'язків між задачами, ресурсами та часовими характеристиками операційної системи реального часу, що забезпечує виявлення прихованих закономірностей у навантаженні системи, формування оптимальних груп задач і адаптивне коригування стратегії розподілу ресурсів з метою підвищення загальної ефективності функціонування операційної системи реального часу;

3) дворівнева модель забезпечення відмовостійкості, яка поєднує апаратний контроль працездатності з програмним предикативним модулем на основі стохастичного аналізу станів і, при цьому, реалізує принцип проактивного моніторингу, що дозволяє виявляти тенденції до деградації роботи системи, оцінювати ймовірність виникнення збоїв і здійснювати локальне відновлення окремих компонентів до переходу у критичний стан, забезпечуючи безперервність функціонування операційної системи реального часу.

Впровадження та використання зазначених результатів дозволили підвищити якість викладання військово-спеціальних дисциплін та удосконалити навчально-лабораторний комплекс другої кафедри Другого навчально-наукового інституту Воєнної академії імені Євгенія Березняка.

Голова комісії: полковник

Члени комісії: полковник

полковник

полковник

Олександр ЧУМАК

Михайло РУДЕНКО

Андрій ГОНЧАРУК

Олександр ВОЛКОВ

ДОДАТОК В

Лістинг псевдокоду та програмного коду менеджера для роботи з бібліотекою
TensorFlow

КЛАС TensorFlowManager РЕАЛІЗУЄ IDisposable

ПРИВАТНІ ЗМІННІ:

session

graph

x, y, W, b, pred, loss, optimizer

МЕТОД TensorFlowManager()

Створити новий граф

Ініціалізувати сесію з цим графом

// Побудова графа моделі

x ← створити placeholder для вхідних даних

y ← створити placeholder для вихідних даних

W ← створити змінну для ваг

b ← створити змінну для зміщення

pred ← виконати матричне множення x та W, додати b

// Функція втрат

loss ← обчислити середньоквадратичну похибку між pred і y

// Оптимізатор

optimizer ← створити алгоритм градієнтного спуску з швидкістю 0.01

// Ініціалізувати всі змінні в сесії

Виконати `session.run(ініціалізація змінних)`

МЕТОД `Train(xData, yData, epochs = 1000)`

`xArray` ← перетворити `xData` у масив і змінити форму

`yArray` ← перетворити `yData` у масив і змінити форму

ДЛЯ `epoch` ВІД 0 ДО `epochs - 1`

Виконати крок градієнтного спуску та отримати значення втрат

ЯКЩО `epoch + 1` кратне 100

Вивести номер епохи і поточні втрати

Вивести "Training completed."

МЕТОД `TensorFlowManager(modelPath)`

Створити новий граф

Ініціалізувати сесію

Завантажити модель з `modelPath`

Вивести "Model loaded successfully."

МЕТОД `Predict(imageData, height, width, inputNodeName, outputNodeName) ->`

`NDArray`

Створити форму (1, height, width, 1) для зображення

`npImageData` ← перетворити `imageData` у масив з цією формою

Отримати вхідний тензор з графа

Отримати вихідний тензор з графа

Виконати інференцію у сесії та отримати результат

Повернути result

МЕТОД Dispose()

Якщо session існує, викликати session.Dispose()

```
public class TensorFlowManager : IDisposable
{
    private readonly Session _session;
    private readonly Graph _graph;
    private Tensor x, y, W, b, pred, loss, optimizer;

    public TensorFlowManager()
    {
        // Створюємо новий граф і сесію
        _graph = new Graph().as_default();
        _session = tf.Session(_graph);

        // Створення моделі (побудова графу)
        x = tf.placeholder(tf.float32, shape: (-1, 1), name: "input");
        y = tf.placeholder(tf.float32, shape: (-1, 1), name: "output");

        W = tf.Variable(tf.random.normal(new Shape(1, 1)), name: "weight");
        b = tf.Variable(tf.zeros(new Shape(1)), name: "bias");
        pred = tf.add(tf.matmul(x, W), b);

        // Визначаємо функцію втрат (середньоквадратична помилка)
        loss = tf.reduce_mean(tf.square(pred - y));

        // Визначаємо оптимізатор (градієнтний спуск)
        optimizer = tf.train.GradientDescentOptimizer(0.01f).minimize(loss);

        // Ініціалізуємо всі змінні
        _session.run(tf.global_variables_initializer());
    }
}
```

```

public void Train(float[] xData, float[] yData, int epochs = 1000)
{
    // Перетворення даних на NumPy масиви
    var xArray = np.array(xData).reshape(new Shape(-1, 1));
    var yArray = np.array(yData).reshape(new Shape(-1, 1));

    // Тренуємо модель
    for (int epoch = 0; epoch < epochs; epoch++)
    {
        // Виконуємо один крок градієнтного спуску і обчислюємо поточне значення
втрат
        var (_, lossValue) = _session.run((optimizer, loss),
                                           (x, xArray),
                                           (y, yArray));

        // Виводимо значення втрат кожні 100 епох для відстеження прогресу
        if ((epoch + 1) % 100 == 0)
        {
            Console.WriteLine($"Epoch: {epoch + 1}, Loss: {lossValue}");
        }
    }
    Console.WriteLine("Training completed.");
}

public TensorFlowManager(string modelPath)
{
    _graph = new Graph().as_default();
    _session = Tensorflow.Binding.tf.Session(_graph);
    var metaFilePath = Path.Combine(modelPath, "saved_model.meta");
    Tensorflow.Binding.tf.train.import_meta_graph(metaFilePath);

    Console.WriteLine("Model loaded successfully.");
}

public NArray Predict(float[] imageData, int height, int width, string inputNodeName,
string outputNodeName)
{
    var shape = new Shape(1, height, width, 1); // 1 - кількість зображень, 1 - кількість
каналів
    var npImageData = np.array(imageData).reshape(shape);

    // Отримуємо вхідний і вихідний тензори з графу

```

```
var inputTensor = _graph.OperationByName(inputNodeName).output[0];
var outputTensor = _graph.OperationByName(outputNodeName).output[0];

// Виконуємо інференцію і отримуємо результат
var result = _session.run(outputTensor, new FeedItem(inputTensor, npImageData));

return result;
}

public void Dispose()
{
    _session?.Dispose();
}
}
```

ДОДАТОК Г

Результати збору даних з модифікованої FreeRTOS на основі тензорної декомпозиції

Таблиця Г.1 – Таблиця збору телеметричних даних з планувальника задач в першій серії експериментів до оновлення політик планувальника.

Task Name	CPU Usage (%)	Memory Usage (KB)	Execution Time (ms)	Time (s)
Task 1	18.7275	8.480	2508.777	12
Task 1	16.6106	8.499	1900.619	24
Task 1	13.6275	5.565	1388.510	36
Task 1	19.5728	6.949	1919.808	48
Task 1	12.1325	6.700	2480.797	60
Task 1	14.9058	5.639	2360.202	72
Task 1	18.7349	9.311	1781.745	84
Task 1	14.4725	8.970	1949.500	96
Task 1	15.3317	5.519	2069.230	108
Task 1	12.7753	6.017	1345.737	120
Task 1	12.7675	7.724	1956.517	132
Task 1	15.5479	6.109	1748.514	144
Task 1	18.0925	6.091	2147.873	156
Task 1	18.9375	8.278	2331.349	168
Task 1	15.535	7.161	2380.352	180
Task 1	11.0675	9.256	2121.642	192
Task 1	15.1125	8.222	1712.740	204
Task 1	10.8575	5.851	1949.561	216
Task 1	15.5412	6.405	1967.029	228
Task 1	19.7854	8.306	2118.190	240
Task 1	16.8153	4.910	2104.218	252
Task 1	18.5123	8.850	2425.791	264
Task 1	10.8654	8.942	1650.690	276
Task 1	14.9606	8.065	2081.472	288
Task 1	17.4475	4.808	1533.599	300
Task 1	12.9859	4.956	1929.428	312
Task 1	15.1075	6.727	1703.119	324
Task 1	19.7825	7.973	1318.631	336
Task 1	15.7475	8.647	2029.900	348
Task 1	20.8525	7.216	2007.172	360
Task 1	16.1825	4.854	2416.117	372
Task 1	19.3659	6.690	2423.990	384
Task 1	16.3857	7.161	2076.194	396
Task 1	13.8375	7.225	2209.058	408
Task 1	13.6393	9.043	2308.647	420
Task 1	20.4316	8.785	2197.716	432

Продовження таблиці Г.1

Task 1	19.7955	9.367	1636.674	444
Task 1	11.0659	6.377	2261.461	456
Task 1	18.3155	5.057	1437.488	468
Task 1	11.9256	9.329	1505.637	480
Task 1	14.9053	4.762	2413.496	492
Task 1	12.3375	6.192	1464.585	504
Task 1	18.7225	6.635	2055.284	516
Task 1	14.2736	6.063	2451.087	528
Task 1	11.2825	9.007	1364.935	540
Task 1	15.7475	5.030	2394.333	552
Task 1	13.4025	8.047	2032.547	564
Task 1	21.0755	9.385	2004.576	576
Task 1	12.3375	5.519	1911.978	588
Task 1	15.9675	4.946	2607.492	600
Task 1	10.8625	7.189	1401.635	612
Task 1	14.6825	7.826	2291.170	624
Task 1	16.3955	9.192	1756.430	636
Task 1	20.4225	7.392	1646.321	648
Task 1	11.9175	6.276	1663.797	660
Task 1	19.7951	5.519	2029.026	672
Task 1	12.1465	5.675	1762.512	684
Task 1	20.4353	6.367	1725.829	696
Task 1	17.8775	6.940	1929.446	708
Task 1	13.4025	6.091	1983.605	720
Task 1	14.0425	6.967	2586.503	732
Task 1	18.0825	7.585	2224.804	744
Task 1	21.0625	5.786	2181.061	756
Task 1	18.9375	4.771	1901.501	768
Task 1	11.2821	7.586	1933.762	780
Task 1	10.8551	7.262	2604.905	792
Task 1	11.7053	6.035	1382.394	804
Task 1	19.3651	9.127	2331.410	816
Task 1	11.2959	8.167	2278.928	828
Task 1	16.5952	5.564	2272.829	840
Task 1	14.2748	8.232	1709.201	852
Task 1	15.9675	6.875	1310.741	864
Task 1	19.7854	6.229	1354.493	876
Task 1	17.6675	4.965	1908.474	888
Task 1	12.5625	9.007	1936.428	900
Task 1	18.0925	8.370	2594.411	912
Task 1	13.4075	9.274	1702.210	924

Продовження таблиці Г.1

Task 1	17.0257	4.938	2079.707	936
Task 1	20.6455	4.789	1736.333	948
Task 1	13.6225	7.567	1339.638	960
Task 1	12.1353	9.330	2508.751	972
Task 1	13.4125	8.296	1392.888	984
Task 1	15.3258	8.951	2267.595	996
Task 1	18.9457	5.925	1444.496	1008
Task 1	14.8925	5.454	1528.330	1020
Task 1	13.8377	7.871	1898.836	1032
Task 1	20.0109	5.076	1692.651	1044
Task 1	14.2575	6.358	1412.968	1056
Task 1	20.4244	5.758	1983.570	1068
Task 1	15.5357	5.989	2472.077	1080
Task 1	15.9575	7.503	1544.102	1092
Task 1	15.7575	7.761	2251.849	1104
Task 1	20.0051	7.420	1516.114	1116
Task 1	19.1612	5.104	1649.807	1128
Task 1	16.6025	8.702	2174.970	1140
Task 1	13.6225	7.041	1976.623	1152
Task 1	20.9534	5.094	1660.293	1164
Task 1	10.6475	4.651	1855.154	1176
Task 1	15.7425	5.556	1980.118	1188
Task 1	11.9275	6.413	2469.446	1200
Task 2	12.3452	7.834	2112.956	12
Task 2	15.5456	5.039	2385.525	24
Task 2	10.6525	7.087	2404.810	36
Task 2	19.7925	8.019	2055.284	48
Task 2	12.1275	5.583	1814.941	60
Task 2	15.5375	5.712	1375.429	72
Task 2	12.7725	5.140	1690.003	84
Task 2	14.0456	7.798	1399.879	96
Task 2	12.1275	8.979	2573.422	108
Task 2	17.8812	5.684	1977.453	120
Task 2	17.8711	5.777	1954.752	132
Task 2	17.4525	8.231	2014.215	144
Task 2	11.5025	5.140	2472.050	156
Task 2	19.3725	9.339	1395.553	168
Task 2	18.7225	5.491	2007.198	180
Task 2	19.1667	5.694	2497.435	192
Task 2	16.3858	9.108	2060.509	204
Task 2	11.9125	6.792	1689.968	216

Продовження таблиці Г.1

Task 2	12.3475	7.115	1700.497	228
Task 2	12.1325	6.349	2473.815	240
Task 2	12.9825	8.729	1527.499	252
Task 2	17.6651	6.884	1454.920	264
Task 2	17.4475	5.306	2169.719	276
Task 2	20.6375	6.026	2497.426	288
Task 2	15.3257	8.619	2201.211	300
Task 2	15.5453	5.399	1783.510	312
Task 2	18.3051	7.235	1973.102	324
Task 2	17.8725	6.635	2297.313	336
Task 2	19.7875	5.242	1380.655	348
Task 2	18.9325	5.039	1426.128	360
Task 2	15.1059	5.472	2006.359	372
Task 2	16.1885	8.785	2096.345	384
Task 2	17.8725	6.487	1430.506	396
Task 2	10.6375	6.617	1470.640	408
Task 2	17.8859	8.979	2555.098	420
Task 2	18.3058	8.592	2493.066	432
Task 2	20.4321	7.752	1995.803	444
Task 2	13.4058	7.272	2169.754	456
Task 2	11.0652	6.949	1370.169	468
Task 2	10.6448	7.900	1787.005	480
Task 2	11.4944	4.817	1356.223	492
Task 2	15.9657	9.422	2597.906	504
Task 2	13.4175	5.528	1488.999	516
Task 2	18.7325	6.266	1420.003	528
Task 2	16.1871	6.663	1347.467	540
Task 2	12.3456	8.683	1406.913	552
Task 2	13.6256	7.604	1974.858	564
Task 2	13.6269	9.367	2364.580	576
Task 2	15.9625	4.753	2050.906	588
Task 2	19.3659	7.705	1392.897	600
Task 2	13.1977	6.986	2531.496	612
Task 2	17.2375	9.191	2096.353	624
Task 2	18.7359	6.082	1624.475	636
Task 2	20.8547	8.988	2096.327	648
Task 2	17.2325	8.111	1555.409	660
Task 2	11.2809	8.545	2375.048	672
Task 2	16.8175	7.078	2375.048	684
Task 2	18.3055	8.453	1892.702	696
Task 2	15.3275	6.155	1391.167	708

Продовження таблиці Г.1

Task 2	18.7225	5.002	1897.115	720
Task 2	20.0055	7.235	1363.196	732
Task 2	13.6253	8.268	1863.927	744
Task 2	19.5856	4.716	2249.271	756
Task 2	16.6025	7.392	2556.845	768
Task 2	18.0855	6.939	2209.049	780
Task 2	13.1981	9.209	2611.878	792
Task 2	17.2327	4.928	1837.651	804
Task 2	11.9293	5.436	1905.827	816
Task 2	15.1075	8.462	1776.537	828
Task 2	16.3907	5.666	2558.593	840
Task 2	10.8575	6.414	2042.133	852
Task 2	19.3677	9.228	1873.504	864
Task 2	19.9975	9.154	1513.518	876
Task 2	11.7025	5.298	1767.746	888
Task 2	16.3956	8.416	1612.225	900
Task 2	16.3825	8.075	2244.902	912
Task 2	19.7925	8.481	1786.988	924
Task 2	19.7953	7.161	2272.812	936
Task 2	14.6875	8.241	2268.469	948
Task 2	16.8233	8.822	2003.729	960
Task 2	12.7756	6.377	1527.482	972
Task 2	16.3959	5.518	1350.945	984
Task 2	15.7425	5.491	2310.403	996
Task 2	18.7291	6.358	1496.890	1008
Task 2	17.4525	4.946	1518.744	1020
Task 2	10.6525	8.010	2253.631	1032
Task 2	16.1815	7.180	1615.702	1044
Task 2	13.1968	6.607	1403.418	1056
Task 2	19.1475	7.124	2035.999	1068
Task 2	11.0775	5.177	1351.828	1080
Task 2	18.7225	7.761	1604.334	1092
Task 2	11.7125	8.268	2227.391	1104
Task 2	12.1415	4.780	2012.441	1116
Task 2	16.1751	6.238	2429.251	1128
Task 2	12.3456	8.259	1316.857	1140
Task 2	18.3025	8.712	1856.037	1152
Task 2	17.4475	6.755	2425.773	1164
Task 2	19.3725	5.306	1938.201	1176
Task 2	19.7928	7.050	1574.651	1188
Task 2	15.5425	8.803	1868.261	1200

Продовження таблиці Г.1

Task 3	13.6175	8.148	1358.818	12
Task 3	17.2475	9.274	1944.248	24
Task 3	17.2375	7.991	1769.511	36
Task 3	15.5275	5.934	1724.099	48
Task 3	19.1453	5.242	1334.369	60
Task 3	20.8575	5.519	1378.025	72
Task 3	20.8555	8.259	2393.451	84
Task 3	12.9971	4.836	2541.973	96
Task 3	18.7217	8.933	1628.862	108
Task 3	10.8525	9.071	1891.863	120
Task 3	11.0746	5.999	1769.520	132
Task 3	20.8525	5.934	1920.664	144
Task 3	18.3025	4.965	2477.284	156
Task 3	20.8625	5.611	2081.507	168
Task 3	14.0575	4.992	2210.779	180
Task 3	20.8625	6.321	2207.337	192
Task 3	10.8625	8.278	2112.930	204
Task 3	10.8558	6.663	1965.229	216
Task 3	16.3857	5.832	1627.062	228
Task 3	15.3252	6.109	2444.088	240
Task 3	18.3075	6.589	2014.215	252
Task 3	17.2325	6.773	1955.634	264
Task 3	17.4512	4.688	2217.779	276
Task 3	10.6375	5.288	2372.444	288
Task 3	14.4725	9.145	1733.667	300
Task 3	10.8651	6.940	1334.360	312
Task 3	19.7958	6.728	1933.771	324
Task 3	15.9575	9.395	2238.785	336
Task 3	17.2475	6.211	2443.223	348
Task 3	11.7187	6.155	2406.523	360
Task 3	17.2319	6.174	1385.041	372
Task 3	11.2819	8.231	2618.869	384
Task 3	12.7675	6.396	1420.029	396
Task 3	14.0425	9.376	2019.431	408
Task 3	18.5225	4.872	1779.106	420
Task 3	15.5375	6.128	1436.605	432
Task 3	11.5025	8.499	2389.903	444
Task 3	17.6625	7.734	1331.704	456
Task 3	11.7025	9.182	2548.998	468
Task 3	15.1125	7.973	1368.456	480
Task 3	12.7652	7.761	1513.527	492

Продовження таблиці Г.1

Task 3	20.6425	9.219	1405.139	504
Task 3	13.8425	9.320	2332.257	516
Task 3	17.4537	7.622	1406.860	528
Task 3	12.3475	6.608	1509.106	540
Task 3	11.7025	7.382	1406.913	552
Task 3	13.8275	5.823	2615.365	564
Task 3	16.5956	5.251	1339.629	576
Task 3	11.0725	6.930	1854.298	588
Task 3	14.6814	8.434	2527.092	600
Task 3	18.0875	8.712	1963.507	612
Task 3	14.4725	6.284	1829.822	624
Task 3	15.9667	6.146	1681.247	636
Task 3	11.0659	5.445	1845.568	648
Task 3	14.4759	4.965	1914.573	660
Task 3	14.6925	5.768	2451.070	672
Task 3	17.2495	8.767	2484.310	684
Task 3	11.9281	6.201	2591.815	696
Task 3	19.1458	6.635	1585.119	708
Task 3	17.4648	5.140	2076.256	720
Task 3	18.9425	8.997	2534.091	732
Task 3	18.7256	8.813	2457.230	744
Task 3	14.2675	7.936	2586.555	756
Task 3	18.9351	4.614	1888.342	768
Task 3	10.6457	8.176	1551.931	780
Task 3	12.7759	7.816	1855.171	792
Task 3	16.3825	6.359	1760.782	804
Task 3	15.7575	9.265	2510.533	816
Task 3	21.0746	6.367	2046.493	828
Task 3	15.3253	9.293	2594.428	840
Task 3	17.0207	8.010	1615.746	852
Task 3	16.8051	8.379	2029.070	864
Task 3	19.3625	5.685	2324.367	876
Task 3	17.8866	5.842	1973.128	888
Task 3	11.9175	5.140	2095.462	900
Task 3	13.8358	6.201	1522.222	912
Task 3	15.9675	5.980	1784.375	924
Task 3	20.6359	5.343	2239.598	936
Task 3	20.8557	6.570	1510.014	948
Task 3	10.8525	6.478	1788.744	960
Task 3	10.6525	6.718	2113.768	972
Task 3	20.6425	6.700	2399.550	984

Кінець таблиці Г.1

Task 3	12.9775	8.674	1315.136	996
Task 3	11.4925	8.315	2486.014	1008
Task 3	18.3552	5.039	1718.874	1020
Task 3	17.2425	6.755	1375.464	1032
Task 3	12.3535	7.419	1313.388	1044
Task 3	19.1551	5.925	1874.352	1056
Task 3	17.8776	8.435	1413.021	1068
Task 3	14.4675	9.339	2079.725	1080
Task 3	20.6351	8.047	1750.279	1092
Task 3	13.8375	6.995	2137.396	1104
Task 3	11.0775	8.951	1811.463	1116
Task 3	14.6852	4.716	2590.042	1128
Task 3	15.5309	7.650	1479.413	1140
Task 3	17.4593	7.909	2050.024	1152
Task 3	15.5468	8.370	1934.697	1164
Task 3	12.7754	8.047	2547.207	1176
Task 3	20.8606	4.679	2227.417	1188
Task 3	12.1379	5.731	2584.816	1200

Таблиця Г.2 – Таблиця збору телеметричних даних з планувальника задач в першій серії експериментів після оновлення політик планувальника.

Task Name	CPU Usage (%)	Memory Usage (KB)	Execution Time (us)	Time (s)
Task 1	19.7727	8.228	1940.8174	12
Task 1	12.194	6.876	2278.0923	24
Task 1	11.9146	7.965	2018.5328	36
Task 1	15.431	5.512	2335.5049	48
Task 1	14.0737	6.116	1465.5006	60
Task 1	14.5152	8.049	1662.3083	72
Task 1	15.3711	6.61	2271.7192	84
Task 1	13.0207	6.183	2104.1252	96
Task 1	17.2714	7.699	1941.8702	108
Task 1	14.9907	7.755	1406.2644	120
Task 1	17.3334	8.698	1349.4348	132
Task 1	13.6866	5.698	1683.8522	144
Task 1	12.9592	5.945	1502.6476	156
Task 1	10.3391	8.189	1875.7159	168
Task 1	15.1452	6.746	1542.4642	180
Task 1	12.8974	7.599	1286.439	192
Task 1	12.2311	7.324	1668.6062	204
Task 1	14.7795	7.044	1896.395	216

Продовження таблиці Г.2

Task 1	13.6505	6.773	1137.5313	228
Task 1	13.6219	7.332	2047.6526	240
Task 1	13.6807	6.604	1188.4768	252
Task 1	20.0304	8.315	1218.0479	264
Task 1	15.1928	6.698	1809.4114	276
Task 1	16.3289	6.639	2331.2185	288
Task 1	13.4017	8.453	2199.042	300
Task 1	15.3105	7.895	1746.4532	312
Task 1	19.6161	7.139	1228.8196	324
Task 1	18.987	6.365	1424.8756	336
Task 1	14.9881	5.842	1601.0046	348
Task 1	17.3387	7.977	1383.3296	360
Task 1	14.2639	7.817	1686.5406	372
Task 1	17.065	6.452	2004.4148	384
Task 1	19.5334	8.838	2147.1563	396
Task 1	10.3336	6.256	1418.4464	408
Task 1	13.9801	7.549	1469.298	420
Task 1	13.5316	5.946	1509.5468	432
Task 1	15.6314	8.61	1354.2098	444
Task 1	16.0142	6.725	1601.5498	456
Task 1	11.1083	7.853	2251.8674	468
Task 1	18.8558	6.589	2000.0532	480
Task 1	19.376	6.227	1358.947	492
Task 1	13.599	5.278	1374.3812	504
Task 1	15.4925	5.75	1704.9636	516
Task 1	11.0059	5.626	1531.6545	528
Task 1	14.3606	5.458	1526.109	540
Task 1	19.807	7.709	2334.7151	552
Task 1	14.4763	8.157	2119.3713	564
Task 1	15.4512	6.599	1548.1226	576
Task 1	11.6793	5.571	2144.4492	588
Task 1	18.8567	8.111	2076.2649	600
Task 1	12.9034	6.693	1781.8145	612
Task 1	15.16	8.554	1953.2058	624
Task 1	19.9259	8.535	1962.6993	636
Task 1	12.4668	6.879	1580.7954	648
Task 1	13.1176	6.478	1135.0496	660
Task 1	10.6012	6.033	2342.6484	672
Task 1	15.1938	8.22	1885.623	684
Task 1	19.8565	7.105	1522.067	696
Task 1	14.9627	8.129	1545.848	708

Продовження таблиці Г.2

Task 1	19.7438	7.08	2004.1892	720
Task 1	11.0769	8.393	1277.9604	732
Task 1	17.744	7.385	2265.8352	744
Task 1	15.7824	7.766	1608.9377	756
Task 1	18.0562	8.04	1149.5814	768
Task 1	18.6692	8.323	1660.485	780
Task 1	10.3483	5.41	1405.8132	792
Task 1	10.4204	7.807	2075.0806	804
Task 1	18.5073	7.955	2146.6113	816
Task 1	16.564	6.379	1781.9272	828
Task 1	19.3883	8.776	1155.785	840
Task 1	11.2206	6.171	2155.0144	852
Task 1	11.5656	8.57	2126.9849	864
Task 1	10.8553	7.231	1315.0323	876
Task 1	14.0824	6.176	2189.5107	888
Task 1	18.0267	7.077	1983.4915	900
Task 1	15.2003	7.128	1610.4604	912
Task 1	15.137	7.446	2088.5784	924
Task 1	11.1497	7.004	1176.6898	936
Task 1	18.1824	6.643	1387.2585	948
Task 1	14.8799	7.174	2040.8098	960
Task 1	10.5025	6.294	1927.282	972
Task 1	14.8088	5.353	1871.129	984
Task 1	16.5314	7.289	1117.0776	996
Task 1	13.5731	7.75	2089.7627	1008
Task 1	15.5459	5.742	2043.9116	1020
Task 1	16.7784	8.818	1558.5372	1032
Task 1	15.4175	6.933	1615.7054	1044
Task 1	11.7487	7.001	1353.063	1056
Task 1	16.3608	6.861	2111.2124	1068
Task 1	16.2773	6.091	2144.6938	1080
Task 1	14.6844	6.194	1785.875	1092
Task 1	17.3756	5.771	1582.4874	1104
Task 1	16.5408	6.218	1493.9248	1116
Task 1	20.0243	6.567	1531.3162	1128
Task 1	11.6759	8.39	1454.3337	1140
Task 1	15.9224	7.602	1757.7701	1152
Task 1	17.1799	6.683	1712.8217	1164
Task 1	12.2379	7.94	2240.1934	1176
Task 1	11.2359	8.74	1371.8434	1188
Task 1	10.3577	7.939	1782.6604	1200

Продовження таблиці Г.2

Task 2	18.9971	8.708	2263.2034	12
Task 2	13.7908	7.37	1324.4696	24
Task 2	14.5094	8.008	1590.308	36
Task 2	13.03	6.058	2057.5974	48
Task 2	13.1346	5.838	1312.9458	60
Task 2	15.8606	5.656	1541.5616	72
Task 2	19.0718	5.818	1505.7118	84
Task 2	11.8722	8.239	1362.1053	96
Task 2	11.9889	7.913	1222.9355	108
Task 2	11.7486	7.948	2161.707	120
Task 2	15.7728	6.183	2040.7346	132
Task 2	11.0171	8.545	1637.55	144
Task 2	17.2789	8.13	1742.9189	156
Task 2	13.7988	8.562	1163.6244	168
Task 2	15.0813	7.248	1559.1014	180
Task 2	16.5585	8.047	1324.6576	192
Task 2	18.2231	5.979	2314.6377	204
Task 2	11.8548	8.118	1146.1036	216
Task 2	14.9659	6.257	1812.5886	228
Task 2	12.0901	6.912	2252.2058	240
Task 2	11.5403	7.257	2071.8284	252
Task 2	13.2604	6.334	1935.892	264
Task 2	10.8954	6.752	1780.9308	276
Task 2	18.7058	8.009	2244.9116	288
Task 2	11.4493	6.056	2304.2607	300
Task 2	12.8594	5.63	1819.7134	312
Task 2	15.8388	7.779	1367.8768	324
Task 2	14.9991	6.108	1283.525	336
Task 2	19.6691	6.371	1529.662	348
Task 2	12.9955	7.524	1240.3248	360
Task 2	10.2643	7.479	1401.1888	372
Task 2	17.0639	5.877	1504.8282	384
Task 2	15.4179	7.694	1653.8114	396
Task 2	13.5125	5.434	1564.4966	408
Task 2	12.1412	6.699	1168.606	420
Task 2	18.494	7.484	2273.5615	432
Task 2	15.4149	8.62	1810.4454	444
Task 2	11.7541	6.28	1386.9578	456
Task 2	16.938	5.38	2113.9009	468
Task 2	10.3829	6.599	1414.4423	480
Task 2	13.8786	6.843	2057.3154	492

Продовження таблиці Г.2

Task 2	12.564	5.281	1458.4696	504
Task 2	17.6235	6.167	1676.7838	516
Task 2	11.3247	8.79	1372.2568	528
Task 2	12.5279	6.356	2144.1675	540
Task 2	18.2861	7.555	2221.7888	552
Task 2	14.6663	7.204	1917.0364	564
Task 2	17.413	7.925	1819.751	576
Task 2	10.5814	6.653	1512.7428	588
Task 2	18.7641	6.52	1347.63	600
Task 2	16.0353	5.734	2122.3979	612
Task 2	12.2414	7.775	2121.6648	624
Task 2	14.7581	6.327	1760.8156	636
Task 2	11.3415	5.496	1126.7782	648
Task 2	17.1993	5.302	1272.6028	660
Task 2	16.904	7.607	1334.734	672
Task 2	19.2766	6.673	1714.1188	684
Task 2	10.8623	6.901	2309.9941	696
Task 2	11.1268	8.55	1735.437	708
Task 2	19.9942	5.79	1903.7454	720
Task 2	15.8555	8.085	1122.3978	732
Task 2	19.6704	8.654	1643.9792	744
Task 2	18.2869	6.423	1874.6256	756
Task 2	17.6059	6.993	2269.4258	768
Task 2	13.7138	6.455	1891.432	780
Task 2	17.6093	7.774	1727.5225	792
Task 2	16.6481	6.964	2209.9268	804
Task 2	13.3328	7.785	1796.5154	816
Task 2	19.5702	7.578	1922.9395	828
Task 2	12.0226	7.58	1155.3528	840
Task 2	11.6791	6.517	1721.0743	852
Task 2	18.2688	7.433	2005.7872	864
Task 2	17.6433	5.547	1274.6708	876
Task 2	17.7545	6.889	1910.25	888
Task 2	16.349	7.727	1543.8364	900
Task 2	11.0073	7.909	1645.972	912
Task 2	19.652	7.957	2160.4097	924
Task 2	17.1484	7.222	1970.9333	936
Task 2	11.1262	8.735	1273.9564	948
Task 2	17.6425	6.679	1621.834	960
Task 2	14.8149	7.101	1557.8606	972
Task 2	20.0483	5.597	1472.1554	984

Продовження таблиці Г.2

Task 2	13.9796	6.787	1431.1923	996
Task 2	11.7678	7.513	1298.1132	1008
Task 2	19.851	8.772	2296.8162	1020
Task 2	13.7976	5.332	1472.6818	1032
Task 2	10.3779	5.427	2227.2969	1044
Task 2	14.4772	6.079	2195.583	1056
Task 2	18.9788	7.065	1970.9897	1068
Task 2	13.2403	8.847	1491.2552	1080
Task 2	13.2899	7.297	2334.0007	1092
Task 2	14.3643	7.947	1830.9554	1104
Task 2	13.917	5.973	1562.1469	1116
Task 2	10.7201	6.859	1970.6326	1128
Task 2	13.1843	6.942	1347.0472	1140
Task 2	17.5229	6.984	2341.0127	1152
Task 2	13.8847	8.591	1744.047	1164
Task 2	16.0809	7.189	1813.5472	1176
Task 2	18.03	8.805	2250.2319	1188
Task 2	17.5467	6.96	1858.4772	1200
Task 3	17.1906	8.842	1780.6112	12
Task 3	13.3924	8.578	2245.5698	24
Task 3	19.4474	6.155	1143.359	36
Task 3	18.7462	5.638	2325.5413	48
Task 3	17.2485	5.645	1268.5986	60
Task 3	16.8025	6.901	1922.7513	72
Task 3	18.309	6.008	1663.0416	84
Task 3	12.8544	8.058	1400.5496	96
Task 3	16.4374	7.015	1506.1442	108
Task 3	11.4136	7.892	1796.0454	120
Task 3	15.2859	5.45	1344.1334	132
Task 3	14.9608	6.031	2202.0686	144
Task 3	12.1008	5.47	2319.9954	156
Task 3	12.6948	7.896	2077.2048	168
Task 3	11.4718	7.349	1707.4073	180
Task 3	19.2839	8.051	2323.8115	192
Task 3	19.0827	7.03	1996.8762	204
Task 3	13.6146	5.955	1437.9598	216
Task 3	15.4605	6.474	1501.5948	228
Task 3	17.5426	7.31	1515.525	240
Task 3	19.7266	6.013	2211.562	252
Task 3	14.6418	7.413	1938.0914	264
Task 3	13.3456	7.495	1631.459	276

Продовження таблиці Г.2

Task 3	18.9556	6.904	1682.9874	288
Task 3	13.8763	6.638	1866.7864	300
Task 3	13.7452	5.916	1633.3202	312
Task 3	18.721	6.978	1461.7406	324
Task 3	10.2414	5.425	1959.936	336
Task 3	18.8758	8.623	2021.3149	348
Task 3	16.3936	8.605	1967.418	360
Task 3	12.6803	7.795	1281.194	372
Task 3	15.6193	7.768	1172.1028	384
Task 3	19.5685	7.187	2193.8911	396
Task 3	15.5555	7.325	1612.1147	408
Task 3	13.8174	8.312	2302.5874	420
Task 3	16.8077	7.127	1533.2712	432
Task 3	18.2984	7.45	1595.252	444
Task 3	11.6999	7.42	1395.173	456
Task 3	18.226	6	1734.6285	468
Task 3	13.0253	6.594	1253.1644	480
Task 3	14.909	6.429	1592.1689	492
Task 3	12.7955	6.133	1836.3506	504
Task 3	19.4999	8.226	1406.0952	516
Task 3	10.991	8.364	1929.5566	528
Task 3	16.7444	7.444	2011.3892	540
Task 3	14.311	7.008	1915.382	552
Task 3	18.7326	7.059	1605.7418	564
Task 3	14.9235	6.53	1156.161	576
Task 3	18.6481	5.713	1139.6366	588
Task 3	19.2153	5.489	1536.016	600
Task 3	14.5181	8.069	2314.3181	612
Task 3	16.5416	8.687	1146.7616	624
Task 3	12.9638	6.356	1883.7056	636
Task 3	17.9768	6.919	1526.015	648
Task 3	12.9501	8.723	1972.5502	660
Task 3	15.0639	7.127	1343.0994	672
Task 3	19.5356	6.39	1914.3105	684
Task 3	18.5646	8.544	1173.118	696
Task 3	17.6224	5.592	1177.1974	708
Task 3	17.2093	7.646	2110.9304	720
Task 3	15.7068	5.534	2290.2178	732
Task 3	16.4473	6.705	1921.097	744
Task 3	19.6286	8.742	1449.0326	756
Task 3	17.1965	7.936	2275.9868	768

Кінець таблиці Г.2

Task 3	16.3214	6.208	1298.1508	780
Task 3	14.5936	7.647	1209.6635	792
Task 3	17.5626	5.744	1536.7867	804
Task 3	16.7442	8.391	1588.0144	816
Task 3	13.7231	6.33	1844.9042	828
Task 3	13.4724	5.913	2330.6169	840
Task 3	17.518	8.466	2345.2239	852
Task 3	15.0712	8.505	1952.7922	864
Task 3	17.7557	6.519	1601.1926	876
Task 3	11.0523	5.948	2107.6594	888
Task 3	13.0886	6.276	2007.0654	900
Task 3	18.8964	6.871	1272.0388	912
Task 3	15.7629	6.493	1625.7819	924
Task 3	12.8022	7.695	1984.5627	936
Task 3	15.3683	5.644	2043.0844	948
Task 3	12.4381	8.385	2067.542	960
Task 3	12.5884	8.672	1437.4146	972
Task 3	11.6907	6.598	1865.0756	984
Task 3	12.9507	8.538	1202.0122	996
Task 3	17.4156	6.238	1625.011	1008
Task 3	10.5761	7.373	1478.848	1020
Task 3	18.1868	8.001	1344.9418	1032
Task 3	11.9295	8.827	1620.6309	1044
Task 3	13.0473	7.126	1850.2244	1056
Task 3	19.5697	6.817	1884.8334	1068
Task 3	10.2588	5.547	2175.9377	1080
Task 3	15.8987	6.68	1431.6622	1092
Task 3	16.0521	6.041	1876.9189	1104
Task 3	16.9128	6.721	1250.1002	1116
Task 3	13.3759	6.379	2105.7795	1128
Task 3	15.9518	8.565	2155.8792	1140
Task 3	10.6486	7.878	1762.8459	1152
Task 3	17.4493	5.904	1557.4282	1164
Task 3	10.5066	5.943	2270.2341	1176
Task 3	10.6826	5.651	1309.9941	1188
Task 3	18.5558	5.829	1257.7516	1200

Таблиця Г.3 – Таблиця збору телеметричних даних з планувальника задач в другій серії експериментів до оновлення політик планувальника.

Task Name	CPU Usage (%)	Memory Usage (KB)	Execution Time (us)	Time (s)
Task 1	25.705	7.149	2089.8889	12
Task 1	26.7803	5.837	2161.2891	24
Task 1	23.8605	9.714	2061.3486	36
Task 1	24.2246	8.086	2596.5251	48
Task 1	23.1221	6.73	2200.584	60
Task 1	22.9215	6.463	2262.8042	72
Task 1	24.0403	8.66	1993.5251	84
Task 1	27.031	6.96	2077.3276	96
Task 1	23.6974	7.402	2563.6267	108
Task 1	25.6194	6.257	1844.7802	120
Task 1	26.9565	9.791	2571.8057	132
Task 1	24.9095	7.221	2157.3584	144
Task 1	27.7122	7.101	2116.5007	156
Task 1	26.5302	6.406	2294.7383	168
Task 1	26.1402	8.543	1833.6594	180
Task 1	23.7665	9.555	2124.1057	192
Task 1	21.9476	6.765	1996.7236	204
Task 1	21.81	7.098	2400.1599	216
Task 1	25.6228	7.353	2416.6274	228
Task 1	22.0083	9.652	1865.0686	240
Task 1	27.6851	6.331	2341.7971	252
Task 1	24.1153	7.521	2348.5479	264
Task 1	21.9353	7.66	1958.173	276
Task 1	22.5411	6.197	1867.7542	288
Task 1	22.7057	8.737	2289.2085	300
Task 1	26.6023	5.861	2127.4019	312
Task 1	22.1093	5.899	1895.8186	324
Task 1	26.7713	8.88	2419.667	336
Task 1	25.2908	7.92	2376.4656	348
Task 1	26.9271	7.307	2324.231	360
Task 1	25.6759	9.511	1867.0583	372
Task 1	22.6049	6.386	2008.2836	384
Task 1	22.2215	8.819	2141.0615	396
Task 1	27.2461	8.046	2526.5288	408
Task 1	24.6662	8.93	2109.396	420
Task 1	26.4483	6.405	2121.1272	432
Task 1	22.7793	7.369	2520.5107	444
Task 1	22.2154	7.266	2121.7864	456
Task 1	27.6117	9.596	1860.3199	468

Продовження таблиці Г.3

Task 1	23.2077	7.622	2445.6929	480
Task 1	22.7452	6.51	1852.2878	492
Task 1	22.9914	7.153	1897.5154	504
Task 1	23.5434	7.945	2557.3521	516
Task 1	26.2894	7.727	2469.4724	528
Task 1	22.4006	8.177	2273.925	540
Task 1	27.2304	8.896	2441.2249	552
Task 1	23.9927	7.416	2237.9624	564
Task 1	23.5995	7.405	2279.9675	576
Task 1	26.5268	9.31	2522.9033	588
Task 1	25.0884	8.682	2540.3965	600
Task 1	26.9909	6.445	1939.667	612
Task 1	25.5445	6.345	2564.6887	624
Task 1	26.4048	5.845	1936.7128	636
Task 1	23.2218	9.699	2382.5327	648
Task 1	23.0691	6.47	2499.9783	660
Task 1	23.0304	7.234	2211.1189	672
Task 1	23.816	8.719	2052.8767	684
Task 1	26.7869	6.133	1955.1335	696
Task 1	25.2587	7.929	2156.3696	708
Task 1	23.4636	7.761	2306.7747	720
Task 1	25.0418	7.872	2608.8911	732
Task 1	27.4398	7.173	1845.708	744
Task 1	27.0909	6.259	2146.3596	756
Task 1	26.7504	8.611	2139.145	768
Task 1	22.8631	8.238	2342.2366	780
Task 1	27.3369	9.005	2441.762	792
Task 1	26.5969	7.897	1907.4399	804
Task 1	25.2969	9.473	2395.7407	816
Task 1	26.5616	9.758	2190.4277	828
Task 1	27.1344	6.819	2330.7739	840
Task 1	26.308	6.282	2277.1233	852
Task 1	22.0094	9.034	1964.0448	864
Task 1	25.9181	7.367	2090.9631	876
Task 1	25.9885	6.8	2300.2559	888
Task 1	27.2554	5.85	2292.895	900
Task 1	24.2942	8.982	1931.5736	912
Task 1	27.2142	7.003	2581.3516	924
Task 1	27.5556	6.136	2137.7046	936
Task 1	22.9703	6.039	2392.2617	948
Task 1	26.0576	6.004	2385.8286	960

Продовження таблиці Г.3

Task 1	23.2308	9.605	2399.0735	972
Task 1	21.9916	9.016	2230.626	984
Task 1	23.4841	9.287	2012.715	996
Task 1	25.7056	7.788	2414.2959	1008
Task 1	23.2495	7.108	2481.6431	1020
Task 1	21.9532	6.626	1966.5106	1032
Task 1	25.328	9.031	1991.7675	1044
Task 1	26.1447	9.022	2504.4827	1056
Task 1	23.0514	6.452	1966.5472	1068
Task 1	26.5964	9.154	2132.8706	1080
Task 1	23.5331	9.708	2166.9165	1092
Task 1	22.781	7.391	2143.8816	1104
Task 1	22.9177	9.573	2191.3066	1116
Task 1	27.5775	9.788	2517.2026	1128
Task 1	23.6309	6.757	2175.657	1140
Task 1	27.521	7.586	2220.0056	1152
Task 1	27.6298	9.408	1893.4016	1164
Task 1	27.0527	8.714	1906.9882	1176
Task 1	23.6525	6.614	2255.4922	1188
Task 1	26.7706	9.292	1820.3414	1200
Task 2	23.4587	8.539	1861.382	12
Task 2	23.9931	7.21	2537.6252	24
Task 2	25.0105	6.417	2463.0269	36
Task 2	27.1512	6.403	2240.1355	48
Task 2	24.8617	8.079	1851.9583	60
Task 2	23.5752	6.04	2227.6108	72
Task 2	23.0473	9.767	2395.3989	84
Task 2	22.7225	7.27	2473.4885	96
Task 2	25.5088	8.412	2548.0137	108
Task 2	26.8313	8.178	2488.6987	120
Task 2	22.3438	9.129	1951.9843	132
Task 2	24.1709	9.289	2021.675	144
Task 2	26.0563	8.725	2537.0271	156
Task 2	24.2923	7.063	2558.3777	168
Task 2	25.8063	7.191	2420.3018	180
Task 2	27.2775	7.398	2422.2183	192
Task 2	25.6604	7.206	2606.5107	204
Task 2	22.9897	7.131	2274.1448	216
Task 2	22.8411	9.639	2487.832	228
Task 2	26.3235	7.535	2612.6021	240
Task 2	21.9774	9.043	2543.9609	252

Продовження таблиці Г.3

Task 2	22.4546	7.82	2073.5068	264
Task 2	25.3544	9.161	2242.0764	276
Task 2	26.0902	6.636	2130.3193	288
Task 2	23.1635	6.422	1936.8716	300
Task 2	26.7185	7.568	2193.9678	312
Task 2	25.7525	9.445	2370.5452	324
Task 2	23.8589	8.64	1912.8842	336
Task 2	23.8751	6.532	2572.77	348
Task 2	26.3124	7.812	1873.0156	360
Task 2	26.9416	9.178	2560.7336	372
Task 2	27.4108	8.483	1897.7961	384
Task 2	23.2683	7.561	2039.1191	396
Task 2	22.2675	8.989	2548.1724	408
Task 2	24.3279	6.048	2300.1338	420
Task 2	25.0271	8.104	1822.551	432
Task 2	24.9994	6.694	2523.4526	444
Task 2	26.1153	6.538	2256.9082	456
Task 2	23.8178	8.742	2476.2598	468
Task 2	27.4892	8.44	2024.6536	480
Task 2	25.2246	8.121	2013.2397	492
Task 2	23.8869	6.895	2036.7388	504
Task 2	23.8174	8.948	2176.5969	516
Task 2	26.0782	9.428	1841.5942	528
Task 2	24.6002	7.028	2296.0078	540
Task 2	24.1136	7.498	1925.1648	552
Task 2	27.2331	9.353	1888.653	564
Task 2	22.5753	6.006	2010.884	576
Task 2	22.2824	9.603	2559.2444	588
Task 2	21.8266	8.785	2559.696	600
Task 2	23.471	7.644	1899.9324	612
Task 2	23.2619	8.111	2270.0186	624
Task 2	21.9275	9.033	2590.446	636
Task 2	22.5572	6.193	1977.8024	648
Task 2	24.4374	8.51	2297.9365	660
Task 2	26.9194	8.681	2156.4917	672
Task 2	25.6919	9.506	1977.1064	684
Task 2	24.3366	6.16	2017.012	696
Task 2	22.5386	7.638	2170.1514	708
Task 2	24.8286	7.653	2607.8169	720
Task 2	22.2315	6.21	2506.7532	732
Task 2	23.6961	6.122	1914.8132	744

Продовження таблиці Г.3

Task 2	26.6809	7.715	2004.0844	756
Task 2	24.7711	7.966	1961.469	768
Task 2	22.7028	9.332	2566.8616	780
Task 2	27.667	6.519	2308.7156	792
Task 2	26.2703	6.429	2087.24	804
Task 2	27.0281	7.555	2354.981	816
Task 2	24.7621	8.059	1944.9282	828
Task 2	23.7803	6.786	2036.2017	840
Task 2	27.2133	8.467	2507.1194	852
Task 2	24.3807	7.36	2525.833	864
Task 2	24.5654	6.58	1828.4714	876
Task 2	22.0675	9.362	2134.7871	888
Task 2	21.916	9.516	2351.209	900
Task 2	26.1104	7.316	2611.5278	912
Task 2	22.7691	8.08	2404.9939	924
Task 2	25.0599	7.684	1841.9604	936
Task 2	25.2848	8.265	2107.0156	948
Task 2	21.9981	6.363	2549.564	960
Task 2	24.5295	6.749	2061.9712	972
Task 2	21.7885	6.189	2595.2434	984
Task 2	26.7259	6.4	2062.0444	996
Task 2	24.3254	6.437	2077.4375	1008
Task 2	27.5028	7.773	2186.9241	1020
Task 2	23.7847	8.316	2432.1062	1032
Task 2	26.1454	8.61	2303.7961	1044
Task 2	25.8734	8.476	2332.666	1056
Task 2	21.8451	7.34	2367.3467	1068
Task 2	23.0798	8.502	2390.0645	1080
Task 2	21.9497	6.581	1919.757	1092
Task 2	23.9233	8.417	1852.7272	1104
Task 2	26.213	8.638	1990.7299	1116
Task 2	22.2049	8.183	2506.6189	1128
Task 2	23.0451	7.002	2197.4468	1140
Task 2	26.2565	9.198	2175.1443	1152
Task 2	21.7624	5.853	1844.585	1164
Task 2	22.1913	7.798	1819.255	1176
Task 2	24.8377	7.56	1824.3455	1188
Task 2	27.1541	8.756	2237.4009	1200
Task 3	27.571	8.441	2198.2646	12
Task 3	22.5855	9.294	2126.4009	24
Task 3	25.438	9.185	2039.0582	36

Продовження таблиці Г.3

Task 3	24.1534	6.116	2313.0002	48
Task 3	24.6805	8.904	2596.0735	60
Task 3	27.4777	7.871	2060.3352	72
Task 3	23.1305	5.997	1892.718	84
Task 3	25.3747	7.738	2473.1101	96
Task 3	26.4013	9.8	1889.9592	108
Task 3	21.8171	9.085	2243.4924	120
Task 3	23.1374	6.086	1986.2253	132
Task 3	22.1839	8.966	1951.1783	144
Task 3	26.2647	6.567	2117.4529	156
Task 3	23.5102	9.623	1963.3612	168
Task 3	24.2401	6.391	1857.073	180
Task 3	22.7373	5.986	1934.723	192
Task 3	22.3019	9.643	1916.0338	204
Task 3	24.9629	9.172	1907.4521	216
Task 3	23.7197	6.848	2442.7754	228
Task 3	27.6911	5.987	2611.2471	240
Task 3	23.0442	8.465	2123.2146	252
Task 3	23.0817	6.421	2563.2727	264
Task 3	23.3575	6.394	2080.8069	276
Task 3	24.2014	7.155	2368.2258	288
Task 3	23.5229	7.471	2576.4932	300
Task 3	25.2728	7.057	1818.2906	312
Task 3	23.4348	6.968	2108.688	324
Task 3	21.9246	6.68	2593.1804	336
Task 3	23.2691	6.097	2113.3513	348
Task 3	25.4423	7.766	2160.0439	360
Task 3	22.7328	9.507	2063.9121	372
Task 3	23.017	8.747	2073.5923	384
Task 3	26.3142	6.783	2571.3174	396
Task 3	21.9966	8.238	1876.0063	408
Task 3	24.5327	5.948	2083.9807	420
Task 3	22.1348	8.544	2596.989	432
Task 3	23.1893	6.964	2387.8672	444
Task 3	23.8419	7.158	2253.6978	456
Task 3	24.6159	7.229	2106.3931	468
Task 3	24.4343	8.26	2550.8457	480
Task 3	26.9258	7.079	2200.5107	492
Task 3	26.3846	6.216	1980.5978	504
Task 3	23.2634	7.288	1868.6454	516
Task 3	27.5717	9.081	2522.6714	528

Продовження таблиці Г.3

Task 3	25.0548	6.565	2446.7549	540
Task 3	23.5307	6.097	1944.2202	552
Task 3	25.2567	7.061	2508.2546	564
Task 3	27.7031	7.195	2099.96	576
Task 3	27.6077	7.86	2564.0906	588
Task 3	25.6866	6.491	2032.0634	600
Task 3	21.8769	6.486	2562.699	612
Task 3	25.485	9.789	2270.2261	624
Task 3	25.1156	8.401	2590.0188	636
Task 3	23.7584	6.083	1931.6224	648
Task 3	26.9054	8.502	1969.4771	660
Task 3	26.0735	7.052	2220.958	672
Task 3	24.6098	8.625	2362.0366	684
Task 3	25.1526	6.562	1930.4016	696
Task 3	22.6233	7.044	2066.1094	708
Task 3	27.1249	8.17	2539.6641	720
Task 3	25.6217	7.971	2317.6514	732
Task 3	23.7753	8.616	2115.9026	744
Task 3	25.8119	7.052	2390.0889	756
Task 3	24.7954	9.331	2013.8258	768
Task 3	26.1823	8.875	1894.3538	780
Task 3	24.0327	7.931	2375.6233	792
Task 3	21.7524	8.462	2278.9055	804
Task 3	25.7413	9.722	2430.9709	816
Task 3	22.823	8.775	1929.657	828
Task 3	22.8312	7.195	2397.0591	840
Task 3	24.2262	6.212	2470.0706	852
Task 3	23.1188	7.531	2178.5745	864
Task 3	22.5751	8.108	2261.0586	876
Task 3	27.577	8.097	2467.7146	888
Task 3	27.5145	6.239	1901.0677	900
Task 3	26.5844	7.946	1844.1456	912
Task 3	24.9698	6.313	2526.0527	924
Task 3	26.1762	7.322	1875.0786	936
Task 3	24.3983	9.481	2509.4268	948
Task 3	26.2939	7.154	1815.1899	960
Task 3	22.0382	9.827	2162.1313	972
Task 3	23.0352	8.559	2106.6738	984
Task 3	24.1592	7.875	2032.6128	996
Task 3	22.9039	8.342	2279.2107	1008
Task 3	22.3118	8.086	2006.8066	1020

Кінець таблиці Г.3

Task 3	27.0603	6.495	2049.9836	1032
Task 3	22.5782	8.359	2008.1494	1044
Task 3	27.5706	9.698	2423.6953	1056
Task 3	26.8218	9.529	1897.5764	1068
Task 3	27.6576	6.818	1992.512	1080
Task 3	25.9368	8.871	2173.5327	1092
Task 3	25.9501	8.41	2036.1406	1104
Task 3	23.0992	9.594	2271.9475	1116
Task 3	24.7333	6.215	2021.7604	1128
Task 3	27.016	6.347	2066.2559	1140
Task 3	26.1987	8.665	2304.0647	1152
Task 3	27.1755	6.716	2567.4109	1164
Task 3	22.6939	8.374	2025.2272	1176
Task 3	23.0972	8.726	1823.8204	1188
Task 3	22.9727	9.65	1814.9458	1200

Таблиця Г.4 – Таблиця збору телеметричних даних з планувальника задач в другій серії експериментів після оновлення політик планувальника.

Task Name	CPU Usage (%)	Memory Usage (KB)	Execution Time (us)	Time (s)
Task 1	24.2231	9.098	2259.1758	12
Task 1	19.6759	7.596	2478.1855	24
Task 1	19.5083	8.806	2309.6406	36
Task 1	21.6181	6.08	2515.4663	48
Task 1	20.8037	6.751	1950.5286	60
Task 1	21.0686	8.899	2078.3259	72
Task 1	21.5821	7.301	2474.0474	84
Task 1	20.1719	6.826	2365.22	96
Task 1	22.7224	8.51	2259.8594	108
Task 1	21.3539	8.572	1912.0638	120
Task 1	22.7595	9.62	1875.1613	132
Task 1	20.5715	6.286	2092.3154	144
Task 1	20.135	6.561	1974.6501	156
Task 1	18.563	9.054	2216.9023	168
Task 1	21.4466	7.451	2000.505	180
Task 1	20.098	8.399	1834.2548	192
Task 1	19.6981	8.093	2082.4155	204
Task 1	21.2272	7.783	2230.3303	216
Task 1	20.5498	7.481	1737.5616	228
Task 1	20.5327	8.102	2328.5496	240
Task 1	20.5679	7.293	1770.6429	252

Продовження таблиці Г.4

Task 1	24.3777	9.195	1789.845	264
Task 1	21.4752	7.398	2173.8474	276
Task 1	22.1568	7.333	2512.6831	288
Task 1	20.4005	9.349	2426.8542	300
Task 1	21.5458	8.727	2132.9656	312
Task 1	24.1292	7.889	1796.8398	324
Task 1	23.7517	7.028	1924.1488	336
Task 1	21.3523	6.447	2038.5184	348
Task 1	22.7627	8.819	1897.1708	360
Task 1	20.9178	8.641	2094.061	372
Task 1	22.5985	7.125	2300.4729	384
Task 1	24.0796	9.776	2393.1624	396
Task 1	18.5597	6.907	1919.974	408
Task 1	20.7475	8.343	1952.9946	420
Task 1	20.4785	6.562	1979.1302	432
Task 1	21.7383	9.523	1878.2618	444
Task 1	21.968	7.428	2038.8724	456
Task 1	19.0245	8.682	2461.1565	468
Task 1	23.673	7.277	2297.6409	480
Task 1	23.9851	6.875	1881.3383	492
Task 1	20.5189	5.82	1891.3602	504
Task 1	21.655	6.345	2106.0242	516
Task 1	18.9631	6.206	1993.4858	528
Task 1	20.9759	6.02	1989.8848	540
Task 1	24.2437	8.521	2514.9536	552
Task 1	21.0453	9.019	2375.1201	564
Task 1	21.6302	7.288	2004.1794	576
Task 1	19.3671	6.146	2391.4045	588
Task 1	23.6735	8.968	2347.1289	600
Task 1	20.1015	7.393	2155.9272	612
Task 1	21.4555	9.46	2267.2205	624
Task 1	24.315	9.44	2273.3853	636
Task 1	19.8396	7.599	2025.3956	648
Task 1	20.2301	7.153	1735.9502	660
Task 1	18.7202	6.659	2520.1052	672
Task 1	21.4758	9.089	2223.3354	684
Task 1	24.2734	7.851	1987.2603	696
Task 1	21.3371	8.988	2002.7021	708
Task 1	24.2058	7.823	2300.3264	720
Task 1	19.0056	9.281	1828.7494	732
Task 1	23.0059	8.161	2470.2266	744

Продовження таблиці Г.4

Task 1	21.829	8.585	2043.6696	756
Task 1	23.1932	8.889	1745.3864	768
Task 1	23.561	9.204	2077.1418	780
Task 1	18.5685	5.967	1911.7708	792
Task 1	18.6118	8.63	2346.3599	804
Task 1	23.4639	8.795	2392.8083	816
Task 1	22.2979	7.044	2156.0005	828
Task 1	23.9925	9.707	1749.4146	840
Task 1	19.0919	6.813	2398.2649	852
Task 1	19.2989	9.478	2380.064	864
Task 1	18.8727	7.99	1852.822	876
Task 1	20.809	6.818	2420.665	888
Task 1	23.1755	7.819	2286.8862	900
Task 1	21.4797	7.876	2044.6584	912
Task 1	21.4417	8.229	2355.1245	924
Task 1	19.0493	7.738	1762.9893	936
Task 1	23.2689	7.338	1899.7222	948
Task 1	21.2874	7.927	2324.106	960
Task 1	18.661	6.949	2250.3867	972
Task 1	21.2448	5.904	2213.9238	984
Task 1	22.2783	8.055	1724.28	996
Task 1	20.5034	8.567	2355.8938	1008
Task 1	21.6871	6.336	2326.1204	1020
Task 1	22.4265	9.753	2010.9423	1032
Task 1	21.61	7.659	2048.0642	1044
Task 1	19.4087	7.735	1877.5172	1056
Task 1	22.176	7.579	2369.8223	1068
Task 1	22.1259	6.724	2391.5632	1080
Task 1	21.1701	6.838	2158.564	1092
Task 1	22.7849	6.367	2026.4941	1104
Task 1	22.284	6.865	1968.986	1116
Task 1	24.3741	7.253	1993.2662	1128
Task 1	19.3651	9.278	1943.2776	1140
Task 1	21.9129	8.403	2140.3142	1152
Task 1	22.6674	7.381	2111.1267	1164
Task 1	19.7023	8.778	2453.5759	1176
Task 1	19.101	9.667	1889.7122	1188
Task 1	18.5741	8.777	2156.4766	1200
Task 2	23.7577	9.632	2468.5176	12
Task 2	20.634	8.144	1858.95	24
Task 2	21.0651	8.854	2031.5724	36

Продовження таблиці Г.4

Task 2	20.1775	6.687	2335.0071	48
Task 2	20.2402	6.442	1851.467	60
Task 2	21.8758	6.241	1999.9189	72
Task 2	23.8026	6.42	1976.6398	84
Task 2	19.4828	9.11	1883.389	96
Task 2	19.5528	8.748	1793.019	108
Task 2	19.4086	8.786	2402.6108	120
Task 2	21.8232	6.826	2324.0571	132
Task 2	18.9697	9.45	2062.249	144
Task 2	22.7268	8.988	2130.6707	156
Task 2	20.6388	9.469	1754.5052	168
Task 2	21.4083	8.009	2011.3083	180
Task 2	22.2946	8.896	1859.0721	192
Task 2	23.2934	6.6	2501.9165	204
Task 2	19.4724	8.975	1743.1281	216
Task 2	21.3391	6.908	2175.9104	228
Task 2	19.6135	7.635	2461.3762	240
Task 2	19.2837	8.019	2344.248	252
Task 2	20.3158	6.994	2255.9775	264
Task 2	18.8968	7.458	2155.3535	276
Task 2	23.583	8.855	2456.6399	288
Task 2	19.2291	6.685	2495.178	300
Task 2	20.0752	6.211	2180.5371	312
Task 2	21.8628	8.599	1887.1364	324
Task 2	21.3589	6.742	1832.3628	336
Task 2	24.1609	7.034	1992.1918	348
Task 2	20.1568	8.316	1804.3105	360
Task 2	18.5181	8.266	1908.7678	372
Task 2	22.5978	6.485	1976.0662	384
Task 2	21.6102	8.505	2072.8083	396
Task 2	20.467	5.994	2014.8118	408
Task 2	19.6442	7.399	1757.74	420
Task 2	23.4559	8.271	2475.2437	432
Task 2	21.6084	9.534	2174.5188	444
Task 2	19.4119	6.934	1899.5269	456
Task 2	22.5223	5.934	2371.5679	468
Task 2	18.5892	7.287	1917.3738	480
Task 2	20.6866	7.559	2334.824	492
Task 2	19.8979	5.824	1945.9633	504
Task 2	22.9336	6.808	2087.7256	516
Task 2	19.1543	9.722	1889.9808	528

Продовження таблиці Г.4

Task 2	19.8762	7.018	2391.2214	540
Task 2	23.3312	8.35	2441.625	552
Task 2	21.1593	7.961	2243.7339	564
Task 2	22.8073	8.762	2180.5615	576
Task 2	18.7083	7.348	1981.2054	588
Task 2	23.6179	7.2	1873.9894	600
Task 2	21.9807	6.327	2377.0854	612
Task 2	19.7044	8.594	2376.6094	624
Task 2	21.2144	6.986	2142.2917	636
Task 2	19.1644	6.062	1730.579	648
Task 2	22.679	5.847	1825.2704	660
Task 2	22.5019	8.408	1865.6152	672
Task 2	23.9255	7.371	2111.969	684
Task 2	18.8769	7.623	2498.9014	696
Task 2	19.0356	9.456	2125.812	708
Task 2	24.356	6.389	2235.1033	720
Task 2	21.8728	8.939	1727.7346	732
Task 2	24.1618	9.571	2066.4238	744
Task 2	23.3316	7.092	2216.1943	756
Task 2	22.923	7.726	2472.5581	768
Task 2	20.5878	7.128	2227.1074	780
Task 2	22.9251	8.594	2120.6729	792
Task 2	22.3484	7.694	2433.9224	804
Task 2	20.3592	8.605	2165.4734	816
Task 2	24.1016	8.376	2247.5669	828
Task 2	19.5731	8.377	1749.134	840
Task 2	19.367	7.197	2116.4858	852
Task 2	23.3208	8.215	2301.364	864
Task 2	22.9455	6.12	1826.6132	876
Task 2	23.0122	7.611	2239.3269	888
Task 2	22.1689	8.541	2001.3962	900
Task 2	18.9639	8.743	2067.718	912
Task 2	24.1507	8.797	2401.7686	924
Task 2	22.6486	7.981	2278.7319	936
Task 2	19.0352	9.662	1826.1492	948
Task 2	22.945	7.377	2052.0439	960
Task 2	21.2484	7.846	2010.5026	972
Task 2	24.3885	6.175	1954.85	984
Task 2	20.7473	7.497	1928.2504	996
Task 2	19.4202	8.303	1841.8356	1008
Task 2	24.2701	9.703	2490.344	1020

Продовження таблиці Г.4

Task 2	20.638	5.88	1955.1918	1032
Task 2	18.5862	5.985	2445.2017	1044
Task 2	21.0458	6.71	2424.6082	1056
Task 2	23.7468	7.806	2278.7686	1068
Task 2	20.3037	9.785	1967.2524	1080
Task 2	20.3334	8.063	2514.4897	1092
Task 2	20.9781	8.785	2187.8369	1104
Task 2	20.7097	6.593	2013.2858	1116
Task 2	18.7916	7.577	2278.5366	1128
Task 2	20.2701	7.669	1873.611	1140
Task 2	22.8732	7.716	2519.0432	1152
Task 2	20.6903	9.501	2131.4031	1164
Task 2	22.008	7.943	2176.533	1176
Task 2	23.1775	9.739	2460.0945	1188
Task 2	22.8875	7.689	2205.7083	1200
Task 3	22.6738	9.78	2155.146	12
Task 3	20.395	9.487	2457.0671	24
Task 3	24.0279	6.795	1741.3458	36
Task 3	23.6072	6.22	2508.9966	48
Task 3	22.7086	6.228	1822.6702	60
Task 3	22.441	7.624	2247.4448	72
Task 3	23.3449	6.632	2078.802	84
Task 3	20.0721	8.909	1908.3525	96
Task 3	22.2219	7.75	1976.9207	108
Task 3	19.2077	8.725	2165.1682	120
Task 3	21.5311	6.011	1871.7188	132
Task 3	21.3359	6.657	2428.8196	144
Task 3	19.62	6.034	2505.3955	156
Task 3	19.9764	8.729	2347.7393	168
Task 3	19.2426	8.122	2107.6111	180
Task 3	23.9299	8.901	2507.8735	192
Task 3	23.8091	7.767	2295.5779	204
Task 3	20.5283	6.572	1932.645	216
Task 3	21.6358	7.15	1973.9666	228
Task 3	22.885	8.078	1983.012	240
Task 3	24.1955	6.637	2434.9844	252
Task 3	21.1446	8.193	2257.4058	264
Task 3	20.3669	8.283	2058.2939	276
Task 3	23.7328	7.627	2091.7539	288
Task 3	20.6853	7.331	2211.104	300
Task 3	20.6066	6.529	2059.5024	312

Продовження таблиці Г.4

Task 3	23.5921	7.71	1948.0872	324
Task 3	18.5044	5.984	2271.5908	336
Task 3	23.685	9.537	2311.4473	348
Task 3	22.1956	9.517	2276.4492	360
Task 3	19.9677	8.617	1830.8492	372
Task 3	21.7311	8.587	1760.0106	384
Task 3	24.1006	7.942	2423.5095	396
Task 3	21.6928	8.095	2045.7325	408
Task 3	20.6499	9.192	2494.0916	420
Task 3	22.4441	7.874	1994.5356	432
Task 3	23.3385	8.233	2034.7828	444
Task 3	19.3794	8.2	1904.8615	456
Task 3	23.2951	6.622	2125.2871	468
Task 3	20.1747	7.282	1812.6479	480
Task 3	21.3049	7.099	2032.7808	492
Task 3	20.0368	6.77	2191.3403	504
Task 3	24.0594	9.095	1911.9539	516
Task 3	18.9541	9.249	2251.8638	528
Task 3	22.4061	8.227	2305.0017	540
Task 3	20.9461	7.742	2242.6597	552
Task 3	23.5991	7.799	2041.5944	564
Task 3	21.3136	7.211	1749.6591	576
Task 3	23.5484	6.304	1738.9288	588
Task 3	23.8887	6.055	1996.3177	600
Task 3	21.0704	8.921	2501.709	612
Task 3	22.2845	9.608	1743.5552	624
Task 3	20.1378	7.018	2222.0903	636
Task 3	23.1456	7.644	1989.8239	648
Task 3	20.1295	9.648	2279.7817	660
Task 3	21.3978	7.875	1871.0474	672
Task 3	24.0808	7.056	2241.9639	684
Task 3	23.4983	9.449	1760.6698	696
Task 3	22.9329	6.169	1763.3188	708
Task 3	22.6851	8.451	2369.6392	720
Task 3	21.7836	6.105	2486.0593	732
Task 3	22.2279	7.405	2246.3706	744
Task 3	24.1367	9.669	1939.835	756
Task 3	22.6774	8.773	2476.8184	768
Task 3	22.1523	6.854	1841.86	780
Task 3	21.1157	8.452	1784.4006	792
Task 3	22.897	6.338	1996.8184	804

Кінець таблиці Г.4

Task 3	22.406	9.279	2030.083	816
Task 3	20.5934	6.989	2196.8948	828
Task 3	20.4429	6.526	2512.2925	840
Task 3	22.8703	9.362	2521.7776	852
Task 3	21.4022	9.405	2266.9519	864
Task 3	23.0129	7.199	2038.6404	876
Task 3	18.9909	6.565	2367.5149	888
Task 3	20.2127	6.929	2302.1941	900
Task 3	23.6973	7.591	1824.9042	912
Task 3	21.8173	7.17	2054.6074	924
Task 3	20.0408	8.506	2287.582	936
Task 3	21.5805	6.227	2325.5833	948
Task 3	19.8224	9.272	2341.4648	960
Task 3	19.9126	9.591	1932.291	972
Task 3	19.3739	7.287	2209.9932	984
Task 3	20.1299	9.443	1779.4323	996
Task 3	22.8089	6.887	2054.1069	1008
Task 3	18.7051	8.149	1959.1958	1020
Task 3	23.2716	8.845	1872.2438	1032
Task 3	19.5172	9.764	2051.2627	1044
Task 3	20.1879	7.873	2200.3494	1056
Task 3	24.1013	7.53	2222.8228	1068
Task 3	18.5148	6.119	2411.8516	1080
Task 3	21.8987	7.378	1928.5555	1092
Task 3	21.9907	6.668	2217.6836	1104
Task 3	22.5072	7.423	1810.6584	1116
Task 3	20.3851	7.044	2366.2942	1128
Task 3	21.9306	9.472	2398.8264	1140
Task 3	18.7486	8.708	2143.6101	1152
Task 3	22.8291	6.516	2010.222	1164
Task 3	18.6635	6.559	2473.083	1176
Task 3	18.769	6.234	1849.5507	1188
Task 3	23.493	6.433	1815.6268	1200

ДОДАТОК Д

Порівняльні таблиці поведінки фільтрів

Таблиця Д.1 – Порівняння поведінки фільтрів при режимних змінах

Time_sec	Sensor_Data	Kalman_Filter	KKF	Adaptive_KF	Hinf_Filter	ETKF
1	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123
2	49.86173569882882	50.213416073452926	50.213416073452926	49.931138660056774	50.4262773677844	50.49671415301123
3	50.647688538100695	50.39165488547389	50.39165488547389	50.31801487516154	50.45238119215656	50.49671415301123
4	51.52302985640802	50.84165551406326	50.84165551406326	50.892105866226615	50.584378789070485	50.49671415301123
5	49.765846625276666	50.41874364371871	50.41874364371871	50.42700920545653	50.48008887019351	50.49671415301123
6	49.76586304305082	50.16320732614738	50.16320732614738	50.15388583472261	50.386854367988015	50.49671415301123
7	51.57921281550739	50.71653034426986	50.71653034426986	50.684180529658256	50.54531611744295	50.49671415301123
8	50.76743472915291	50.73640991039906	50.73640991039906	50.716579852083996	50.5752282536835	50.49671415301123
9	49.53052561406505	50.26558445283847	50.26558445283847	50.224015041977154	50.43315862506478	50.49671415301123
10	50.542560043585965	50.37371778678803	50.37371778678803	50.367149234216456	50.44814411106918	50.49671415301123
11	49.53658230718754	50.04690398359193	50.04690398359193	50.01300420546548	50.32261178777273	50.49671415301123
12	49.53427024642974	49.84677645746603	49.84677645746603	49.69391413281421	50.213618199868606	50.49671415301123
13	50.24196227156603	50.001052727719106	50.001052727719106	49.977545837740216	50.21754843843798	50.49671415301123
14	48.0867197553422	49.25371901558106	49.25371901558106	49.20615379280139	49.92144407451889	50.49671415301123
15	48.275082167486964	48.87167060136232	48.87167060136232	48.754104205684904	49.69229574339176	50.49671415301123
16	49.437712470759024	49.092646698389714	49.092646698389714	49.01551184698322	49.656819671815164	50.49671415301123
17	48.987168879665575	49.051469400273085	49.051469400273085	49.00443646130458	49.56342232779153	50.49671415301123
18	50.31424733259527	49.54444301715711	49.54444301715711	49.441541320512634	49.66820907828034	50.49671415301123
19	49.09197592447879	49.367805200660996	49.367805200660996	49.25668295910986	49.587750164091524	50.49671415301123
20	48.58769629866471	49.06325988735429	49.06325988735429	48.89876805756984	49.448063990169935	50.49671415301123
21	46.3190838920294	47.99196595028267	47.99196595028267	48.195314511143735	49.01089751476696	46.8953087556131
22	44.796801329562115	46.74461137463762	46.74461137463762	47.46765000343535	48.42200930862734	46.8953087556131
23	45.060775384219134	46.08726166776737	46.08726166776737	46.509293235912395	47.95223466555188	46.8953087556131
24	43.71772663240789	45.162223142845	45.162223142845	45.08649340302008	47.36034606024453	44.93422671917952
25	44.51005554792734	44.90762460727392	44.90762460727392	44.836226857349935	46.96190845795561	44.93422671917952
26	45.09983033073888	44.982659454303075	44.982659454303075	44.902352371217184	46.70159634702432	44.93422671917952

Продовження таблиці Д.1

27	43.964105780319926	44.58502811565082	44.58502811565082	44.55836691717483	46.3188881055103	44.93422671917952
28	45.3381282165111	44.879029510857336	44.879029510857336	44.81697930330772	46.18177097325353	44.93422671917952
29	44.45942517907307	44.71522092971616	44.71522092971616	44.57779519053464	45.940969235482854	44.93422671917952
30	44.73747562518605	44.723908900293466	44.723908900293466	44.67305739342869	45.77270545785726	44.93422671917952
31	44.45846404899354	44.620282361745204	44.620282361745204	44.54730525043011	45.588955351148556	44.93422671917952
32	46.66705036605804	45.419316445320675	45.419316445320675	45.34780447564432	45.739690189626266	44.93422671917952
33	44.98785249773586	45.250878010076505	45.250878010076505	45.208067329241835	45.634570573520364	44.93422671917952
34	44.04806016393969	44.7813121123436	44.7813121123436	44.80986790215446	45.412748444598414	44.93422671917952
35	45.74029042089287	45.15568593112804	45.15568593112804	45.12949820158352	45.458544765385426	44.93422671917952
36	43.90124071502608	44.66596531719839	44.66596531719839	44.75523839706327	45.24080473443954	44.93422671917952
37	40.167090876003805	42.909657807668104	42.909657807668104	43.80505218501202	44.53140379713976	41.14653970494582
38	38.43226390089618	41.1617360453749	41.1617360453749	42.92770803430569	43.67862761344736	39.75983581541964
39	38.93745116088125	40.29340146590759	40.29340146590759	42.19119411041474	43.01571987242897	39.75983581541964
40	40.1574889886953	40.24034283813592	40.24034283813592	41.47822885402565	42.61608382778914	39.75983581541964
41	40.590773263996326	40.37714674243495	40.37714674243495	41.220902530965986	42.3329059986576	39.75983581541964
42	40.13709462495198	40.28343322761589	40.28343322761589	40.93624542950218	42.02588869413242	39.75983581541964
43	39.90748137408941	40.13666605902714	40.13666605902714	40.48264073093776	41.72969388470438	39.75983581541964
44	39.75911704352857	39.98927537724589	39.98927537724589	39.90027514860894	41.45416859901494	39.75983581541964
45	38.817182407706056	39.53170410888129	39.53170410888129	38.862032722831636	41.08546614762626	39.75983581541964
46	39.42412463328424	39.489706350701525	39.489706350701525	39.09528935311149	40.85317794786082	39.75983581541964
47	39.63148898323217	39.54505661786044	39.54505661786044	39.32992357179619	40.68236179875227	39.75983581541964
48	40.84569778097513	40.05281158453961	40.05281158453961	39.89363052476803	40.705199383811006	39.75983581541964
49	40.27489463165477	40.139510186264566	40.139510186264566	40.06861596397837	40.645034306861334	39.75983581541964
50	38.58956787570981	39.53443099258002	39.53443099258002	39.49369561536622	40.35763961601183	39.75983581541964
51	40.25926717551584	39.817398487652284	39.817398487652284	39.79575666095353	40.34388521089052	39.75983581541964
52	39.691934175666944	39.76841870033032	39.76841870033032	39.75520215563743	40.25272961060549	39.75983581541964
53	39.458462399755234	39.64741541707761	39.64741541707761	39.59893114052723	40.141675402976645	39.75983581541964
54	40.4893410310727	39.97609324475507	39.97609324475507	39.99238525929846	40.1902859093767	39.75983581541964
55	40.82479961799676	40.30741820085115	40.30741820085115	40.57148197419491	40.279003431299685	39.75983581541964

Кінець таблиці Д.1

56	40.74502409529296	40.478254379692984	40.478254379692984	40.68442284095872	40.34416230525155	39.75983581541964
57	39.32862598142189	40.0294530149417	40.0294530149417	40.15772526442762	40.202170310540396	39.75983581541964
58	39.75263009931903	39.92138461430657	39.92138461430657	39.98421317489201	40.139315728150386	39.75983581541964
59	40.26501074512285	40.05553220208926	40.05553220208926	40.125402083677905	40.156890369139695	39.75983581541964
60	40.78043610169789	40.33852613295175	40.33852613295175	40.469616837887884	40.24407435365096	39.75983581541964

Таблиця Д.2 – Порівняння поведінки фільтрів при режимних змінах і атаках

Tim_sec	Sensor_Data_Attack	Kalman_Filter	KKF	Adaptive_KF	Hinf_Filter	ETKF
1	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123	50.49671415301123
2	49.86173569882882	50.214461538461535	50.214461538461535	49.93092981865409	50.42900618556701	50.49671415301123
3	459.97040207878206	218.39093600058854	50.214461538461535	50.0469584272747	98.71291460921711	311.072074050134
4	51.52302985640802	152.01857668318135	51.47109638380637	50.04746712124784	92.89461816167423	189.1138705807939
5	49.765846625276666	111.82357805854198	50.538842729004216	50.047352304721365	87.40007179121004	130.78955347254566
6	49.76586304305082	87.53591686446838	50.197602584850785	50.04723497445092	82.4878705264556	98.32645765712263
7	51.57921281550739	73.48566311312113	50.763865882759326	50.04790129685602	78.38028421653503	79.89947703076649
8	50.76743472915291	64.6145702187972	50.76630391168801	50.733436939314956	74.66207898930436	68.48661372582623
9	49.53052561406505	58.72495071138559	50.28043307641945	50.1319505915048	71.24435513849569	61.07625399301519
10	50.542560043585965	55.52948498010466	50.382016744148295	50.315794563331025	68.4083331896757	56.960976091574764
11	49.53658230718754	53.19121786177867	50.052999963832136	50.00843917555611	65.80995127860228	54.06337840846969
12	49.53427024642974	51.76191186771391	49.848756935397546	49.74191207785655	63.559137122930146	52.293479242314625
13	50.24196227156603	51.16777392493498	50.00151307453868	50.01878415846773	61.71228226770275	52.293479242314625
14	48.0867197553422	49.96624616271532	49.25524558419286	49.22958426072924	59.8193017301289	50.31357478497664
15	48.275082167486964	49.307955323061705	48.874514513231844	48.76129191582664	58.213207880084454	50.31357478497664
16	49.437712470759024	49.35950401392828	49.09527482001044	49.02024019232337	56.99066506534418	50.31357478497664
17	48.987168879665575	49.21525399889329	49.05417667230501	49.00847730238329	55.87479859242147	50.31357478497664
18	50.31424733259527	49.64262993089302	49.54443572752043	49.43798069918396	55.09816334733267	50.31357478497664
19	49.09197592447879	49.42688972369802	49.36702932083915	49.253921851110285	54.25924893221047	50.31357478497664
20	48.58769629866471	49.100177847375925	49.06368620260513	48.89619663286072	53.46737589444008	50.31357478497664

Продовження таблиці Д.2

21	46.3190838920294	48.01482921195721	47.992583426170356	48.192058464907035	52.46877793914807	47.55984954863415
22	44.796801329562115	46.75979781186995	46.74623649738488	47.46413037641435	51.397123974598486	46.22294105210524
23	45.060775384219134	46.09621679820264	46.087949656782676	46.507161248366806	50.511431333073666	46.22294105210524
24	43.71772663240789	45.16856979188958	45.16353004279878	45.087240795891475	49.56214258729485	44.98130012648171
25	44.51005554792734	44.91147191414604	44.90839962342634	44.83599082651698	48.85591156620955	44.98130012648171
26	45.09983033073888	44.9850710548362	44.983198150194376	44.90231893230698	48.33084806109624	44.98130012648171
27	43.964105780319926	44.58489540758313	44.5837536627714	44.556344379454586	47.71979222925791	44.98130012648171
28	45.3381282165111	44.87967933264676	44.878983311553725	44.81616305236598	47.38708053345665	44.98130012648171
29	-196.76001839998307	-49.45360808755072	44.878983311553725	44.64045771241802	13.252793508465714	-08.72836814780847
30	44.73747562518605	-2.681534673114285	44.745217015251185	44.64053264546111	17.655107127593254	-36.63916039789649
31	44.45846404899354	9.625846376140476	44.588913800606754	44.640391806458496	21.40282188715078	-6979860337392054
32	46.66705036605804	24.087446948505793	45.5125646334607	44.642029098034804	24.93557545613704	17.082012537634352
33	44.98785249773586	32.24755707661749	45.29851046789213	44.64231941159401	27.739522946419207	28.08287698383347
34	44.04806016393969	36.85509156287341	44.80228587399102	44.10392590883825	30.020015228038808	34.33872123215029
35	45.74029042089287	40.3236550032594	45.170809886192906	44.91708510576096	32.217954782316056	38.79560288051523
36	43.90124071502608	41.71981790056817	44.67346800912935	44.55434431845888	33.85132168110679	40.78929008852951
37	40.167090876003805	41.11478727507474	42.913744779793284	43.77764475969954	34.73479214219043	40.78929008852951
38	38.43226390089618	40.06667799477816	41.16274354038064	43.01512423887271	35.25145273836298	40.78929008852951
39	38.93745116088125	39.62683619680925	40.29490012426357	42.35294433315599	35.76718265420021	40.78929008852951
40	40.1574889886953	39.83497705594934	40.242234213628514	41.56385293347715	36.38138361287414	40.78929008852951
41	40.590773263996326	40.129729106453716	40.37800026294937	41.25561690589914	36.96983008299073	40.78929008852951
42	40.13709462495198	40.13373874213053	40.285087160383135	40.95641289253525	37.41308173678124	40.78929008852951
43	39.90748137408941	40.04639377660347	40.13865720196568	40.505791082900345	37.762199785106546	40.78929008852951
44	39.75911704352857	39.93458902474695	39.99083376376584	39.92839338385111	38.041531432295955	40.78929008852951
45	38.817182407706056	39.49946661806709	39.53375392108262	38.861590331354336	38.15037662465513	40.78929008852951
46	39.42412463328424	39.46844378782532	39.48934572668539	39.08938217458408	38.327894928804525	40.78929008852951
47	39.63148898323217	39.53151342721235	39.544255498262416	39.32266942666642	38.509954831477	40.78929008852951
48	203.37228435398197	103.49301760057841	39.544255498262416	39.497158206274655	61.5609532839502	166.3423455305561
49	40.275465464	78.81149735846705	40.24275773552139	39.498031496823096	58.5840588010476	102.61292633606931

Кінець таблиці Д.2

50	38.59345345	63.109499274592764	39.33702163536135	39.496948416968756	55.78849558441326	75.05993457894166
51	40.2647457	54.18932430871435	39.74666816295656	39.49790577562192	53.61730598989251	60.96979255972035
52	39.6967892	48.528959144189265	39.72345561553318	39.4981575942685	51.669994199174084	52.5487361396802
53	39.467867867	44.98854447897499	39.61874431419367	39.46233857536078	49.962796404252515	47.41314820988776
54	40.49657867843	43.23236578280273	39.96114999208792	40.06025128004594	48.63831278256968	44.7053478944147
55	40.8278542345	42.29060663938789	40.29726799029669	40.646595053207015	47.54515856656874	43.1874938688495
56	40.7591245785	41.68917198159586	40.47417232991316	40.71782819737522	46.59506146845295	43.1874938688495
57	39.33913477	40.76817907065566	40.02734779681117	40.21472775803722	45.57926267780542	41.3702884491963
58	39.758953254	40.370693972724304	39.91906069347943	40.02606682280386	44.76421685518663	41.3702884491963
59	40.2797324325	40.33138423363917	40.056069745594385	40.151119268783056	44.13583675181867	41.3702884491963
60	40.789289654	40.50651853659696	40.33868854961757	40.480654965641584	43.66662462491893	41.3702884491963

ДОДАТОК Е

Текст програми аналізатора ймовірності відмов у реальному часі для операційних систем кіберфізичних систем з інтенсивними змінами станів

rt_failure_analyzer_pro/FreeRTOSConfig.h

```
#pragma once

#define configCPU_CLOCK_HZ ((unsigned long)64000000)
#define configTICK_RATE_HZ ((TickType_t)1000)
#define configUSE_PREEMPTION 1
#define configUSE_IDLE_HOOK 1
#define configUSE_TIMERS 1
#define configSUPPORT_STATIC_ALLOCATION 1
#define configSUPPORT_DYNAMIC_ALLOCATION 0
#define configMINIMAL_STACK_SIZE(256)
#define configTOTAL_HEAP_SIZE((size_t)(0))
#define configUSE_TRACE_FACILITY 0
#define configGENERATE_RUN_TIME_STATS 0
```

rt_failure_analyzer_pro/src/actions.c

```
#include "actions.h"
#include "platform.h"
using System;

void actions_log(const char* msg) { plat_uart_write(msg); plat_uart_write("\r\n"); }
void actions_restart_task(TaskHandle_t h) { if (h) { vTaskSuspend(h); vTaskResume(h); } }
void actions_restart_task_by_id(uint16_t id)
{
    extern TaskHandle_t actions_lookup_task_by_id(uint16_t id);
```

```

    TaskHandle_t h = actions_lookup_task_by_id(id); actions_restart_task(h);
}

```

```

void actions_raise_alarm(uint16_t code) { plat_alarm(code); }

```

```

void actions_feed_hw_wdt(void) { plat_hw_wdt_feed(); }

```

rt_failure_analyzer_pro/src/actions.h

```

#pragma once

```

```

#include <stdint.h>

```

```

#include "FreeRTOS.h"

```

```

#include "task.h"

```

```

using System;

```

```

TaskHandle_t actions_lookup_task_by_id(uint16_t id);

```

```

void actions_log(const char* msg);

```

```

void actions_restart_task(TaskHandle_t h);

```

```

void actions_restart_task_by_id(uint16_t id);

```

```

void actions_raise_alarm(uint16_t code);

```

```

void actions_feed_hw_wdt(void);

```

rt_failure_analyzer_pro/src/cbor.c

```

#include "cbor.h"

```

```

#include <string.h>

```

```

using System;

```

```

static int put(cbor_w_t* w, const void* data, size_t len)

```

```

{

```

```

    if (w->w + len > w->cap) return -1; memcpy(&w->p[w->w], data, len); w->w += len;
    return 0;

```

```

}

```

```

void cbor_init(cbor_w_t* w, uint8_t* buf, size_t cap)

```

```

{
    w->p = buf; w->cap = cap; w->w = 0;
}
int cbor_u64(cbor_w_t* w, uint64_t v)
{
    if (v < 24) { uint8_t b = (uint8_t)(0x00 | v); return put(w, &b, 1); }
    else { uint8_t b = 0x1B; if (put(w, &b, 1)) return -1; return put(w, &v, 8); }
}
int cbor_f64(cbor_w_t* w, double v)
{
    uint8_t b = 0xFB; if (put(w, &b, 1)) return -1; return put(w, &v, 8);
}
int cbor_text(cbor_w_t* w, const char* s)
{
    size_t n = strlen(s); if (n < 24)
    {
        uint8_t b = (uint8_t)(0x60 | n);
        if (put(w, &b, 1)) return -1;
    }
    else
    {
        uint8_t b = 0x78; if (put(w, &b, 1)) return -1; uint8_t l = (uint8_t)n;
        if (put(w, &l, 1)) return -1;
    }
    return put(w, s, n);
}
int cbor_map(cbor_w_t* w, uint32_t count)
{

```

```

    if (count < 24) {
        uint8_t b = (uint8_t)(0xA0 | count); return put(w, &b, 1);
    }
    return -1;
}

rt_failure_analyzer_pro/src/cbor.h
#pragma once
#include <stdint.h>
#include <stddef.h>
/* Мінімалістичний CBOR-енкодер (тільки unsigned, float, text, map) */
using System;

typedef struct { uint8_t* p; size_t cap; size_t w; }
cbor_w_t;
void cbor_init(cbor_w_t* w, uint8_t* buf, size_t cap);
int cbor_u64(cbor_w_t* w, uint64_t v);
int cbor_f64(cbor_w_t* w, double v);
int cbor_text(cbor_w_t* w, const char* s);
int cbor_map(cbor_w_t* w, uint32_t count);

rt_failure_analyzer_pro/src/collector.c
#include "collector.h"
#include "metrics.h"
#include "platform.h"
#ifndef COL_MAX_TASKS
#define COL_MAX_TASKS 12
#endif
using System;

```

```

static task_mon_t REG[COL_MAX_TASKS];
static uint32_t REG_N = 0;
static volatile uint32_t idle_ticks = 0;
static uint32_t prev_idle_ticks = 0, prev_ms = 0;
static int32_t last_temp_mdegc = 0;
static int32_t last_vcc_mv = 3300;
bool collector_init(void)
{
    REG_N = 0; prev_ms = plat_now_ms(); return true;
}
bool collector_register_task(TaskHandle_t h, uint16_t id, uint16_t
stack_thresh_words, uint32_t hb_period_ms)
{
    if (REG_N >= COL_MAX_TASKS) return false;
    REG[REG_N++] =
        (task_mon_t){.id = id,.h = h,.stack_low_thresh_words = stack_thresh_words,.hb
        _period_ms = hb_period_ms,.last_hb_ms = plat_now_ms()}
    ;
    return true;
}
void collector_hb_ping(uint16_t id)
{
    for (uint32_t i = 0; i < REG_N; i++)
        if (REG[i].id == id) { REG[i].last_hb_ms = plat_now_ms(); return; }
}
void collector_idle_hook_tick(void) { idle_ticks++; }
void collector_feed_temp_mdegc(int32_t mdegc)
{

```

```

    last_temp_mdegc = mdegc;
}
void collector_feed_vcc_mv(int32_t mv) { last_vcc_mv = mv; }
void collector_tick(void)
{
    uint32_t now_ms = plat_now_ms(); uint32_t dt = now_ms - prev_ms;
    if (dt < 50) return; prev_ms = now_ms;
    uint32_t curr_idle = idle_ticks; uint32_t idle_delta = curr_idle -
    prev_idle_ticks; prev_idle_ticks = curr_idle;
    uint32_t ticks = dt / portTICK_PERIOD_MS; if (!ticks) ticks = 1;
    uint32_t cpu_load = 100 - (100 * idle_delta) / ticks;
    if (cpu_load > 85)
    {
        metric_evt_t e = {MET_CPU_HIGH, plat_now_us(), 0,
(uint16_t)cpu_load}; metrics_push(&e);
    }
    uint32_t isr_cnt = metrics_isr_take_and_clear();
    uint32_t isr_rate = (dt == 0) ? 0 : (isr_cnt * 1000u) / dt;
    if (isr_rate > 1200)
    {
        metric_evt_t e = {MET_ISR_RATE, plat_now_us(), 0,
(uint16_t)(isr_rate>65535?65535:isr_rate)}; metrics_push(&e);
    }
    for (uint32_t i = 0; i < REG_N; i++)
    {
        UBaseType_t wm = uxTaskGetStackHighWaterMark(REG[i].h);
        if (wm <= REG[i].stack_low_thresh_words)
        {

```

```

        metric_evt_t e={MET_STACK_LOW, plat_now_us(), REG[i].id,
(uint16_t)wm}; metrics_push(&e);
    }
    if (REG[i].hb_period_ms)
    {
        uint32_t age = now_ms - REG[i].last_hb_ms;
        if (age > (REG[i].hb_period_ms * 3u))
        {
            metric_evt_t e={MET_HEARTBEAT_MISS, plat_now_us(),
REG[i].id, 1}; metrics_push(&e);
            REG[i].last_hb_ms = now_ms;
        }
    }
    if (last_temp_mdegc > 80000)
    {
        metric_evt_t e={MET_TEMP_HIGH,
plat_now_us(), 0, (uint16_t)((last_temp_mdegc-80000)/100)};
        metrics_push(&e);
    }
    if (last_vcc_mv < 3000)
    {
        metric_evt_t e={MET_VOLT_LOW,
plat_now_us(), 0, (uint16_t)(3000-last_vcc_mv)}; metrics_push(&e);
    }
}

```

rt_failure_analyzer_pro/src/collector.h

#pragma once

```

#include <stdint.h>
#include <stdbool.h>
#include "FreeRTOS.h"
#include "task.h"
using System;

typedef struct {
    uint16_t id; TaskHandle_t h;
    uint16_t stack_low_thresh_words;
    uint32_t hb_period_ms; uint32_t last_hb_ms;
}
task_mon_t;
bool collector_init(void);
bool collector_register_task(TaskHandle_t h, uint16_t id, uint16_t
stack_thresh_words, uint32_t hb_period_ms);
void collector_hb_ping(uint16_t id);
void collector_tick(void);
void collector_idle_hook_tick(void);
/* Температура/напруга (з платформи або АЦП) – опціонально */
void collector_feed_temp_mdegc(int32_t mdegc);
void collector_feed_vcc_mv(int32_t mv);
rt_failure_analyzer_pro/src/config.c
#include "config.h"
#include "platform.h"
#include <string.h>
#define CFG_MAGIC 0xA15EFACEu
#define CFG_OFF 0x0000
#define CFG_LEN (sizeof(cfg_blob_t))

```

```

using System;
using System.Reflection.PortableExecutable;

static uint32_t crc32_update(uint32_t c, uint8_t b)
{
    c ^= b;
    for (int k = 0; k < 8; k++) c = (c >> 1) ^ (0xEDB88320u & (-(int)(c & 1)));
    return c;
}

uint32_t config_crc32(const void* data, uint32_t len)
{
    const uint8_t* p = (const uint8_t*)data; uint32_t c = ~0u;
    for (uint32_t i = 0; i < len; i++) c = crc32_update(c, p[i]); return ~c;
}

bool config_load_or_default(cfg_blob_t* out)
{
    cfg_blob_t tmp; if (plat_flash_read(CFG_OFF, &tmp, CFG_LEN) != 0) goto
    dflt;
    if (tmp.magic != CFG_MAGIC) goto dflt;
    uint32_t crc = config_crc32(&tmp, sizeof(cfg_blob_t) - sizeof(uint32_t));
    if (crc != tmp.crc32) goto dflt;
    *out = tmp; return true;
dflt:
    /* значення за замовчуванням */
    memset(out, 0, sizeof(*out));
    out->magic = CFG_MAGIC;
    out->est = (estimator_cfg_t){
        .thr_warn = 0.3f, .thr_crit = 0.7f,

```

```

.w_deadline = 0.30f,.w_heartbeat = 0.10f,.w_stack = 0.10f,
.w_cpu = 0.15f,.w_isr = 0.10f,.w_bus = 0.10f,.w_temp = 0.10f,.w_vcc = 0.05f,
.alpha = 0.2f,
.deadline_bad_ms = 50.f,.stack_min_words = 128,.cpu_high_pct = 85,.isr_high_
rate = 1200,
.temp_high_mdegc = 80000,.vcc_low_mv = 3000,
.trans_ok_warn = 0.2f,.trans_warn_crit = 0.2f,
.trans_warn_ok = 0.2f,.trans_crit_warn = 0.2f, .hysteresis_ms = 2000
}
;
out->pol = (policy_cfg_t){
.thr_warn = 0.3f,.thr_crit = 0.7f,.crit_alarm_code = 1001,
.restart_task_id_on_crit = 2,.cooldown_ms_log = 1000,.cooldown_ms_restart =
5000 }
;
out->crc32 = config_crc32(out, sizeof(cfg_blob_t) - sizeof(uint32_t));
return false;
}
bool config_save(const cfg_blob_t* cfg)
{
    cfg_blob_t tmp = *cfg; tmp.crc32 = config_crc32(&tmp, sizeof(cfg_blob_t) -
sizeof(uint32_t));
    if (plat_flash_erase(CFG_OFF, CFG_LEN) != 0) return false;
    return plat_flash_write(CFG_OFF, &tmp, CFG_LEN) == 0;
}
void config_apply(const cfg_blob_t* cfg)
{
    extern void estimator_init(const estimator_cfg_t*);

```

```

extern void policy_init(const policy_cfg_t*);
estimator_init(&cfg->est);
policy_init(&cfg->pol);
}

rt_failure_analyzer_pro/src/config.h
#pragma once
#include <stdint.h>
#include <stdbool.h>
#include "estimator.h"
#include "policy.h"
using System;

typedef struct {
    uint32_t magic;
    estimator_cfg_t est;
    policy_cfg_t pol;
    uint32_t crc32;
}
cfg_blob_t;
bool config_load_or_default(cfg_blob_t* out);
bool config_save(const cfg_blob_t* cfg);
uint32_t config_crc32(const void* data, uint32_t len);
void config_apply(const cfg_blob_t* cfg);

rt_failure_analyzer_pro/src/debug_ui.c
#include "debug_ui.h"
#include "platform.h"
#include "metrics.h"
#include "actions.h"

```

```

#include "config.h"
#include "secure_hmac.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

using System;
using static OneOf.Types.TrueFalseOrNull;
using static Xla.BufferAllocationProto.Types;

static estimator_cfg_t* E; static policy_cfg_t* P; static risk_out_t* LAST;
static cfg_blob_t CFG;
static uint32_t rate_counter = 0, last_tick = 0;
void debug_ui_set_refs(estimator_cfg_t* e, policy_cfg_t* p, risk_out_t*
last)
{ E = e; P = p; LAST = last; config_load_or_default(&CFG); }
static void outln(const char* s) { plat_uart_write(s); plat_uart_write("\r\n"); }
static int rate_ok(void)
{
    uint32_t now = plat_now_ms();
    if (now - last_tick > 1000) { last_tick = now; rate_counter = 0; }
    if (++rate_counter > 20) return 0; return 1;
}
static int auth_ok(const char* token)
{
    if (!token) return 0; uint8_t mac[32]; demo_hmac_sha256((const
uint8_t*)"key",3,(const uint8_t*)token,strlen(token),mac);
    return mac[0] != 0; /* демо-перевірка */
}

```

```

static void cmd_get_state(void)
{
    char b[160]; snprintf(b, sizeof(b), "p=%.3f state=%u pend=%lu
    thr=%0.2f/%0.2f",
    (double)LAST->p_fail, (unsigned)LAST->state, (unsigned
    long)metrics_pending(),(double)E->thr_warn,(double)E->thr_crit); outln(b);
}
static void cmd_set_thr(char* a, char* b)
{
    if (!a || !b) return; float w = atof(a), c = atof(b);
    E->thr_warn = w; E->thr_crit = c; P->thr_warn = w; P->thr_crit = c;
    outln("ok");
}
static void cmd_dump_cfg(void) { outln("cfg: dumped"); }
static void cmd_save_cfg(void)
{
    CFG.est = *E; CFG.pol = *P; CFG.magic = 0xA15EFACEu; CFG.crc32 = 0;
    if (config_save(&CFG)) outln("cfg: saved"); else outln("cfg: save error");
}
static void cmd_inject(char* what, char* a, char* b)
{
    if (!what) return;
    if (strcmp(what, "miss") == 0 && a && b)
    {
        metric_evt_t e={MET_DEADLINE_MISS, plat_now_us(),
        (uint16_t)atoi(a), (uint16_t)atoi(b)}; metrics_push(&e); outln("inj:miss");
    }
    else if (strcmp(what, "hbmiss") == 0 && a)

```

```

{
    metric_evt_t e={MET_HEARTBEAT_MISS, plat_now_us(),
(uint16_t)atoi(a), 1}; metrics_push(&e); outln("inj:hb");
}
else outln("err");
}
void debug_ui_task(void* arg)
{
    (void)arg; char line[160];
    for (; )
    {
        int n = plat_uart_read_line(line, sizeof(line), pdMS_TO_TICKS(200));
        if (n <= 0) continue;
        if (!rate_ok()) { outln("rate_limited"); continue; }
        char* tok = strtok(line, " \r\n\t"); if (!tok) continue;
        if (strcmp(tok, "auth") == 0)
        {
            char* t = strtok(NULL, " \r\n\t");
            outln(auth_ok(t) ? "ok" : "fail"); continue;
        }
        if (strcmp(tok, "get") == 0)
        {
            char* x = strtok(NULL, " \r\n\t"); if (x &&
strcmp(x, "state") == 0) cmd_get_state();
        }
        else if (strcmp(tok, "set") == 0)
        {
            char* x = strtok(NULL, " \r\n\t"); if (x &&

```

```

strcmp(x, "thr") == 0)
{
    char* a = strtok(NULL, " \r\n\t"); char*
b = strtok(NULL, " \r\n\t"); cmd_set_thr(a, b);
}
}
else if (strcmp(tok, "dump") == 0) { cmd_dump_cfg(); }
else if (strcmp(tok, "save") == 0) { cmd_save_cfg(); }
else if (strcmp(tok, "inject") == 0)
{
    char* w = strtok(NULL, " \r\n\t"); char*
a = strtok(NULL, " \r\n\t"); char* b = strtok(NULL, " \r\n\t");
    cmd_inject(w, a, b);
}
else outln("?");
}
}

```

rt_failure_analyzer_pro/src/debug_ui.h

```

#pragma once
#include <stdint.h>
#include "estimator.h"
#include "policy.h"
void debug_ui_task(void* arg);
void debug_ui_set_refs(estimator_cfg_t* e, policy_cfg_t* p, risk_out_t*last);

```

rt_failure_analyzer_pro/src/estimator.c

```

#include "estimator.h"
#include "platform.h"
using BaseApp.Data.DataContext.Entities;

```

using System;

```

static estimator_cfg_t C;
static float ema_dead = 0, ema_hb = 0, ema_stack = 0, ema_cpu = 0, ema_isr = 0,
ema_bus = 0, ema_temp = 0, ema_vcc = 0;
static uint8_t last_state = 1;
static uint32_t last_change_ms = 0;
static inline float clamp01(float x) { return x < 0 ? 0 : (x > 1 ? 1 : x); }
static inline void ema(float* e, float a, float s) { *e = a * s + (1.f - a) * (*e); }
void estimator_init(const estimator_cfg_t* cfg)
{
    C = *cfg;
    ema_dead = ema_hb = ema_stack = ema_cpu = ema_isr = ema_bus = ema_temp = e
ma_vcc = 0; last_state = 1; last_change_ms = plat_now_ms();
}
void estimator_step(risk_out_t* out)
{
    metric_evt_t e; int k = 0;
    while (k++ < 64 && metrics_pop(&e))
    {
        switch (e.kind)
        {
            case MET_DEADLINE_MISS:
                ema(&ema_dead, C.alpha,
(float)e.value); break;
            case MET_HEARTBEAT_MISS: ema(&ema_hb, C.alpha, 1.f); break;
            case MET_STACK_LOW:
                ema(&ema_stack, C.alpha,

```

```

(float)(C.stack_min_words - e.value)); break;
case MET_CPU_HIGH: ema(&ema_cpu, C.alpha, (float)e.value); break;
case MET_ISR_RATE: ema(&ema_isr, C.alpha, (float)e.value); break;
case MET_BUS_ERR: ema(&ema_bus, C.alpha, 1.f); break;
case MET_TEMP_HIGH:
    ema(&ema_temp, C.alpha, (float)e.value);
    break;
case MET_VOLT_LOW:
    ema(&ema_vcc, C.alpha, (float)e.value);
    break;
default: break;
}
}

float d = clamp01(ema_dead /
(C.deadline_bad_ms > 1 ? C.deadline_bad_ms : 50.f));
float h = clamp01(ema_hb);
float s = clamp01(ema_stack > 0 ? (ema_stack /
(float)C.stack_min_words) : 0);
float c = clamp01(ema_cpu / (float)(C.cpu_high_pct ? C.cpu_high_pct : 85));
float i = clamp01(ema_isr / (float)(C.isr_high_rate ? C.isr_high_rate : 1000));
float b = clamp01(ema_bus);
float t = clamp01(ema_temp / 50.f);
float v = clamp01(ema_vcc / 500.f);
float p = C.w_deadline * d + C.w_heartbeat * h + C.w_stack * s + C.w_cpu * c
+ C.w_isr * i + C.w_bus * b + C.w_temp * t + C.w_vcc * v;
if (p < 0) p = 0; if (p > 1) p = 1;
uint8_t target = (p >= C.thr_crit) ? 3 : (p >= C.thr_warn ? 2 : 1);
uint32_t now = plat_now_ms();

```

```

/* гістерезис: не даємо скакати стану надто часто */
if (target != last_state && (now - last_change_ms) < C.hysteresis_ms)
{
    target = last_state;
}
else if (target != last_state)
{
    last_change_ms = now;
}
/* квазі-марковська поправка: згладжуємо переходи між рівнями */
if (last_state == 1 && target == 2)
{
    p = (1.f - C.trans_ok_warn) * p +
    C.trans_ok_warn * (C.thr_warn + 0.05f);
}
if (last_state == 2 && target == 3)
{
    p = (1.f - C.trans_warn_crit) * p +
    C.trans_warn_crit * (C.thr_crit + 0.05f);
}
if (last_state == 2 && target == 1) { p = (1.f - C.trans_warn_ok) * p; }
if (last_state == 3 && target == 2) { p = (1.f - C.trans_crit_warn) * p; }
last_state = target;
out->p_fail = p; out->state = target;
}

rt_failure_analyzer_pro/src/estimator.h
#pragma once
#include <stdint.h>

```

```

#include "metrics.h"

using System;

typedef struct {
    float thr_warn, thr_crit;
    float w_deadline, w_heartbeat, w_stack, w_cpu, w_isr, w_bus, w_temp,
    w_vcc;
    float alpha;
    float deadline_bad_ms;
    uint16_t stack_min_words;
    uint16_t cpu_high_pct;
    uint16_t isr_high_rate;
    int32_t temp_high_mdegc;
    int32_t vcc_low_mv;
    /* квазі-марковська частина */
    float trans_ok_warn;
    float trans_warn_crit;
    float trans_warn_ok;
    float trans_crit_warn;
    uint32_t hysteresis_ms;
}

estimator_cfg_t;

typedef struct { float p_fail; uint8_t state; }
risk_out_t;

void estimator_init(const estimator_cfg_t* cfg);
void estimator_step(risk_out_t* out);

rt_failure_analyzer_pro/src/flash_log.c
#include "flash_log.h"

```

```

#include "platform.h"
#include <string.h>
using System;

static uint32_t g_base = 0, g_len = 0, g_w = 0;
bool flog_init(uint32_t base_off, uint32_t length)
{
    g_base = base_off; g_len = length; g_w = 0;
    plat_flash_erase(g_base, g_len);
    return true;
}
bool flog_append(const void* rec, uint16_t len)
{
    if (len + 2 > g_len) return false;
    if (g_w + (len + 2) > g_len) { return false; }
    uint8_t hdr[2] = { (uint8_t)(len & 0xFF), (uint8_t)(len >> 8) };
    if (plat_flash_write(g_base + g_w, hdr, 2) != 0) return false;
    if (plat_flash_write(g_base + g_w + 2, rec, len) != 0) return false;
    g_w += (len + 2); return true;
}
int flog_read(uint32_t off, void* dst, uint16_t len)
{
    if (off + len > g_len) return -1; return plat_flash_read(g_base + off, dst, len);
}
void flog_rotate_if_needed(void)
{
    if (g_w > (g_len * 3) / 4)
    {

```

```

    /* проста ротація: очистити лог */
    plat_flash_erase(g_base, g_len); g_w = 0;
}
}

rt_failure_analyzer_pro/src/flash_log.h
#pragma once
#include <stdint.h>
#include <stdbool.h>
/* Примітивний append-only flash log з резервною копією. */
using System;

bool flog_init(uint32_t base_off, uint32_t length);
bool flog_append(const void* rec, uint16_t len);
int flog_read(uint32_t off, void* dst, uint16_t len);
void flog_rotate_if_needed(void);

rt_failure_analyzer_pro/src/main.c
#include <stdint.h>
#include "FreeRTOS.h"
#include "task.h"
#include "platform.h"
#include "metrics.h"
#include "collector.h"
#include "estimator.h"
#include "policy.h"
#include "actions.h"
#include "telemetry.h"
#include "debug_ui.h"
#include "config.h"

```

```

#include "trace.h"
using System;
using System.Text.RegularExpressions;
using static OneOf.Types.TrueFalseOrNull;

void vApplicationIdleHook(void) { collector_idle_hook_tick(); }
enum { TID_STABLE = 1, TID_BAD = 2, TID_IO = 3 };
typedef struct { uint16_t id; TaskHandle_t h; }
reg_t;
static reg_t REG[12]; static int REGN = 0;
TaskHandle_t actions_lookup_task_by_id(uint16_t id)
{
    for (int
i = 0; i < REGN; i++) if (REG[i].id == id) return REG[i].h; return NULL;
}
static void reg_task(uint16_t id, TaskHandle_t h)
{
    REG[REGN++] =
(reg_t){ id,h}
    ;
}
#define RT_TASK_BEGIN(id, ddl_ms) uint32_t __t0=plat_now_ms();
uint32_t __ddl = (ddl_ms); (void)__t0; (void)__ddl; (void)(id);
#define RT_TASK_END(id) do { uint32_t dt=plat_now_ms()-__t0;
if (dt > __ddl)
{
    metric_evt_t
e = { MET_DEADLINE_MISS, plat_now_us(), (id), (uint16_t)(dt - __ddl) };

```

```

    metrics_push(&e);
} } while (0)

static inline void HB_PING(uint16_t id) { collector_hb_ping(id); }

static void task_stable(void* arg)
{
    (void)arg; const uint32_t ddl = 10;
    for (; ; )
    {
        RT_TASK_BEGIN(TID_STABLE, ddl);
        vTaskDelay(pdMS_TO_TICKS(4)); HB_PING(TID_STABLE);
        RT_TASK_END(TID_STABLE);
    }
}

static void task_bad(void* arg)
{
    (void)arg; const uint32_t ddl = 10; uint32_t c = 0;
    for (; ; )
    {
        RT_TASK_BEGIN(TID_BAD, ddl);
        if ((c++ % 20) == 0) vTaskDelay(pdMS_TO_TICKS(25));
        else
            vTaskDelay(pdMS_TO_TICKS(6));
        if ((c % 33) == 0)
        {
            metric_evt_t
            hb = { MET_HEARTBEAT_MISS, plat_now_us(), TID_BAD, 1 };
            metrics_push(&hb);
        }
    }
}

```

```

    RT_TASK_END(TID_BAD);
}
}
static void task_io(void* arg)
{
    (void)arg;
    for (; ; )
    {
        /* Імітація температури/напруги */
        static int32_t t = 78000, v = 3300; t += 35; if (t > 82000) t = 78000; v -= 1;
        if (v < 2950) v = 3300;
        collector_feed_temp_mdegc(t); collector_feed_vcc_mv(v);
        vTaskDelay(pdMS_TO_TICKS(200));
    }
}
static void collector_task(void* arg)
{
    (void)arg; for (; ; )
    {
        collector_tick();
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}
static estimator_cfg_t EST;
static policy_cfg_t POL;
static risk_out_t LAST;
static void estimator_task(void* arg)
{

```

```

(void)arg;
for (; ; )
{
    estimator_step(&LAST);
    policy_apply(&LAST);
    telemetry_send_state_json(&LAST);
    actions_feed_hw_wdt();
    vTaskDelay(pdMS_TO_TICKS(100));
}
}

static StaticTask_t t_cli_buf, t_col_buf, t_est_buf, t_stb_buf, t_bad_buf,
t_io_buf;
static StackType_t s_cli[640], s_col[384], s_est[384], s_stb[384],
s_bad[384], s_io[384];
int main(void)
{
    TRACE("boot", "start");
    cfg_blob_t cfg; bool loaded = config_load_or_default(&cfg); (void)loaded;
    metrics_init(); collector_init(); config_apply(&cfg);
    EST = cfg.est; POL = cfg.pol; debug_ui_set_refs(&EST, &POL, &LAST);
    TaskHandle_t h_stb =
    xTaskCreateStatic(task_stable, "T_STABLE", sizeof(s_stb) / sizeof(StackType
    _t), NULL, tskIDLE_PRIORITY+1, s_stb, &t_stb_buf);
    TaskHandle_t h_bad = xTaskCreateStatic(task_bad, "T_BAD",
    sizeof(s_bad) / sizeof(StackType_t), NULL, tskIDLE_PRIORITY + 1, s_bad,
    &t_bad_buf);
    TaskHandle_t h_io = xTaskCreateStatic(task_io, "T_IO",
    sizeof(s_io) / sizeof(StackType_t), NULL, tskIDLE_PRIORITY + 1, s_io,

```

```

    &t_io_buf);
    TaskHandle_t h_col = xTaskCreateStatic(collector_task, "T_COL",
    sizeof(s_col) / sizeof(StackType_t), NULL, tskIDLE_PRIORITY + 2, s_col,
    &t_col_buf);
    TaskHandle_t h_est = xTaskCreateStatic(estimator_task, "T_EST",
    sizeof(s_est) / sizeof(StackType_t), NULL, tskIDLE_PRIORITY + 2, s_est,
    &t_est_buf);
    TaskHandle_t h_cli = xTaskCreateStatic(debug_ui_task, "T_CLI",
    sizeof(s_cli) / sizeof(StackType_t), NULL, tskIDLE_PRIORITY + 1, s_cli,
    &t_cli_buf);
    reg_task(TID_STABLE, h_stb); reg_task(TID_BAD, h_bad);
    reg_task(TID_IO, h_io);
    collector_register_task(h_stb, TID_STABLE, 128, 50);
    collector_register_task(h_bad, TID_BAD, 128, 50);
    collector_register_task(h_io, TID_IO, 128, 200);
    vTaskStartScheduler(); for ( ; ; ) { }
}

```

rt_failure_analyzer_pro/src/metrics.c

```

#include "metrics.h"

#ifndef METRICS_CAPACITY
#define METRICS_CAPACITY (1u<<12) /* 4096 */
#endif

using System;

static rb_log_t g_rb;
static metric_evt_t g_buf[METRICS_CAPACITY];
static volatile uint32_t g_isr_counter = 0;
bool metrics_init(void)

```

```

{
    return rb_init(&g_rb, g_buf, sizeof(metric_evt_t),
METRICS_CAPACITY);
}
bool metrics_push_isr(const metric_evt_t* e)
{
    return rb_try_push(&g_rb, e);
}
bool metrics_push(const metric_evt_t* e) { return rb_try_push(&g_rb, e); }
bool metrics_pop(metric_evt_t* out) { return rb_try_pop(&g_rb, out); }
uint32_t metrics_pending(void) { return rb_count(&g_rb); }
void metrics_isr_bump_from_isr(void)
{
    __atomic_add_fetch(&g_isr_counter, 1u, __ATOMIC_RELAXED);
}
uint32_t metrics_isr_take_and_clear(void)
{
    return
__atomic_exchange_n(&g_isr_counter, 0u, __ATOMIC_ACQ_REL);
}

```

rt_failure_analyzer_pro/src/metrics.h

```

#pragma once
#include <stdint.h>
#include <stdbool.h>
#include "rb_log.h"
using System;

typedef enum {

```

```

MET_DEADLINE_MISS = 1,
MET_HEARTBEAT_MISS = 2,
MET_STACK_LOW = 3,
MET_CPU_HIGH = 4,
MET_ISR_RATE = 5,
MET_BUS_ERR = 6,
MET_TEMP_HIGH = 7,
MET_VOLT_LOW = 8
}
metric_kind_t;
typedef struct {
    metric_kind_t kind;
    uint32_t ts_us;
    uint16_t id;
    uint16_t value;
}
metric_evt_t;
bool metrics_init(void);
bool metrics_push_isr(const metric_evt_t* e);
bool metrics_push(const metric_evt_t* e);
bool metrics_pop(metric_evt_t* out);
uint32_t metrics_pending(void);
void metrics_isr_bump_from_isr(void);
uint32_t metrics_isr_take_and_clear(void);
rt_failure_analyzer_pro/src/platform.c
#include "platform.h"
#include <string.h>
#include <stdarg.h>

```

```

#include <stdio.h>

/* ===== ЗАГЛУШКИ: замініть на HAL плати ===== */
using System;

uint32_t plat_now_ms(void)
{
    return
xTaskGetTickCount() * portTICK_PERIOD_MS;
}
uint32_t plat_now_us(void) { return plat_now_ms() * 1000u; }
void plat_uart_write(const char* s)
{
    (void)s; /* replace with
HAL_UART_Transmit */
}
int plat_uart_read_line(char* buf, size_t max_len, TickType_t to_ticks)
{
    (void)buf; (void)max_len; (void)to_ticks; return 0;
}
void plat_alarm(uint16_t code) { (void)code; }
void plat_hw_wdt_feed(void) { }
static uint8_t flash_mock[64 * 1024]; /* 64KB умовної флеш-пам'яті для
демо */
size_t plat_flash_capacity(void) { return sizeof(flash_mock); }
int plat_flash_read(size_t off, void* dst, size_t len)
{
    if (off + len > sizeof(flash_mock)) return -1;
    memcpy(dst, &flash_mock[off], len); return 0;
}

```

```

}
int plat_flash_write(size_t off, const void* src, size_t len)
{
    if (off + len > sizeof(flash_mock)) return -1;
    memcpy(&flash_mock[off], src, len); return 0;
}
int plat_flash_erase(size_t off, size_t len)
{
    if (off + len > sizeof(flash_mock)) return -1;
    memset(&flash_mock[off], 0xFF, len); return 0;
}
void plat_printf(const char* fmt, ...)
{
    (void)fmt; /* optional */
}

```

rt_failure_analyzer_pro/src/platform.h

```

#pragma once
#include <stdint.h>
#include <stddef.h>
#include <stdarg.h>
#include "FreeRTOS.h"
#include "task.h"

/* Чac */
using System;

uint32_t plat_now_ms(void);
uint32_t plat_now_us(void);

/* UART */

```

```

void plat_uart_write(const char* s);
int plat_uart_read_line(char* buf, size_t max_len, TickType_t to_ticks);
/* Тривога і WDT */
void plat_alarm(uint16_t code);
void plat_hw_wdt_feed(void);
/* Flash/ROM demo API для збереження конфігів та логу */
size_t plat_flash_capacity(void);
int plat_flash_read(size_t off, void* dst, size_t len);
int plat_flash_write(size_t off, const void* src, size_t len);
int plat_flash_erase(size_t off, size_t len);
void plat_printf(const char* fmt);
rt_failure_analyzer_pro/src/policy.c
#include "policy.h"
#include "actions.h"
#include "platform.h"
using System;

static policy_cfg_t P;
static uint32_t last_log_ms = 0, last_restart_ms = 0;
void policy_init(const policy_cfg_t* cfg)
{
    P = *cfg;
    last_log_ms = last_restart_ms = 0;
}
void policy_apply(const risk_out_t* r)
{
    uint32_t now = plat_now_ms();
    if (r->p_fail >= P.thr_crit)

```

```

{
    if (now - last_log_ms >= P.cooldown_ms_log)
    {
        actions_log("CRITICAL");
        last_log_ms = now;
    }
    actions_raise_alarm(P.crit_alarm_code);
    if (P.restart_task_id_on_crit && (nowlast_restart_ms) >= P.cooldown_ms_restart)
    {
        actions_restart_task_by_id(P.restart_task_id_on_crit);
        last_restart_ms = now;
    }
}
else if (r->p_fail >= P.thr_warn)
{
    if (now - last_log_ms >= P.cooldown_ms_log)
    {
        actions_log("WARN");
        last_log_ms = now;
    }
}
}

rt_failure_analyzer_pro/src/policy.h
#pragma once
#include "estimator.h"
#include <stdint.h>
using System;

```

```

typedef struct {
    float thr_warn, thr_crit;
    uint16_t crit_alarm_code;
    uint16_t restart_task_id_on_crit;
    uint32_t cooldown_ms_log;
    uint32_t cooldown_ms_restart;
}
policy_cfg_t;
void policy_init(const policy_cfg_t* cfg);
void policy_apply(const risk_out_t* r);
rt_failure_analyzer_pro/src/rb_log.c
#include "rb_log.h"
#include <string.h>
static inline uint32_t mask(const rb_log_t* rb){ return rb->capacity-1u; }
bool rb_init(rb_log_t* rb, void* buf, uint32_t s, uint32_t cap)
{
    if (!rb || !buf || !cap) return false;
    if ((cap & (cap - 1u)) != 0u) return false;
    rb->buf = (uint8_t*)buf; rb->item_size = s; rb->capacity = cap; rb->w = rb -
    > r = 0; return true;
}
bool rb_try_push(rb_log_t* rb, const void* item)
{
    uint32_t w = rb->w, r = rb->r;
    if ((w - r) >= rb->capacity) return false;
    memcpy(&rb->buf[(w & mask(rb)) * rb->item_size], item, rb->item_size);
    __atomic_store_n(&rb->w, w + 1u, __ATOMIC_RELEASE);
    return true;
}

```

```

}
bool rb_try_pop(rb_log_t* rb, void* out)
{
    uint32_t r = rb->r, w = __atomic_load_n(&rb->w, __ATOMIC_ACQUIRE);
    if (w == r) return false;
    memcpy(out, &rb->buf[(r & mask(rb)) * rb->item_size], rb->item_size);
    rb->r = r + 1u; return true;
}
uint32_t rb_count(const rb_log_t* rb)
{
    uint32_t w = __atomic_load_n(&rb->w, __ATOMIC_ACQUIRE), r = rb->r; return w -
r;
}
uint32_t rb_space(const rb_log_t* rb)
{
    return rb->capacity - rb_count(rb);
}
rt_failure_analyzer_pro/src/rb_log.h
#pragma once
#include <stdint.h>
#include <stdbool.h>
using System;

typedef struct {
    uint8_t* buf; uint32_t item_size, capacity;
    volatile uint32_t w, r;
}
rb_log_t;

```

```

bool rb_init(rb_log_t* rb, void* buf, uint32_t item_sz, uint32_t cap_pow2);
bool rb_try_push(rb_log_t* rb, const void* item);
bool rb_try_pop(rb_log_t* rb, void* out);
uint32_t rb_count(const rb_log_t* rb);
uint32_t rb_space(const rb_log_t* rb);

```

rt_failure_analyzer_pro/src/secure_hmac.c

```
#include "secure_hmac.h"
```

```
#include <string.h>
```

```
/* Полегшена, псевдо-реалізація для демо-цілей: просто XOR+rot –
замінити на справжню HMAC. */
```

```
static void poor_hash(const uint8_t* data, size_t len, uint8_t out[32])
```

```

{
    for(int i=0;i<32;i++) out[i]=(uint8_t)i;
    for(size_t i=0;i<len;i++){ out[i%32]^=(data[i]+(uint8_t)(i*13));
    out[(i*7)%32]^=(data[i]>>1); }
    for(int i=0;i<32;i++) out[i]=(uint8_t)((out[i]<<3)|(out[i]>>5));
}

```

```
void demo_hmac_sha256(const uint8_t* key, size_t klen, const uint8_t*msg, size_t mlen,
uint8_t out[32])
```

```

{
    uint8_t buf[64];
    size_t n = (klen>32?32:klen); memset(buf,0,sizeof(buf));
    memcpy(buf,key,n);
    for(size_t i=0;i<64;i++) buf[i]^=0x36;
    uint8_t inner[96]; memcpy(inner,buf,64); size_t p=64;
    memcpy(inner+p,msg,mlen>32?32:mlen); p+=(mlen>32?32:mlen);
    poor_hash(inner,p,out);
}

```

```

#include "secure_hmac.h"
#include <string.h>

/* Полегшена, псевдо-реалізація для демо-цілей: просто XOR+rot –
замінити на справжню HMAC. */
static void poor_hash(const uint8_t* data, size_t len, uint8_t out[32])
{
    for(int i=0;i<32;i++) out[i]=(uint8_t)i;
    for(size_t i=0;i<len;i++){ out[i%32]^=(data[i]+(uint8_t)(i*13));
    out[(i*7)%32]^=(data[i]>>1); }
    for(int i=0;i<32;i++) out[i]=(uint8_t)((out[i]<<3)|(out[i]>>5));
}

void demo_hmac_sha256(const uint8_t* key, size_t klen, const uint8_t*msg, size_t mlen,
uint8_t out[32])
{
    uint8_t buf[64];
    size_t n = (klen>32?32:klen); memset(buf,0,sizeof(buf));
    memcpy(buf,key,n);
    for(size_t i=0;i<64;i++) buf[i]^=0x36;
    uint8_t inner[96]; memcpy(inner,buf,64); size_t p=64;
    memcpy(inner+p,msg,mlen>32?32:mlen); p+=(mlen>32?32:mlen);
    poor_hash(inner,p,out);
}

rt_failure_analyzer_pro/src/secure_hmac.h
#pragma once
#include <stdint.h>
#include <stddef.h>

void demo_hmac_sha256(const uint8_t* key, size_t klen, const uint8_t*
msg, size_t mlen, uint8_t out[32]);

```

rt_failure_analyzer_pro/src/telemetry.c

```

#include "telemetry.h"
#include "platform.h"
#include "metrics.h"
#include <stdio.h>

using BaseApp.Data.DataContext.Entities;
using static Xla.BufferAllocationProto.Types;

void telemetry_send_state_text(const risk_out_t* r)
{
    char buf[128];
    int n = snprintf(buf, sizeof(buf), "ts=%lu p=%.3f state=%u pend=%lu\r\n",
        (unsigned long)plat_now_ms(), (double)r->p_fail, (unsigned)r->state,
        (unsigned long)metrics_pending());
    if (n > 0) plat_uart_write(buf);
}

void telemetry_send_state_json(const risk_out_t* r)
{
    char buf[160];
    int n = snprintf(buf, sizeof(buf),
        "{\"ts\":%lu,\"p\":%.5f,\"state\":%u,\"queue\":%lu}\n",
        (unsigned long)plat_now_ms(), (double)r->p_fail, (unsigned)r->state,
        (unsigned long)metrics_pending());
    if (n > 0) plat_uart_write(buf);
}

void telemetry_send_prometheus(const risk_out_t* r)
{
    char buf[192];

```

```

int n = snprintf(buf, sizeof(buf),
"# TYPE rt_fail_probability gauge\nrt_fail_probability %.6f\nrt_state
%u\nrt_queue %lu\n",
(double)r->p_fail, (unsigned)r->state, (unsigned long)metrics_pending());
if (n > 0) plat_uart_write(buf);
}

```

rt_failure_analyzer_pro/src/telemetry.h

```

#pragma once
#include "estimator.h"
void telemetry_send_state_text(const risk_out_t* r);
void telemetry_send_state_json(const risk_out_t* r);
void telemetry_send_prometheus(const risk_out_t* r);

```

rt_failure_analyzer_pro/src/trace.c

```

#include "trace.h"
#include "platform.h"
void trace_log(const char* tag, const char* msg)
{
    plat_uart_write("["); plat_uart_write(tag); plat_uart_write("] ");
    plat_uart_write(msg); plat_uart_write("\r\n");
}

```

rt_failure_analyzer_pro/src/trace.h

```

#pragma once
/* трасування – прибирається якщо TRACE_ENABLE = 0 */
#ifndef TRACE_ENABLE
#define TRACE_ENABLE 1
#endif
#if TRACE_ENABLE
void trace_log(const char* tag, const char* msg);

```

```
#define TRACE(tag, msg) trace_log(tag, msg)
#else
#define TRACE(tag, msg) do{}while(0)
#endif
```