Міністерство освіти і науки України

Харківський національний університет імені В. Н. Каразіна

Кваліфікаційна наукова праця
на правах рукопису

ТЕЛЕЖЕНКО ДЕНИС ОЛЕКСАНДРОВИЧ

УДК 004.94

ДИСЕРТАЦІЯ

МЕТОДИ ТА МОДЕЛІ СИНТЕЗУ АРХІТЕКТУРИ ВІРТУАЛЬНИХ РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМ

Спеціальність 122 Комп'ютерні науки

Галузь знань 12 Інформаційні технології

Подається на здобуття ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ **Д. О. Тележенко**

Науковий керівник: **Толстолузька Олена Геннадіївна**
доктор технічних наук, старший науковий співробітник

Харків – 2025

АНОТАЦІЯ

Тележенко Д. О. Методи та моделі синтезу архітектури віртуальних розподілених комп'ютерних систем. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття ступеня доктора філософії за спеціальністю 122 Комп'ютерні науки (Галузь знань 12 Інформаційні технології). – Харківський національний університет імені В. Н. Каразіна, Міністерство освіти і науки України, Харків, 2025.

Дисертація присвячена вирішенню актуальної наукової проблеми – розробці методів і моделей синтезу архітектури віртуальних розподілених комп'ютерних систем (ВРКС) для забезпечення їх надійності, продуктивності та адаптивності до змінних умов роботи. Основна мета дослідження полягає у створенні нових алгоритмічних підходів, які дозволять прогнозувати та мінімізувати вплив збоїв у системі, забезпечувати оптимальний розподіл ресурсів та підтримувати стабільну роботу в умовах динамічних змін.

У дисертації виконано глибокий аналіз сучасних підходів до управління ВРКС, зокрема, методів балансування навантаження, динамічного масштабування, автоматичного резервування та моніторингу. Встановлено, що основними викликами у цій сфері є відсутність ефективних засобів для проактивного виявлення та попередження збоїв, а також недостатня інтеграція інструментів машинного навчання для адаптивного управління архітектурою системи.

У першому розділі описано основи аналізу існуючих підходів до синтезу архітектури ВРКС. Зокрема, розглянуто еволюцію технологій віртуалізації, яка сприяла створенню динамічних і масштабованих обчислювальних середовищ. Підкреслено, що перехід від монолітних до мікросервісних і гібридних архітектур дозволив покращити продуктивність, гнучкість і управління ресурсами.

Також у розділі описано класифікацію архітектур ВРКС, таких як монолітні, мікросервісні, централізовані та децентралізовані моделі. Детально розглянуто переваги та недоліки кожного типу: від простоти управління монолітних систем до складності управління, але високої адаптивності мікросервісних структур. Особливу увагу приділено впливу різних архітектур на масштабованість і оптимізацію системи.

Описано важливість впровадження методів оптимізації ресурсів у ВРКС, включаючи динамічне розміщення віртуальних машин, автоматизоване управління ресурсами та алгоритми планування завдань. Визначено, що застосування інтелектуальних підходів, таких як машинне навчання, дозволяє підвищити ефективність використання ресурсів і адаптацію системи до змін у навантаженні.

Далі у першому розділі розглянуті інструменти прогнозування та управління, які забезпечують проактивний підхід до підтримки стабільності ВРКС. Проведено аналіз існуючих алгоритмів машинного навчання для прогнозування навантаження та управління ресурсами. Визначено, що традиційні підходи не враховують динаміку сучасних систем і не забезпечують достатньої точності прогнозування.

У другому розділі описано процес розробки моделі прогнозування навантаження на сервери з використанням алгоритму LSTM. Основна увага приділяється створенню концептуальної моделі синтезу архітектури віртуальних розподілених систем, яка включає компоненти апаратного забезпечення, гіпервізора, віртуальних машин та модуля управління. Докладно розглянуто їх взаємодію та внесок у загальну продуктивність системи.

Далі у розділі досліджено методи паралельної обробки інформації, такі як метод суміщення незалежних операторів, метод конвеєрної обробки та метод декомпозиційної обробки. Ці підходи використовуються для оптимізації обчислювальних процесів у віртуальних машинах та забезпечення високої ефективності використання ресурсів. Показано, що поєднання

декількох методів дозволяє підвищити продуктивність системи у динамічних умовах.

Далі у другому розділі розглянуто процес інтеграції алгоритму LSTM у архітектуру віртуальних розподілених систем. Акцент зроблено на важливості якісної підготовки даних, зокрема їх очищення, нормалізації та структуризації для тренування моделей. Наведено підходи до збору даних у реальних умовах роботи серверів та їх перетворення для відповідності вимогам алгоритму LSTM. Окрему увагу приділено адаптації LSTM для роботи з часовими послідовностями, що характерні для прогнозування навантаження.

У третьому розділі представлено розробку і тестування модифікованого методу відновлення віртуальних розподілених систем (ВРС) після збоїв. Основна увага приділяється прогнозуванню навантаження та моделюванню поведінки системи під час аварійних ситуацій. Описано створення експериментального середовища за допомогою Docker-контейнерів, балансувальника навантаження Nginx, та інструментів моніторингу, таких як Prometheus і Node Exporter. Проведено моделювання збоїв, яке включало зупинку контейнерів та імітацію перевантаження вузлів для оцінки ефективності методу.

Далі було досліджено, як використання алгоритмів машинного навчання, зокрема LSTM, впливає на точність прогнозування збоїв вузлів. Виявлено, що LSTM забезпечує високу точність у передбаченні аварійних ситуацій, дозволяючи заздалегідь перенаправляти трафік для запобігання втратам даних і збоїв. Порівняльний аналіз показав, що підхід з використанням прогнозування дозволяє значно зменшити час реакції системи на збої та мінімізувати кількість втрачених запитів.

Далі у третьому розділі досліджено процес збору даних, який включав генерацію реалістичних сценаріїв навантаження за допомогою Apache Benchmark та моніторинг ключових показників роботи системи, таких як CPU, RAM, затримка відповіді та кількість оброблених запитів. Для навчання

моделі були використані нормалізовані часові ряди, які дозволили моделі LSTM ефективно ідентифікувати аномалії та тренди у поведінці системи.

У четвертому розділі описано процес розробки моделі прогнозування навантаження на сервери у віртуальних розподілених системах з використанням алгоритму LSTM. Основна увага приділяється методології збору, обробки та нормалізації даних, які є ключовими для забезпечення якості моделі. Для дослідження було використано відкриті набори даних з платформи Kaggle, які включають показники використання CPU, RAM, дискових і мережевих ресурсів, що дозволяє враховувати різноманітність параметрів і забезпечити комплексний аналіз.

Також було досліджене налаштування архітектури LSTM-моделі, яка включала декілька рівнів обробки: вхідний шар для нормалізації даних, LSTM-шари для аналізу часових залежностей, Dropout-шар для запобігання перенавчанню та Dense-шар для отримання остаточного прогнозу. Було визначено оптимальну кількість нейронів для кожного шару та параметри тренування, такі як навчальна швидкість, кількість епох та розмір батчу.

Далі у розділі розглянуто процес навчання та валідації моделі. Використання крос-валідації дозволило мінімізувати ризик перенавчання, а метрики оцінки, зокрема MSE, MAE і R^2, забезпечили кількісну оцінку точності моделі. Результати тестування підтвердили здатність моделі LSTM ефективно прогнозувати навантаження на сервери навіть за наявності аномалій у даних.

В наступній частині четвертого розділу було проаналізовано ефективність моделі у реальних умовах роботи серверів. Виконані експерименти показали, що впровадження LSTM дозволяє зменшити затримки, мінімізувати втрати запитів і підвищити загальну стабільність системи. Було створено гістограми розподілу затримок, які продемонстрували значне зниження частоти високих затримок у системі після впровадження прогнозування.

В останньому розділі обговорено практичну значущість отриманих результатів. Використання LSTM у прогнозуванні навантаження сприяє

оптимізації розподілу ресурсів, що є критично важливим для забезпечення продуктивності та надійності віртуальних розподілених систем. Отримані результати підтверджують ефективність підходу, забезпечуючи фундамент для подальшого розвитку моделей прогнозування у динамічних обчислювальних середовищах.

Сукупність результатів, викладених у дисертації, демонструє ефективність запропонованих методів і моделей для синтезу архітектури віртуальних розподілених систем. Запропонований прогнозно-адаптивний підхід із використанням алгоритму LSTM забезпечує високу точність прогнозування навантаження, оптимізацію розподілу ресурсів та мінімізацію наслідків збоїв у системах. Практична значущість отриманих результатів підтверджується їхньою здатністю підвищувати продуктивність і надійність віртуальних систем у динамічних умовах. Це відкриває нові перспективи для впровадження інтелектуальних рішень у хмарних середовищах, IoT та інших масштабованих обчислювальних платформах, роблячи внесок у розвиток сучасних інформаційних технологій.

Ключові слова: *віртуальні розподілені системи, синтез архітектури, прогнозування навантаження, алгоритм LSTM, машинне навчання, оптимізація ресурсів, інформаційні технології, модифікація, комп'ютерні системи, інтернет речей (IoT), паралельна обробка, моделювання збоїв, моніторинг систем, інтелектуальні обчислювальні платформи.*

ABSTRACT

Telezhenko D. Methods and models for the synthesis of architecture in virtual distributed computer systems. – Qualification scholarly paper: a manuscript.

The dissertation submitted for obtaining the Doctor of Philosophy degree in Information Technology: Speciality 122 Computer science. V. N Karazin Kharkiv National University, Ministry of Education and Science of Ukraine, Kharkiv, 2025.

The dissertation addresses a pressing scientific problem: the development of methods and models for the synthesis of architecture in virtual distributed computer systems (VDCS) to ensure their reliability, performance, and adaptability to changing operational conditions. The primary objective of the research is to create new algorithmic approaches that enable the prediction and mitigation of system failures, ensure optimal resource allocation, and maintain stable operation under dynamic changes.

The dissertation includes a comprehensive analysis of contemporary approaches to VDCS management, particularly methods for load balancing, dynamic scaling, automated redundancy, and monitoring. It has been established that the main challenges in this field include the lack of effective tools for proactive failure detection and prevention, as well as insufficient integration of machine learning tools for adaptive system architecture management.

The first chapter outlines the fundamentals of analyzing existing approaches to the synthesis of VDCS architectures. Specifically, it examines the evolution of virtualization technologies, which have facilitated the creation of dynamic and scalable computing environments. It emphasizes that the transition from monolithic to microservice and hybrid architectures has improved performance, flexibility, and resource management.

The text then delves into the classification of VDCS architectures, including monolithic, microservice, centralized, and decentralized models. The advantages and disadvantages of each type are discussed in detail, ranging from the simplicity

of managing monolithic systems to the complexity but high adaptability of microservice structures. Special attention is given to the impact of various architectures on scalability and system optimization.

The importance of implementing resource optimization methods in VDCS is described, including dynamic virtual machine placement, automated resource management, and task scheduling algorithms. It is established that the application of intelligent approaches, such as machine learning, enhances resource utilization efficiency and system adaptation to workload changes.

The chapter also explores prediction and management tools that support a proactive approach to maintaining VDCS stability. An analysis of existing machine learning algorithms for load prediction and resource management is presented. It is noted that traditional approaches fail to account for the dynamics of modern systems and do not provide sufficient prediction accuracy.

The second chapter describes the process of developing a server load forecasting model using the LSTM algorithm. The focus is on creating a conceptual model for synthesizing the architecture of virtual distributed systems, which includes components such as hardware, hypervisor, virtual machines, and a management module. The interaction between these components and their contributions to the overall system performance are examined in detail.

The chapter also explores methods of parallel data processing, such as the method of overlapping independent operators, pipeline processing, and decomposition-based processing. These approaches are employed to optimize computational processes within virtual machines and ensure high resource utilization efficiency. It is demonstrated that combining multiple methods can enhance system performance under dynamic conditions.

Furthermore, the second chapter investigates the integration of the LSTM algorithm into the architecture of virtual distributed systems. Emphasis is placed on the importance of high-quality data preparation, including data cleaning, normalization, and structuring, for training the models. Approaches to data collection under real-world server operating conditions and its transformation to

meet the requirements of the LSTM algorithm are presented. Special attention is given to adapting LSTM to handle time-series data, which is characteristic of load forecasting.

The third chapter presents the development and testing of a modified method for restoring virtual distributed systems (VDS) after failures. The primary focus is on load forecasting and modeling system behavior during failure scenarios. The creation of an experimental environment is described, utilizing Docker containers, the Nginx load balancer, and monitoring tools such as Prometheus and Node Exporter. Failure modeling included stopping containers and simulating node overloads to evaluate the method's effectiveness.

The study further investigates how machine learning algorithms, particularly LSTM, influence the accuracy of predicting node failures. It was found that LSTM provides high accuracy in forecasting failure scenarios, enabling proactive traffic redirection to prevent data loss and system downtime. Comparative analysis revealed that the predictive approach significantly reduces system response time to failures and minimizes the number of lost requests.

Additionally, the third chapter examines the data collection process, which involved generating realistic load scenarios using Apache Benchmark and monitoring key system performance metrics, such as CPU, RAM, response latency, and the number of processed requests. Normalized time-series data were used for model training, enabling the LSTM model to effectively identify anomalies and trends in system behavior.

The fourth chapter describes the development of a server load forecasting model for virtual distributed systems using the LSTM algorithm. The primary focus is on the methodology for data collection, processing, and normalization, which are crucial for ensuring model quality. Open datasets from the Kaggle platform were used for the study, including metrics on CPU, RAM, disk, and network resource usage. These datasets provided a diverse range of parameters and enabled comprehensive analysis.

The chapter then explores the configuration of the LSTM model architecture, which included multiple processing layers: an input layer for data normalization, LSTM layers for analyzing temporal dependencies, a Dropout layer to prevent overfitting, and a Dense layer for generating the final forecast. The optimal number of neurons for each layer and training parameters, such as learning rate, epochs, and batch size, were determined.

The process of training and validating the model is discussed further. Cross-validation was used to minimize the risk of overfitting, while evaluation metrics such as MSE, MAE, and R^2 provided a quantitative assessment of the model's accuracy. Testing results confirmed the LSTM model's ability to effectively predict server load even in the presence of data anomalies.

In the next section of the chapter, the model's efficiency in real-world server operation conditions was analyzed. Experiments demonstrated that implementing the LSTM model reduced latencies, minimized request losses, and improved the overall system stability. Histograms of latency distribution revealed a significant decrease in the frequency of high latencies in the system after incorporating load forecasting.

The final part of the chapter discusses the practical significance of the obtained results. The use of LSTM for load forecasting contributes to resource allocation optimization, which is critical for ensuring the performance and reliability of virtual distributed systems. The results validate the effectiveness of this approach, providing a foundation for further development of forecasting models in dynamic computing environments.

The body of results presented in the dissertation demonstrates the effectiveness of the proposed methods and models for synthesizing the architecture of virtual distributed systems. The proposed predictive-adaptive approach using the LSTM algorithm ensures high accuracy in load forecasting, resource allocation optimization, and minimization of failure impacts within the systems.

The practical significance of the obtained results is confirmed by their ability to enhance the performance and reliability of virtual systems under dynamic

conditions. This opens new prospects for implementing intelligent solutions in cloud environments, IoT, and other scalable computing platforms, contributing to the advancement of modern information technologies.

Keywords: *virtual distributed systems, architecture synthesis, load forecasting, LSTM algorithm, machine learning, resource optimization, information technologies, modification, computer systems, Internet of Things (IoT), parallel processing, failure modeling, system monitoring, intelligent computing platforms.*

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у наукових фахових виданнях, що входять до міжнародних наукометричних баз Scopus та Web of Science:

1. **Telezhenko D.,** Tolstoluzka O. Development and training of LSTM models for control of virtual distributed systems using Tensorflow and Keras. *Radioelectronic and Computer Systems*. 2024. Vol. 2024. Issue 3. P.27-37 DOI: 10.32620/reks.2024.3.02

Статті у наукових фахових виданнях України

2. **Telezhenko D.,** Tolstoluzka O. Conceptual model for synthesis of virtual distributed systems architecture. *Bulletin of V.N. Karazin Kharkiv National University, series "Mathematical modelling. Information technology. Automated control systems*. 2022. Vol. 55. P.49-55. DOI: 10.26565/2304-6201-2022-55

(*Особистий внесок: Особистий внесок здобувача: розробка основних ідей, створення концептуальної моделі, написання тексту та його переклад. Особистий внесок Olena Tolstoluzka: перевірка наукової достовірності отримуваних результатів, перевірка тексту роботи, редагування. Відповідні результатом є матеріалами публікації*).

3. Тележенко Д.О. Прогнозування та аналітика у віртуальних розподілених системах: Використання моделей машинного навчання та аналітичних інструментів для прогнозування поведінки системи. *Вісник Харківського національного університету імені В. Н. Каразіна «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління»*. 2023. Т. 60. С. 46-51. DOI: https://doi.org/10.26565/2304-6201-2023-60-05

(*Особистий внесок здобувача: розробка основних ідей, аналіз моделей машинного навчання, написання тексту та його переклад. Особистий*

внесок Олени Геннадіївни Толстолузької: перевірка наукової достовірності отримуваних результатів, перевірка тексту роботи)

Монографії:

4. Тележенко Д.О. Модифікація методу відновлення архітектури віртуальних комп'ютерних систем після збоїв. *Moderní aspekty vědy. Svazek XLVIII mezinárodní kolektivní monografie*. 2024. С. 274–283. DOI: 10.52058/48-2024

Наукові праці, які засвідчують апробацію матеріалів дисертації:

5. Telezhenko D. Model for predicting server load using LSTM. March 8, 2024; Zagreb, Croatia. III International Scientific and Theoretical Conference «Scientific method: reality and future trends of researching» March 8, 2024.
6. Telezhenko D.O., Tolstoluzka O.G., Setting the task of researching the architecture of virtual distributed systems. Proceedings of the 8rd International Conference "Computer Modeling in high-technology (CMHT-2022), pp. 179–181.
7. Telezhenko D., Tolstoluzka O. “Training of LSTM models for control of virtual distributed systems using Tensorflow and Keras”. Міжнародний науковий журнал «Грааль науки», 38 (квітень, 2024).
8. Telezhenko D., Tolstoluzka O. Method for modification of recovery architecture of virtual Computer systems after failures (CMHT, секція 2, 2024)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	–	application programming interface, інтерфейс програмування застосунків
AWS	–	Amazon Web Services, компнія надає платформу хмарних обчислень в оренду
CI/CD	–	Continuous Integration/Continuous Deployment, набір методик дозволяє частіше і надійніше розгортати зміни в програмному забезпеченні.
GCP	–	Google Cloud Platform, набір хмарних служб
IoT	–	The Internet of Things або Інтернет речей – система фізичних об’єктів («речей»), взаємопов’язаних між собою за допомогою вбудованих датчиків, програмного забезпечення та/або інших технологій
LSTM	–	Long Short-Term Memory, архітектура рекурентних нейронних мереж
ВРС	–	віртуальна розподілена система
ВРКС	–	віртуальна розподілена комп’ютерна система

ВСТУП

Обґрунтування вибору теми дослідження. Сучасні віртуальні машини все ще є монолітними, тобто вони складаються з тісно пов’язаних між собою сервісних компонентів, які таким чином копіюються на всі комп’ютери в організації. Проблема полягає в тому, що така архітектура є неефективною з точки зору масштабованості, використання ресурсів та керованості, оскільки при збільшенні кількості клієнтів виникає надмірне навантаження на обчислювальну інфраструктуру, а також ускладнюється адміністрування та забезпечення безперервності сервісів.

Тому для того, щоб вирішити цю проблему, необхідно розробити та впровадити нову системну архітектуру для мережевих обчислень на базі розподілених віртуальних машин. Ця архітектура потрібна для того, щоб зменшити потреби у ресурсах клієнта та розмір надійної обчислювальної бази, що встановлює фізичну ізоляцію між ними та службою віртуальних машин і створює єдину точку адміністрування. Така розподілена архітектура віртуальної машини може забезпечити суттєво кращу цілісність та керованість, ніж монолітна архітектура, так як вона повинна добре масштабуватися із збільшенням кількості клієнтів і не спричиняти великих накладних витрат.

Актуальність дослідження полягає у зростаючій потребі сучасних інформаційних технологій у вдосконаленні та оптимізації віртуальних розподілених систем (ВРС). ВРС є фундаментом для багатьох критично важливих застосунків, включаючи хмарні обчислення, великі дані, «інтернет речей» (IoT), які вимагають високої продуктивності, масштабованості, надійності та ефективного управління ресурсами. Водночас, із постійним розвитком та розширенням обсягу даних та комп'ютерних ресурсів, управління ресурсами та прогнозування навантаження на сервери стають все більш складними завданнями, що вимагають вдосконалених методів та інструментів.

Застосування алгоритмів машинного навчання, зокрема LSTM (Long Short-Term Memory), відкриває нові можливості для прогнозування навантаження на сервери та оптимізації використання ресурсів у ВРС. LSTM, з його здатністю ефективно обробляти та аналізувати часові послідовності даних, може значно підвищити точність прогнозування навантаження на сервери, що є критично важливим для планування ресурсів, забезпечення безперервності сервісу та зниження витрат.

Однак, незважаючи на значний потенціал, практичне застосування LSTM та інших алгоритмів машинного навчання у сфері ВРС зіштовхується з рядом викликів, включаючи високі вимоги до обчислювальних ресурсів, необхідність великих наборів даних для тренування моделей, а також складності інтеграції цих моделей в існуючі системи.

У зв'язку з цим, розробка та дослідження методів та моделей синтезу архітектури віртуальних розподілених комп’ютерних систем на основі LSTM, які б враховували специфіку та вимоги сучасних ВРС, стає актуальним і важливим завданням. Такі дослідження мають велике значення як для теоретичної інформатики, так і для практичної інженерії, оскільки вони сприяють підвищенню ефективності, надійності та гнучкості віртуальних розподілених

Актуальність проблеми розробки концепцій, принципів створення синтезу архітектури віртуальних розподілених систем (ВРС), методів розробки архітектури ВРС може бути висвітлена через призму діяльності та проектів визнаних на світовому рівні компаній. Розробка концепцій, принципів архітектурного синтезу та методів розробки архітектури ВРС є критично важливою з кількох причин, ілюстрованих проектами та ініціативами провідних технологічних компаній світу:

1. Amazon Web Services (AWS): AWS пропонує комплексну платформу хмарних обчислень, що демонструє необхідність надійної архітектури ВРС для підтримки масштабованих, надійних та безпечних застосунків у різних галузях. Інновації AWS у сфері архітектур без серверів, мікросервісів та

контейнерних сервісів (наприклад, AWS Lambda, Amazon ECS) підкреслюють важливість безперервної розробки принципів ВРС для задоволення еволюційних технологічних потреб.

2. Google Cloud Platform (GCP): Зобов'язання Google до відкритих технологій для управління контейнеризованими застосунками, як-от Kubernetes, висвітлює значення розробки нових концептуальних архітектур для ВРС, які можуть автоматизувати розгортання, масштабування та операції застосунків у кластерах хостів.

3. Microsoft Azure: Глобальна мережа дата-центрів та сервісів Azure підкреслює необхідність розширених архітектур ВРС, які можуть забезпечити високу доступність, відновлення після збоїв та безперервне масштабування. Проекти, як-от Azure IoT Hub, показують необхідність архітектур ВРС, здатних обробляти величезні обсяги даних та надавати реальні інсайти.

4. IBM Cloud: Сфокусування IBM на гібридних хмарних середовищах та сервісах аналітики, керованих AI, ілюструє складність інтеграції різноманітних систем у згуртовану архітектуру ВРС. Розробка принципів такої інтеграції є життєво важливою для ефективної роботи застосунків на рівні підприємства.

5. Відкриті проекти та ініціативи: Зростання відкритих проектів, як-от OpenStack, відкрите програмне забезпечення для хмарних обчислень, підкреслює роль спільноти у формуванні майбутньої архітектури ВРС. Це підкреслює колективні зусилля у розробці нових концепцій та методів для ефективніших, масштабованих та безпечних розподілених систем.

Ці приклади демонструють постійну потребу в інноваційних підходах до архітектури ВРС, що викликана викликами забезпечення масштабованих, надійних та безпечних обчислювальних ресурсів за запитом. Актуальність цієї проблеми ще більше підкреслюється швидким темпом цифрової трансформації, яка вимагає від бізнесу глобального розгортання складних застосунків, ефективного управління великими обсягами даних та забезпечення неперервної доступності сервісів.

Теоретичним дослідженням в галузі розробки принципів і методів побудови й застосування ВРС приділяється велика увага як вченими в Україні, так і за її межами. Серед вітчизняних дослідників необхідно відзначити: Жолткевича Г. М., Толстолузьку О.Г., Шевченко К. Л., Дученчука В., Савенко О.С., Бублик В., серед дослідників зарубіжжя: Sarker IH, Furhad MH, Nowrozy R., Clark, C., Fraser, K., Hand, S., Hansen, J. G., Roosta, Seyed H., Wood, T., Shenoy, P., Venkataramani, A., Smith, J., Nair, R. Роботи цих учених створили наукову основу для досліджень, теоретичні й методичні передумови подальшого пошуку шляхів підвищення ефективності розподілених комп'ютерних систем.

Викладене вище обумовлює актуальність рішення *науково-прикладного завдання* розробки ефективних методів і моделей синтезу архітектури віртуальних розподілених комп'ютерних систем (ВРКС), які базуються на використанні компонентів штучного інтелекту, для покращення прогнозування навантаження на сервери та оптимізації ресурсного управління.

Мета і задачі дослідження. Основною ***метою*** дисертаційної роботи є покращення прогнозування навантаження на сервери та оптимізація ресурсного управління за рахунок застосування розроблених ефективних методів і моделей для синтезу архітектури віртуальних розподілених комп'ютерних систем (ВРКС) з використанням компонентів штучного інтелекту.

Для досягнення даної мети як рішення поставленого наукового завдання в цілому, були сформульовані ряд ***задач***. До їхнього числа належать.

1. Теоретичний аналіз існуючих підходів до синтезу архітектури ВРКС, з акцентом на використанні алгоритмів машинного навчання для прогнозування навантаження та управління ресурсами.
2. Дослідження можливостей компонентів штучного інтелекту (зокрема, алгоритму LSTM) у контексті ВРКС, включаючи аналіз його потенціалу

для точного прогнозування навантаження на сервери та оптимізації ресурсного розподілу.

3. Розробка концептуальної моделі синтезу архітектури ВРКС на основі LSTM, яка буде включати методи та механізми для ефективного прогнозування та управління ресурсами.
4. Реалізація прототипу моделі з використанням сучасних інструментів програмування та аналізу даних, щоб демонструвати практичну придатність і ефективність запропонованого підходу.
5. Експериментальна перевірка моделі за допомогою реальних даних та сценаріїв використання, для оцінки її продуктивності, точності прогнозування та ефективності управління ресурсами.
6. Порівняльний аналіз отриманих результатів з існуючими методами та моделями, для визначення переваг та обмежень розробленої моделі.
7. Розробка рекомендацій для застосування розробленої моделі у практичних аспектах синтезу архітектури ВРКС, з урахуванням особливостей різних типів віртуальних середовищ.

Об'єкт дослідження – процес синтезу віртуальних розподілених комп’ютерних систем (ВРКС), які забезпечують високу доступність, масштабованість та розподіл обчислювальних ресурсів між різними застосунками та сервісами.

Предмет дослідження – методи та моделі синтезу архітектури ВРКС, які використовують алгоритми машинного навчання для оптимізації розподілу та управління ресурсами, прогнозування навантаження на сервери та забезпечення ефективної взаємодії між компонентами системи.

Методи дослідження. Ґрунтуються на застосуванні широкого спектру теоретичних та прикладних наукових підходів, що дозволяють комплексно підійти до вивчення та розробки моделей і методів синтезу архітектури віртуальних розподілених комп’ютерних систем з використанням алгоритмів машинного навчання. Основу теоретичних досліджень складають принципи системного аналізу, які дозволяють виявити основні властивості та зв’язки між

елементами ВРКС, та методи математичного моделювання, що забезпечують формалізацію процесів та явищ, що відбуваються в системі. Імітаційне моделювання використовується для дослідження динаміки системи та оцінки ефективності запропонованих рішень на основі комп'ютерних експериментів.

Специфіка дослідження передбачає застосування алгоритму LSTM, що є розширенням класичних рекурентних нейронних мереж і дозволяє ефективно працювати з даними, що мають довготривалі залежності. Для цього використовуються методи машинного навчання та глибокого навчання, які забезпечують побудову та тренування моделей на основі великих обсягів даних. Аналітичні методи, зокрема статистичний аналіз та методи оптимізації, застосовуються для обробки результатів експериментів, виявлення закономірностей та визначення оптимальних параметрів системи.

Важливе місце в методології дослідження займає використання теорії складних систем та апарату часових паралельних граф-схем для формалізації архітектури ВРКС та моделювання взаємодії її компонентів. Це дозволяє розробляти моделі, що враховують часові залежності та паралелізм виконання задач, що є ключовим для оптимізації розподілу ресурсів та підвищення ефективності системи.

Загалом, комбінація зазначених методів дослідження створює міцну наукову основу для розробки інноваційних методів та моделей у сфері синтезу архітектури ВРКС, що відкриває нові можливості для підвищення продуктивності, надійності та адаптивності віртуальних розподілених комп'ютерних систем.

Наукова новизна отриманих результатів полягає в наступному.

1. ***Вперше розроблено*** модель прогнозування навантаження на сервери у віртуальних розподілених системах, що відрізняється від існуючих можливістю прогнозування навантаження з високою точністю і адаптації до динамічних змін у системі.

2. ***Удосконалено*** метод оптимізації розподілу ресурсів у віртуальних розподілених системах, який відрізняється від існуючих використанням

методів машинного навчання, що забезпечує ефективне використання обчислювальних потужностей і зниження затримок у відповідях системи. Це включає алгоритми динамічного балансування навантаження та автоматичного масштабування.

3. ***Удосконалено*** метод відновлення архітектури віртуальних розподілених комп'ютерних систем після збоїв, який, на відміну від відомих, ґрунтується на прогнозуванні LSTM, дозволяючи системі антиципувати потенційні відмови та автоматично перерозподіляти ресурси для мінімізації впливу збоїв на продуктивність.

4. ***Отримала подальший розвиток*** методологія адаптації архітектури ВРКС до змінних умов експлуатації, яка відрізняється від відомих можливістю аналізу даних про ефективність системи, отриманих із LSTM моделей. Це забезпечує здатність системи до адаптивності в реальному часі.

Практична значимість отриманих результатів.

1. **Підвищення ефективності ВРКС.** Розроблені моделі та методи, які базуються на LSTM, дозволять значно покращити точність прогнозування навантаження на сервери, що призведе до більш ефективного розподілу ресурсів у віртуальних розподілених системах. Це, в свою чергу, забезпечить підвищення продуктивності системи та зниження операційних витрат.

2. **Оптимізація розподілу ресурсів.** Запропоновані методи оптимізації дозволять автоматично адаптувати розподіл ресурсів у відповідності до змінюваного навантаження, що забезпечить більш раціональне використання обчислювальних потужностей, зниження витрат на енергоспоживання та підвищення загальної доступності сервісів.

3. **Забезпечення високої доступності та надійності ВРКС.** Вдосконалення механізмів відновлення після збоїв на основі прогнозування LSTM сприятиме підвищенню стійкості ВРКС до відмов та забезпеченню неперервності бізнес-процесів.

4. **Адаптація ВРКС до динамічних умов експлуатації.** Розробка та впровадження методів адаптації архітектури ВРКС, що базуються на LSTM,

дозволить системам швидко реагувати на зміни у вимогах до обчислювальних ресурсів, що підвищить ефективність та гнучкість відповіді на потреби користувачів.

Отримані результати мають значний потенціал для практичного застосування в індустрії, зокрема в сферах, де використовуються великі обчислювальні потужності, таких як хмарні сервіси, великі дані, штучний інтелект, що робить дослідження вкрай актуальним та корисним для подальшого розвитку технологій ВРКС.

РОЗДІЛ 1.
АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО СИНТЕЗУ АРХІТЕКТУРИ ВІРТУАЛЬНИХ РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМ.

1.1 Еволюція віртуальних розподілених систем.

В останні десятиліття архітектури ВРКС стали фундаментальним елементом у розробці та управлінні сучасними обчислювальними системами, забезпечуючи надійність, масштабованість та ефективність обробки даних. Використання технологій віртуалізації та розподілу дозволяє створювати гнучкі та адаптивні обчислювальні середовища, які можуть швидко реагувати на змінні обчислювальні потреби та оптимізувати використання ресурсів. У рамках сучасної динаміки розвитку обчислювальних технологій, архітектури віртуальних розподілених комп'ютерних систем набувають особливої ваги. Вони лежать в основі розробки та управління сучасними обчислювальними системами, оскільки забезпечують критично важливі аспекти, такі як масштабованість, гнучкість, високу доступність, та ефективне управління ресурсами. Використання технологій віртуалізації дозволяє створювати адаптивні обчислювальні середовища, що можуть швидко реагувати на зміни в обчислювальних потребах, оптимізуючи при цьому використання апаратних ресурсів і покращуючи продуктивність системи в цілому.

Визначення ключових понять, таких як ВРКС, архітектура систем, машинне навчання та LSTM, є критично важливим для забезпечення чіткого розуміння предмета дослідження та контексту застосування розроблених методів та моделей [1]. Актуальність дослідження обумовлена стрімким розвитком технологій віртуалізації та зростаючими вимогами до ефективності, масштабованості та надійності ВРКС у контексті сучасних обчислювальних завдань. Розробка та оптимізація архітектури цих систем з використанням передових алгоритмів машинного навчання, як-от LSTM, відкриває нові можливості для покращення їх продуктивності та адаптації до динамічно змінюваних обчислювальних навантажень.

Основна мета цього аналізу полягає в оцінці ролі та значення архітектур ВРКС у сучасному світі обчислень, з особливим акцентом на вивченні потенціалу застосування алгоритмів машинного навчання, зокрема LSTM, для підвищення ефективності та оптимізації роботи цих систем. Аналіз спрямований на ідентифікацію викликів, з якими стикаються розробники та адміністратори ВРКС, та на визначення шляхів їх вирішення за допомогою інноваційних технологічних рішень.

У цьому контексті, основними об'єктами аналізу виступають різноманітні архітектурні моделі ВРКС, включно з монолітними, мікросервісними, централізованими, та децентралізованими підходами. Окрему увагу приділено методам управління ресурсами в межах цих систем, а також потенціалу використання LSTM для прогнозування навантаження на системи та оптимізації розподілу ресурсів [2]. Еволюція ВРС протягом останніх десятиліть є відображенням широкомасштабних змін у сфері інформаційних технологій, що охоплює перехід від простих віртуалізованих середовищ до складних і гнучких хмарних платформ. Цей процес не лише змінив підходи до розробки та управління обчислювальними ресурсами, а й створив нові можливості для оптимізації та масштабування обчислювальних процесів.

Перші кроки у віртуалізації були зроблені з метою ефективнішого використання фізичних обчислювальних ресурсів шляхом їх розділення на ізольовані віртуальні середовища, які можуть виконувати різні операційні системи та додатки на одному фізичному сервері. Це дозволило підвищити гнучкість управління ресурсами та забезпечити кращу доступність сервісів. Зі зростанням обсягів даних та обчислювальних потреб, концепція розподілених обчислень набула особливої актуальності. ВРС почали використовувати мережеві технології для об'єднання обчислювальних ресурсів, розташованих у різних географічних локаціях, в єдину обчислювальну систему. Це дало можливість розробникам та адміністраторам ефективно масштабувати системи, забезпечуючи високу продуктивність та надійність [3].

Завдяки прогресу в технологіях віртуалізації та розподілених обчислень, була створена основа для розвитку хмарних обчислень. Хмарні платформи відрізняються від традиційних ВРС тим, що вони пропонують як інфраструктуру як сервіс (IaaS), так і платформу як сервіс (PaaS), дозволяючи користувачам еластично управляти обчислювальними ресурсами через Інтернет. Це значно спростило розгортання та масштабування обчислювальних систем, роблячи їх доступними для ширшого кола користувачів.

Сучасні ВРС інтегруються з інноваційними технологіями, такими як контейнеризація, оркестрація контейнерів, безсерверні обчислення, та машинне навчання, для подальшої оптимізації управління ресурсами та автоматизації обчислювальних процесів. Використання LSTM алгоритмів у контексті ВРС відкриває нові можливості для прогнозування обчислювальних потреб та автоматичного масштабування систем, що забезпечує вищу ефективність та адаптивність до змінюваних умов [4].

Еволюція ВРС демонструє значний прогрес у способах розробки та управління обчислювальними системами, відкриваючи нові горизонти для інновацій та ефективності в сфері інформаційних технологій. Оптимізація цих систем за допомогою передових методів машинного навчання, зокрема LSTM, може значно підвищити їх продуктивність та адаптованість, відповідаючи сучасним та майбутнім викликам в області обчислень.

1.2. Класифікація архітектур віртуальних розподілених комп’ютерних систем.

ВРКС відіграють ключову роль у сучасному світі обчислень, надаючи гнучкі та ефективні рішення для розробки та управління програмними додатками. Однак, вибір відповідної архітектури ВРКС може суттєво вплинути на продуктивність, масштабованість та надійність систем.

Монолітні архітектури характеризуються єдиним, неділимим блоком програмного коду, де всі компоненти програми тісно інтегровані та працюють

в єдиному процесі [4]. Цей підхід спрощує розробку та розгортання програм, але може ускладнити масштабування та оновлення окремих частин системи.

Переваги монолітних архітектур включають простоту управління, оскільки всі компоненти розміщені в одному репозиторії, також до переваг таких архітектур можна віднести легкість розгортання, адже потрібно встановлювати лише один пакет. До недоліків монолітних архітектур можна віднести наступне:

1. Складність масштабування: монолітні системи часто виявляються важкими для масштабування, особливо коли потреба в ресурсах змінюється динамічно. Масштабування вимагає збільшення ресурсів для цілого застосунку, навіть якщо лише одна його частина потребує додаткових ресурсів.

2. Ускладнене оновлення та розгортання: внесення змін у монолітну архітектуру може бути складним та ризикованим процесом, оскільки невелика зміна в одному компоненті вимагає повного перерозгортання всієї системи.

3. Обмежена гнучкість: монолітні архітектури часто залежать від конкретних технологій та мов програмування, що обмежує можливість використання новітніх технологій та підходів у розробці.

Мікросервісні архітектури, навпаки, складаються з набору незалежних сервісів, що спілкуються між собою за допомогою визначеного API. Кожен мікросервіс відповідає за конкретну функцію та може бути розроблений, розгорнутий та масштабований незалежно від інших [5].

Переваги мікросервісних архітектур включають гнучкість масштабування та оновлення окремих сервісів без необхідності зупиняти всю систему [6]. Крім того, розподіленість сервісів сприяє кращому використанню ресурсів та розподілу навантаження. Основними недолікоми є складність управління, високі вимоги до мережі, складність розподілу даних.

1. Складність управління: управління великою кількістю мікросервісів може бути складним завданням, що вимагає використання складних

інструментів для моніторингу, оркестрації та забезпечення безперервності роботи сервісів.

2. Високі вимоги до мережі: мікросервісні архітектури залежать від інтенсивного мережевого спілкування між сервісами, що може призвести до затримок та вимагає високої пропускної спроможності мережі.

3. Складність розподілу даних: у мікросервісних архітектурах кожен сервіс може мати власну базу даних, що ускладнює управління даними та їх консистентність між сервісами.

Централізовані системи управляються з єдиного центру, що спрощує контроль та координацію ресурсів, але може створювати «вузькі місця» та знижувати загальну надійність системи через залежність від одного центру управління [7]. Недоліки централізованих систем:

1. Централізація управління може створювати «вузькі місця», які обмежують загальну продуктивність системи, а також створюють ризик відмови всієї системи у разі збою центрального вузла.

2. Масштабування централізованих систем часто вимагає значних зусиль та інвестицій в апаратну інфраструктуру, оскільки всі ресурси управляються через один центральний вузол.

Децентралізовані системи розподіляють управління та обчислювальні ресурси між багатьма вузлами, забезпечуючи вищу стійкість до відмов та гнучкість у масштабуванні [8]. Однак, це також збільшує складність управління та координації між вузлами. До недоліків децентралізованих систем можна віднести наступне:

1. У децентралізованих системах може виникати складність забезпечення ефективної координації та консистентності даних між розподіленими вузлами.
2. Децентралізація управління та даних може створювати додаткові виклики для забезпечення безпеки системи, оскільки кожен вузол стає потенційною точкою вторгнення.

Вплив структури управління на ефективність та надійність ВРКС полягає в здатності системи адаптуватися до змінних умов використання та забезпечувати високий рівень сервісу навіть у разі часткових відмов. Вибір між централізованими та децентралізованими архітектурами залежить від конкретних вимог до додатку, включаючи необхідний рівень масштабованості, надійності та гнучкості управління [9].

Вибір між різними архітектурами ВРКС вимагає ретельного аналізу потреб додатків, доступних ресурсів та цілей розробки, а також повинен базуватися на конкретних потребах додатку та обчислювальному контексті. Розуміння недоліків кожного підходу допоможе прийняти обґрунтоване рішення, забезпечивши оптимальне використання ресурсів та високу продуктивність системи.

1.3. Архітектурні моделі віртуальних розподілених комп’ютерних систем.

У сучасному світі інформаційних технологій, архітектурні моделі ВРКС еволюціонували, щоб задовольнити зростаючі вимоги до гнучкості, продуктивності та масштабованості. Серед цих моделей, гібридні архітектури та контейнеризація з оркестрацією стають все більш популярними завдяки своїй здатності забезпечувати оптимізовані рішення для різноманітних обчислювальних завдань.

Гібридні архітектурні моделі ВРКС представляють собою інноваційне рішення, яке об'єднує в собі елементи монолітної та мікросервісної структур. Ці моделі розроблені з метою досягнення балансу між контролем та гнучкістю, дозволяючи розробникам використовувати переваги обох підходів у залежності від конкретних потреб проекту.

Переваги гібридних моделей полягають у здатності до оптимального розподілу ресурсів, де критичні компоненти можуть бути ізольовані як мікросервіси для забезпечення кращої масштабованості та легшого оновлення, тоді як менш критичні частини можуть залишатися частиною монолітної бази для спрощення управління та розгортання [10].

Контейнеризація — це метод віртуалізації на рівні операційної системи, який дозволяє запускати додатки та їх залежності в ресурсо-ізольованих процесах, званих контейнерами. Вона використовується для створення легких, переносних та консистентних середовищ для розробки, розгортання та запуску додатків. На відміну від традиційної віртуалізації, що використовує гіпервізори для емуляції цілковитих віртуальних машин з власними операційними системами, контейнеризація дозволяє додаткам ділити одну операційну систему хоста. Це зменшує накладні витрати, оскільки кожен контейнер використовує лише необхідну частину системних ресурсів (наприклад, бібліотеки, системні виклики) [11].

Контейнеризація, особливо з використанням технологій, таких як Docker, революціонізувала спосіб розробки, розгортання та масштабування додатків у ВРКС. Контейнери дозволяють упаковувати додатки разом із їх залежностями в стандартизовані одиниці для розгортання, що забезпечує консистентність середовища від розробки до продакшну та спрощує процеси CI/CD.

Оркестрація відноситься до автоматизованого управління, координації та взаємодії між комп'ютерними системами, програмами та сервісами. Цей термін часто використовується для опису процесів, які забезпечують автоматичне розгортання, масштабування, керування життєвим циклом додатків та сервісів в контейнерах або віртуальних машинах на одному або кількох хостах.

Оркестрація контейнерів, за допомогою інструментів як Kubernetes, надає додатковий рівень управління, дозволяючи автоматично розподіляти, масштабувати та керувати життєвим циклом контейнерів у ВРКС [12]. Переваги включають в себе здатність до швидкого масштабування сервісів відповідно до змінюваних потреб, автоматичне відновлення від збоїв та ефективне використання ресурсів за допомогою розумного розподілу навантаження.

Гібридні моделі разом із контейнеризацією та оркестрацією відкривають нові можливості для розробників та адміністраторів ВРКС, пропонуючи гнучкість у виборі підходів до архітектури систем та ефективні інструменти для їх управління [13]. Ці технології створюють міцну основу для розвитку масштабованих, надійних та високопродуктивних обчислювальних систем, що можуть адаптуватися до змінюваних умов та вимог.

У таблиці 1.1 представлено порівняння різних архітектурних моделей віртуальних розподілених комп'ютерних систем, зокрема монолітних, мікросервісних, гібридних моделей, а також систем, що використовують контейнеризацію та оркестрацію.

Таблиця 1.1

Порівняльний аналіз архітектурних моделей віртуальних розподілених комп'ютерних систем

Характеристика	Монолітна архітектура	Мікросервісна архітектура	Гібридна модель	Контейнеризація та оркестрація
Складність управління	Низька	Висока	Середня	Висока
Масштабованість	Обмежена	Висока	Висока	Висока
Гнучкість розгортання	Обмежена	Висока	Висока	Висока
Надійність	Висока	Залежить від імплементації	Залежить від імплементації	Залежить від імплементації
Оновлення та розробка	Складні	Простіші	Залежить від компонентів	Простіші
Використання ресурсів	Може бути неефективним	Ефективне	Ефективне	Найбільш ефективне
Автономність компонентів	Ні	Так	Частково	Так
Технологічна залежність	Висока	Низька	Середня	Низька

Таблиця демонструє, що кожна архітектурна модель має свої переваги та недоліки, які роблять її більш або менш підходящою для конкретних видів додатків та бізнес-вимог. Вибір архітектури ВРКС повинен базуватися на комплексному аналізі потреб проекту, включаючи вимоги до масштабованості, гнучкості розгортання, надійності, а також рівня технічної підтримки та можливостей команди розробників [14].

1.4. Методи та моделі синтезу архітектури віртуальних розподілених комп'ютерних систем

Динамічна природа віртуальних розподілених систем означає, що ці системи постійно змінюються і адаптуються до різних чинників, таких як зміни в навантаженні, ресурси та завдання. Розгляд цієї динамічної природи важливий для оптимізації ресурсів у віртуальних розподілених системах. Ключові аспекти динамічної природи віртуальних розподілених систем:

- **Змінюючись навантаження та завдання:** віртуальні розподілені системи мають здатність адаптуватися до потреб користувачів та завдань, які змінюються. Навантаження може зростати та зменшуватися відповідно до попиту. Один із способів визначення зміни навантаження може бути виразом, де P_{actual} представляє поточний рівень навантаження, P_{target} - цільовий рівень навантаження, і $\Delta N = (N_{max} - N_{min}) \times (P_{actual} - P_{target})$. Де:

– ΔN – зміна кількості ресурсів,

– N_{max} і N_{min} – максимальна і мінімальна кількість ресурсів,

– P_{actual} – поточний рівень навантаження,

– P_{target} – цільовий рівень навантаження.

Формула ілюструє, як система може реагувати на зміни навантаження, збільшуючи або зменшуючи кількість ресурсів у відповідь на попит користувачів та завдань.

- **Динамічне масштабування ресурсів:** дозволяє системі автоматично збільшувати або зменшувати кількість ресурсів у відповідь на

зміни навантаження. Це допомагає забезпечити ефективне використання ресурсів. Концепцію динамічного масштабування ресурсів у віртуальних розподілених системах можна виразити наступною формулою: $\Delta N = (F_{actual} - F_{target}) \times R$ де ΔN представляє зміну кількості ресурсів, F_{actual} і F_{target} – фактичний та цільовий показники навантаження, а R - коефіцієнт, який визначає, на скільки одиниць змінюється кількість ресурсів відповідно до зміни навантаження.

- **Автоматизація управління:** ефективне управління динамічною природою системи вимагає автоматизованих механізмів та інтелектуальних алгоритмів.

Вираз автоматизації управління може бути узагальненим, оскільки конкретні алгоритми та механізми будуть залежати від конкретного контексту та завдань системи. Однак, загальна формула для визначення ефективності автоматизованого управління може виглядати наступним чином: $Efficiency = \frac{Output}{Input}$, де:

- $Efficiency$ – ефективність автоматизованого управління системою;
- $Output$ – вихідні результати, які були досягнуті завдяки автоматизованому управлінню (наприклад, оптимізація ресурсів, зниження часу відгуку системи тощо);
- $Input$ – вхідні дані та ресурси, які були витрачені на реалізацію автоматизованого управління.

За даною формулою можна виміряти ефективність системи управління, де більше значення відображає більшу ефективність використання ресурсів та досягнення поставлених цілей системи.

- **Моніторинг та аналіз даних:** збір та аналіз даних про динаміку системи допомагають виявляти патерни та прогнозувати зміни, що можуть виникнути в майбутньому.

Ці аспекти динамічної природи віртуальних розподілених систем досліджуються та розвиваються у відповідних дисциплінах, таких як обчислювальна наука, хмарні обчислення та управління ресурсами. Вони є ключовими для забезпечення продуктивності та надійності віртуальних розподілених систем [15]. Отримані результати опубліковані в статті Тележенко Д.О., та Толстолузької О.Г. «Методи оптимізації ресурсів в віртуальних розподілених системах», у «Віснику» Харківського національного університету імені Каразіна, серія “Математичне моделювання. Інформаційні технології. Автоматизовані системи управління. 2022. Вип. 55. С.150-153.

1.5. Методи оптимізації ресурсів в віртуальних розподілених системах.

Методи оптимізації ресурсів в віртуальних розподілених системах здійснюються на основі сучасних наукових досягнень і технологій з метою ефективного управління обмеженими ресурсами у віртуальних середовищах. Наукова новизна даної підтеми полягає в розвитку нових методологій, алгоритмів та інструментів, які дозволяють досягти оптимального використання обчислювальних, мережевих та сховища даних ресурсів у розподілених обчислювальних середовищах [16]. Ці методи допомагають максимізувати використання доступних ресурсів, зменшити витрати енергії, підвищити продуктивність і ефективність роботи системи.

Методи оптимізації ресурсів включають в себе ряд підходів та стратегій. До них входять:

- **Динамічне розміщення віртуальних машин.** Динамічне розміщення віртуальних машин (VM) дозволяє автоматично розміщувати VM на фізичних серверах з урахуванням їхнього поточного навантаження. Цей підхід допомагає забезпечити балансування навантаження та максимізувати використання ресурсів.
- **Віртуалізація ресурсів.** Віртуалізація дозволяє розділити фізичні ресурси, такі як CPU, пам'ять і сховища, на віртуальні екземпляри, які можуть

бути виділені окремим завданням або користувачам. Цей підхід покращує використання ресурсів і забезпечує ізоляцію між різними користувачами або завданнями.

- **Автоматизоване управління ресурсами.** Автоматизоване управління ресурсами використовує інтелектуальні алгоритми та системи моніторингу для динамічного реагування на зміни в навантаженні та ресурсах. Цей підхід допомагає оптимізувати використання ресурсів та забезпечує автоматичне управління процесами. Математично цю оптимізацію можна виразити за допомогою функції цілі та обмежень. Найпоширенішою формулою для цього є задача лінійного програмування (Linear Programming, LP), яка має наступний вигляд: припустимо, що ми маємо N ресурсів (наприклад, сервери, обчислювальні вузли тощо) і M видів завдань, які можна виконати на цих ресурсах.

– Позначимо через x_i кількість ресурсу i, яка буде виділена для виконання завдань.

– Позначимо через c_i вартість одиниці ресурсу i.

– Позначимо через a_{ij} кількість ресурсу i, необхідну для виконання одиниці завдання j.

– Позначимо через b_j кількість завдань j, які потрібно виконати.

Тоді функція цілі виглядає так:

$Minimize\ Z\ = \Sigma\ (c_i\ \times x_i)$, де $i\ =\ 1$ до N

Ця задача лінійного програмування допомагає оптимізувати виділення ресурсів для виконання завдань з мінімальними витратами. Автоматизовані системи використовують алгоритми оптимізації для розрахунку значень x_i, щоб досягти максимального значення функції цілі Z при врахуванні обмежень.

- **Оптимізація задач та планування.** Оптимізація задач та планування включає в себе використання алгоритмів оптимізації для вибору оптимального розташування завдань і ресурсів. Це допомагає зменшити час виконання завдань та максимізувати використання ресурсів.

Дослідження в області методів оптимізації ресурсів в віртуальних розподілених системах має великий потенціал для покращення продуктивності, забезпечення стабільності та зниження витрат у сучасних обчислювальних інфраструктурах [17]. Методи оптимізації ресурсів в віртуальних розподілених системах є ключовими для забезпечення ефективного використання обчислювальних ресурсів та досягнення високої продуктивності. Застосування цих методів дозволяє покращити ефективність системи та оптимізувати використання обмежених ресурсів. Ці методи не лише підвищують продуктивність та ресурсозбереження, але і забезпечують більш гнучке та ефективне управління віртуальними розподіленими системами. Вони стають ключовими для досягнення успіху та конкурентоспроможності в сучасному інформаційному світі, де обчислювальні ресурси стають дорогоцінними ресурсами. З підтримкою правильних методів оптимізації, організації можуть ефективно використовувати свої ресурси та досягати більших результатів.

1.6. Порівняльний аналіз моделей машинного навчання для задач прогнозування навантаження та управління ресурсами

Для забезпечення високої точності прогнозування навантаження та ефективного управління ресурсами у віртуальних розподілених системах важливим є вибір оптимальної моделі машинного навчання. У цьому контексті необхідно враховувати різні характеристики моделей, зокрема їхню здатність обробляти довготривалі залежності, точність прогнозування, адаптивність до змінних умов, обчислювальні витрати та стійкість до шуму.

Для аналізу та порівняння було обрано наступні моделі:

– LSTM (Long Short-Term Memory): нейронна мережа, яка спеціалізується на обробці послідовностей і довготривалих залежностей.

– GRU (Gated Recurrent Unit): спрощена версія LSTM, яка використовує менше параметрів, але зберігає здатність обробляти часові ряди.

– Transformer: сучасна архітектура з механізмом уваги, яка дозволяє ефективно працювати з великими наборами даних.

– ARIMA (AutoRegressive Integrated Moving Average): класична статистична модель для прогнозування часових рядів, що підходить для простих лінійних залежностей.

– Deep Q-Learning: метод навчання з підкріпленням, який використовується для прийняття рішень у динамічних середовищах шляхом побудови адаптивних стратегій.

Для оцінки ефективності моделей LSTM, GRU, Transformer, ARIMA і Deep Q-Learning було проведено їхній порівняльний аналіз. Таблиця 1.2 ілюструє ключові параметри кожної моделі, які є важливими для задач прогнозування та управління ресурсами у динамічних обчислювальних середовищах. Аналіз дозволяє обґрунтувати вибір найбільш підходящої моделі, яка не лише забезпечує високу точність прогнозування, але й відповідає практичним вимогам системи. Було обрано шість ключових характеристик, що визначають їхню ефективність у задачах прогнозування та управління ресурсами у віртуальних розподілених системах. **Точність прогнозування** оцінює здатність моделі надавати коректні передбачення на основі вхідних даних, що є критично важливим для забезпечення стабільності системи. **Складність моделі** відображає простоту її реалізації та інтеграції, що впливає на практичне впровадження. **Обчислювальні витрати** враховують ресурси, необхідні для навчання та роботи моделі, що важливо для систем із обмеженими ресурсами. **Масштабованість** оцінює здатність моделі ефективно працювати з великими наборами даних, що є ключовим у динамічних середовищах. **Швидкість прогнозу** відображає, наскільки швидко модель здатна робити передбачення, що є критичним для задач реального часу. **Стійкість до шуму** характеризує здатність моделі працювати з даними, які містять аномалії або похибки, забезпечуючи надійність прогнозів у реальних умовах. Ці характеристики дозволяють комплексно оцінити моделі та обрати оптимальну для поставлених задач.

Таблиця 1.2

Порівняльний аналіз моделей за їх характеристиками.

Модель	**Точність прогнозування**	**Складність моделі**	**Обчислювальні витрати**	**Масштабованість**	**Швидкість прогнозування**	**Стійкість до шуму**
LSTM	Висока	Складна	Високі	Висока	Середня	Висока
GRU	Середня	Середня	Низькі	Середня	Висока	Середня
Transformer	Висока	Дуже складна	Дуже високі	Висока	Середня	Середня
ARIMA	Низька	Проста	Низькі	Низька	Висока	Низька
Deep Q-Learning	Середня	Дуже складна	Дуже високі	Середня	Низька	Середня

Кожна модель оцінювалася за шістю ключовими характеристиками. Нижче наведено детальний аналіз кожної характеристики з висновками щодо придатності моделей:

1. Точність прогнозування.

a. LSTM забезпечує високу точність завдяки здатності враховувати довготривалі часові залежності та нелінійність даних. Це робить її ідеальною для складних сценаріїв, що вимагають врахування трендів і повторюваних патернів.

b. GRU демонструє середню точність через спрощену архітектуру, яка не завжди ефективна для довгих залежностей.

c. Transformer має високу точність через механізм уваги, який дозволяє працювати з великими обсягами даних.

d. ARIMA демонструє низьку точність, особливо в нелінійних залежностях.

e. Deep Q-Learning є менш точною для прогнозування, оскільки більше орієнтована на прийняття рішень, а не на аналіз часових рядів.

LSTM і Transformer є найкращими моделями за точністю. LSTM переважає завдяки здатності обробляти складні залежності з меншою обчислювальною вартістю.

2. Складність моделі.

a. LSTM має помірний рівень складності, який забезпечує баланс між точністю та реалізацією.

b. GRU є простішою, що спрощує її реалізацію та зменшує обчислювальні витрати.

c. Transformer має дуже складну архітектуру через механізм багатоголової уваги, що ускладнює її інтеграцію.

d. ARIMA є найпростішою, але обмежена лише простими сценаріями.

e. Deep Q-Learning має складну реалізацію через необхідність навчання стратегій на основі підкріплення.

GRU є найкращою за простотою реалізації, але LSTM забезпечує баланс між складністю та функціональністю.

3. Обчислювальні витрати.

a. LSTM потребує значних ресурсів для навчання, але забезпечує ефективну продуктивність.

b. GRU має найнижчі витрати серед нейронних мереж, завдяки спрощеній архітектурі.

c. Transformer має дуже високі витрати через складні обчислення у великих наборах даних.

d. ARIMA має найнижчі обчислювальні витрати, але її застосування обмежене.

e. Deep Q-Learning має дуже високі витрати через тривалий процес навчання та складність алгоритму.

GRU є найкращою за обчислювальними витратами, але LSTM забезпечує прийнятний баланс між витратами та продуктивністю.

4. Масштабованість.

a. LSTM забезпечує високу масштабованість, дозволяючи працювати з великими наборами даних.

b. GRU має середню масштабованість через обмеження архітектури.

c. Transformer демонструє відмінну масштабованість завдяки паралельній обробці, але потребує більше ресурсів.

d. ARIMA має низьку масштабованість, оскільки ефективна лише для малих наборів даних.

e. Deep Q-Learning має середню масштабованість, але складність реалізації може стати бар'єром.

LSTM і Transformer є найкращими за масштабованістю, але LSTM є більш збалансованим вибором.

5. Швидкість прогнозу.

a. GRU забезпечує найвищу швидкість прогнозу завдяки простішій архітектурі.

b. ARIMA також працює швидко, але це не компенсує її низьку точність.

c. LSTM має середню швидкість прогнозу, але достатню для більшості задач.

d. Transformer є відносно повільним через високі обчислювальні витрати.

e. Deep Q-Learning має найнижчу швидкість прогнозу через тривалий процес навчання.

GRU є найкращою за швидкістю, але LSTM пропонує збалансовану швидкість разом із високою точністю.

6. Стійкість до шуму.

a. LSTM демонструє високу стійкість до шуму, що дозволяє працювати з реальними, неідеальними даними.

b. GRU має середню стійкість через обмежену архітектуру.

c. Transformer потребує додаткової обробки для роботи з шумними даними.

d. ARIMA має найнижчу стійкість до шуму, що обмежує її застосування.

е. Deep Q-Learning має середню стійкість, оскільки більше орієнтована на рішення, ніж на дані.

LSTM є найкращою моделлю для роботи із шумними даними.

На основі проведеного аналізу, LSTM є найкращою моделлю для задач прогнозування навантаження та управління ресурсами у віртуальних розподілених системах, демонструючи переваги за більшістю ключових характеристик, таких як точність прогнозування, адаптивність, масштабованість та стійкість до шуму. Хоча моделі GRU і Transformer також показують сильні сторони, такі як простота реалізації та висока масштабованість відповідно, вони поступаються LSTM у здатності забезпечувати точні та надійні прогнози у складних сценаріях. Модель ARIMA є доцільною для простих задач, але обмежена у динамічних середовищах, тоді як Deep Q-Learning більше підходить для адаптивного управління, але не для аналізу часових рядів. Таким чином, LSTM забезпечує оптимальний баланс між продуктивністю та обчислювальною ефективністю, що робить її найкращим вибором для вирішення поставлених задач.

1.7. Постановка задачі дослідження сінтезу архітетури віртуальних розподілених систем

Важливим аспектом розподілених віртуальних систем є також як продуктивність так і керованість. Важливим аспектом розподілених віртуальних систем є як продуктивність, так і керованість. Вхідними даними для дослідження є метрики продуктивності системи, що включають поточне завантаження серверів, середній час відгуку, обсяги оброблюваних даних, а також характеристики апаратної інфраструктури, такі як параметри серверів, мережеві параметри та сховища даних. Крім цього, використовуються лог-файли, що містять історичні дані про навантаження, відмови та операції адміністрування, аналіз користувацької активності, який відображає типи завдань і їх періодичність, а також моделі даних і сценарії використання, які описують різноманітні типи робочих навантажень, зокрема пікові. Додатково залучаються результати імітаційного моделювання для перевірки

запропонованих рішень у різних сценаріях. У ході дослідження розглядаються проблеми, пов’язані з підвищенням точності прогнозування навантаження на сервери, забезпеченням ефективного розподілу ресурсів, розробкою архітектури, здатної адаптуватися до змінних умов експлуатації, та підвищенням стійкості до збоїв із можливістю швидкого відновлення працездатності системи.

Так як некоректно налаштована система, може буде занадто складна у керованості, що згодом призведе до проблем також і у масштабуванні, що у свою чергу погано впливає на продуктивність таких систем. Для того, щоб мінімізувати такі потенційні проблеми необхідно розробити архітектуру для системи, що у свою чергу буде робити систему «гнучкою» для змін.

На рисунку 1.1 представлено зміст дисертаційного дослідження, де зображення представляє схему взаємозв'язків між завданнями дослідження, розділами дисертації та науковою новизною отриманих результатів.

Завдання 1, теоретичний аналіз існуючих підходів до синтезу архітектури ВРС, було реалізовано у вступі та першому розділі. Завдання 2, дослідження можливостей використання алгоритмів машинного навчання для прогнозування навантаження та управління ресурсами, було реалізовано у першому та другому розділах дисертації. Завдання 3, розробка концептуальної моделі синтезу архітектури ВРКС з акцентом на LSTM, реалізовано в другому та третьому розділах. Завдання 4, реалізація прототипу розробленої моделі, та завдання 5, експериментальна перевірка моделі, були реалізовані у третьому та четвертому розділах. Завдання 6, порівняльний аналіз результатів із існуючими підходами, реалізовано у розділах три, чотри та висновках.

Завдання 7, розробка рекомендацій для практичного використання, було реалізовано у четвертому розділі та висновках. І також на рисунку 1.1 показан зв’язок між розділами та науковою новизною, що ці розділи розкривають.

Рис. 1.1. Зміст дисертаційного дослідження.

Висновки до розділу 1.

Перший розділ присвячений аналізу існуючих підходів до синтезу архітектури віртуальних розподілених комп'ютерних систем, що стають все більш важливими в сучасному технологічному середовищі. У цьому розділі детально розглянуто еволюцію віртуальних розподілених систем, що дозволяє зрозуміти, як технології віртуалізації сприяли створенню гнучких і адаптивних обчислювальних середовищ, здатних до динамічної зміни та оптимізації використання ресурсів. Аналіз еволюції показує, що перехід від монолітних архітектур до розподілених структур, включаючи мікросервісні та гібридні моделі, забезпечив кращу масштабованість, продуктивність та гнучкість управління.

Значну увагу було приділено класифікації архітектур ВРКС, таких як монолітні, мікросервісні, централізовані та децентралізовані моделі. Кожен підхід має свої переваги та недоліки, що залежать від потреб конкретного застосування та контексту використання. Наприклад, монолітні архітектури, хоча й простіші в управлінні, часто виявляються менш гнучкими у масштабуванні, тоді як мікросервісні архітектури дозволяють забезпечувати масштабування та оновлення окремих компонентів, але вимагають складнішого управління і більш високих ресурсів.

Крім того, було розглянуто методи оптимізації ресурсів у ВРКС, такі як динамічне розміщення віртуальних машин, віртуалізація ресурсів, автоматизоване управління ресурсами, та оптимізація задач і планування. Ці методи є ключовими для ефективного використання обчислювальних ресурсів і досягнення високої продуктивності систем. Аналіз методів показав, що застосування інтелектуальних алгоритмів і автоматизації значно підвищує ефективність розподілу ресурсів і дозволяє системам швидше адаптуватися до змін у навантаженні.

У підсумку, Розділ 1 створює науково обґрунтовану основу для подальшої розробки моделей синтезу архітектури ВРКС. Розглянуто

підходи, класифікації архітектур і методи оптимізації показують значний потенціал для вдосконалення продуктивності, надійності та масштабованості систем, що є критично важливими для задоволення вимог сучасних обчислювальних завдань.

РОЗДІЛ 2.
РОЗРОБКА МОДЕЛІ ПРОГНОЗУВАННЯ НАВАНТАЖЕННЯ НА СЕРВЕРИ З ВИКОРИСТАННЯМ АЛГОРИТМУ LSTM.

2.1. Схема моделі синтезу архітектури віртуальних розподілених систем

Віртуальні розподілені системи є ключовим компонентом сучасних технологій обчислення. Вони дозволяють ізолювати різні обчислювальні процеси один від одного, що забезпечує підвищення ефективності використання ресурсів і гарантує безпеку. Цей розділ пропонує схему моделі синтезу архітектури ВРС, що базується на новому підході до віртуалізації. Важливо розробити схему моделі синтезу архітектури віртуальних розподілених систем, що включає в себе ключові компоненти, їх взаємодію, стратегії управління, технології, методи оптимізації та тестування. Крім того, модель повинна бути масштабованою та гнучкою, адаптованою до змінних умов та нових технологічних трендів [19]. Очікується, що робота допоможе інженерам та науковцям краще зрозуміти, як можна ефективно проектувати та розробляти віртуальні розподілені системи, враховуючи різні фактори, що впливають на їх продуктивність, надійність та ефективність.

Схема моделі синтезу архітектури ВРС включає в себе чотири основні компоненти: апаратне забезпечення, гіпервізор, віртуальні машини та модуль управління (рис. 1). Ці компоненти взаємодіють між собою задля забезпечення роботи всієї системи.

1. *Компоненти апаратного забезпечення*: ці компоненти є основою віртуальних розподілених систем. Параметри, які описують фізичні ресурси, такі як CPU (ядра, частота), RAM (об'єм), сховище (тип, об'єм, швидкість) та мережеві інтерфейси. Вони включають центральні процесори, оперативну пам'ять, пристрої зберігання

даних, мережеві інтерфейси та інші ресурси. Для віртуальних розподілених систем важливо мати гнучке та ефективне апаратне забезпечення, яке може бути динамічно перерозподілене між віртуальними машинами відповідно до їх потреб. Без надійного, масштабованого та високопродуктивного апаратного забезпечення ефективна робота ВРС не можлива.

2. *Гіпервізор*: програмне забезпечення, яке забезпечує віртуалізацію апаратного забезпечення [20]. Гіпервізор розподіляє ресурси апаратного забезпечення між віртуальними машинами, забезпечує ізоляцію та незалежність віртуальних машин та контролює їх виконання. Гіпервізор є критично важливим для ВРС, оскільки він дозволяє використовувати одну фізичну машину для запуску багатьох незалежних віртуальних машин.
3. *Віртуальні машини:* «віртуальні комп'ютери», які працюють на одному фізичному сервері, управляються гіпервізором. Кожна віртуальна машина має власну операційну систему та набір програм. В контексті даної теми, віртуальні машини є важливими, оскільки вони дозволяють виконувати декілька незалежних завдань на одному сервері, оптимізуючи використання ресурсів та підвищуючи гнучкість системи [21]. Окрім цього, застосування методів паралельної обробки інформації при виконанні кожного окремого завдання на віртуальній машині дозволяє значно зменшити загальний час виконання завдань. На даний час в літературі описано п’ять методів паралельної обробки інформації:
   - метод суміщення незалежних операторів (СО);
   - метод конвеєрної обробки (КО);
   - метод декомпозиційної обробки (ДО);
   - кодово-матричний метод (КММ);
   - метод суміші алгоритмів (СА).

Сутність методу суміщення незалежних операторів полягає в одночасному початку виконання у кожний з дискретних моментів часу деякої кількості операторів алгоритму, для яких виконуються такі умови:
а) ці оператори не пов’язані інформаційними та/або керуючими зв’язками;
б) для кожного з таких операторів до моменту часу, що розглядається, є значення відповідних операндів.

Будемо використовувати надалі таке визначення методу суміщення операцій – це метод паралельної обробки, для якого:

- об’єктами операційного рівня є бітові коди чисел або частини кодів (до окремого біта), що розглядаються як неподільні цілі;
- об’єктами алгоритмічного рівня є фрагменти алгоритмів чи алгоритми, які розглядаються як неподільні цілі;
- операторами операційного рівня є загальноприйняті операції/функці обробки даних (при роботі з кодами чисел) та операції булевої алгебри (при роботі з однобітовими даними);
- операторами алгоритмічного рівня є операції над графами або часовими паралельними граф-схемами;
- часові відносини між (хоча б деякими) операторами задовольняють вимогу наявності принаймні однієї пари операторів Pi∈P та Pj∈P алгоритму P з непорожнім перетином їх інтервалів активності IAi і IAj

$$IA_i \cap IA_j \neq O \quad \text{для } P_i, P_j \in P, \tag{2.1}$$

виконання яких починається одночасно, тобто для яких t_i^H t_j^H;

- характер часових відносин між інтервалами активності I(SDk) реалізації алгоритму в різних наборах вхідних даних SDk∈SD визначається наступним співвідношенням:

$$I(SDk) \in I(SDl) \neq O \tag{2.2}$$

хоча б для деякої пари номерів k , lj вхідних наборів даних алгоритму де ki, l ∈ 1,2,...,sd; sd = | SD |, тобто має місце перетин часових інтервалів реалізацій алгоритму для різних наборів даних.

Метод конвеєрної обробки (КО) – це метод паралельної обробки даних, в якому в якості об'єктів виступають:

- алгоритми виконання операторів – загальноприйнятих операцій/функцій відомих мов програмування;
- об'єкти алгоритмічного рівня – фрагменти алгоритмів або алгоритми, що розглядаються як неподільне ціле [22].

Сутність методу полягає у виконанні для алгоритму наступних перетворень:

а) поділу часового алгоритму виконання операцій/функцій (на операційному рівні) або часового алгоритму розв'язання задачі (на алгоритмічному рівні) на «конвеєрні» фрагменти (F) рівної часової глибини ТК (такт конвеєра), такі, що кожен попередній фрагмент формує вхідні дані для суміжного наступного фрагмента, а параметр tн(Fj) початку кожного наступного фрагмента Fj визначається параметром tк(Fi) кінця попереднього фрагмента

Fi(i,j ∈ NF;NF= 0,1,...,nf 1; де nf=| NF| – кількість конвеєрних фрагментів або глибина конвеєра);

б) послідовної реалізації у часі фрагментів F0, F1, ..., Fnf-1 – у разі одиничної потужності (sd = 1) множини SD різних вхідних наборів даних алгоритму;

в) суміщення (при sd>1) інтервалів виконання «різнотипних» конвеєрних фрагментів алгоритму, що належать до різних наборів вхідних даних;

г) номери ρ поєднаних фрагментів Fρ, що відповідають вхідним наборам даних з номерами δ (δ=1,2,...,nf-1, nf, nf+1,…), задовольняють (у режимі роботи конвеєра) наступному співвідношенню ρ + δmod (nf +1) = nf [23].

Метод конвеєрної обробки характеризується введенням наборів вхідних даних з інтервалом дискретності ТК та виведенням результатів виконання алгоритму (для множини SD вхідних наборів даних) з інтервалом ТК.

Сутність методу паралельної суміші алгоритмів (СА) полягає у створенні на операційному рівні суміші завдань, використанні такої суміші для досягнення 100% завантаження обладнання та забезпечення за рахунок цього потенційно можливого підвищення ефективності реалізації аналізованої сукупності алгоритмів [24]. Суміш алгоритмів формально розглядається і виконується як єдине завдання. Суміш завдань використовується в тих випадках, коли окремо виконувані завдання, що використовують усі або частину розглянутих вище методів паралельної обробки, не можуть забезпечити 100% завантаження обладнання та досягнення потенційного значення продуктивності.

Наразі тільки метод суміщення незалежних операторів використовується в усіх сучасних процесорах. Метод конвеєрної обробки на апаратному рівні застосовується в скалярних та суперскалярних процесорах. Декомпозиційний метод, також на апаратному рівні, застосовується в процесорах *VLIW* з паралелізмом на рівні команд, *Very Long Instruction Word*. Між іншим, дослідження, проведені на факультеті комп'ютерних наук Харківського національного університету в межах наукового напрямку «На виконання завдань перспективного плану розвитку наукового напряму «Технічні науки» Харківського національного університету імені В.Н. Каразіна» показали, що застосування раціональної сукупності методів паралельної обробки інформації (мультипаралельна обробка інформації) дозволяють підвищити ефективність обчислювальних систем різних класів. В таблиці 2.1 наведено доцільний склад різних методів паралельної обробки для конкретних комбінацій вимог і обмежень користувачів.

Таблиця 2.1

Порівняльний аналіз методів паралельної обробки для конкретних комбінацій вимог і обмежень користувачів

Вимоги/ обмеження		Методи паралельної обробки			
		Поєднання незалежних операцій (c=1)	Метод конвеєрної обробки (k=1)	Метод декомпозиційної обробки (d=1)	Кодово-матричний метод (q=1)
Час виконання алгоритму t_0 (алгоритмів t_i)	Z=1	+	-	-	+
	Z>1	+	+	+	+
Період оновлення вхідних даних T_{BX}	Z=1	-	+	+	+
	Z>1	-	+	+	+
Спільні вимоги $(t_0 / t_v) \& T_{BX}$	Z=1	+	+	+	+
	Z>1	+	+	+	+
Вимоги і обмеження $(t_0 / t_v) \& Q$	Z=1	+	-	-	+
	Z>1	+	+	+	+
Вимоги і обмеження $T_{BX} \& Q$	Z=1	-	+	+	+
	Z>1	-	+	+	+
Вимоги і обмеження $(t_0/t_i)\&T_{BX}\&Q$	Z=1	+	+	+	+
	Z>1	+	+	+	+

4. *Модуль управління* є відповідальним за моніторинг, управління та оптимізацію роботи ВРС. Це включає моніторинг використання ресурсів, управління розподілом ресурсів, забезпечення безпеки системи та реагування на зміни в навантаженні або стані системи. Модуль управління критично важливий для синтезу архітектури віртуальних розподілених систем, оскільки він забезпечує їх стабільну та ефективну роботу. Урахування особливостей організації обчислювального процесу в окремих вузлах розподіленої системи сприяє підвищенню продуктивності всієї

віртуальної обчислювальної системи [25]. Наприклад, паралельне виконання послідовних ниток операцій в скалярних та суперскалярних процесорах здійснюється за допомогою апаратно реалізованого конвеєру (або декількох одночасно працюючих конвеєрів для суперскалярних процесорів). Во *VLIW* процесорах паралельно виконуються команди, які входять до довгого командного слова, тобто команди для яких на даний момент часу готові всі вхідні дані і є вільний ресурс для реалізації.

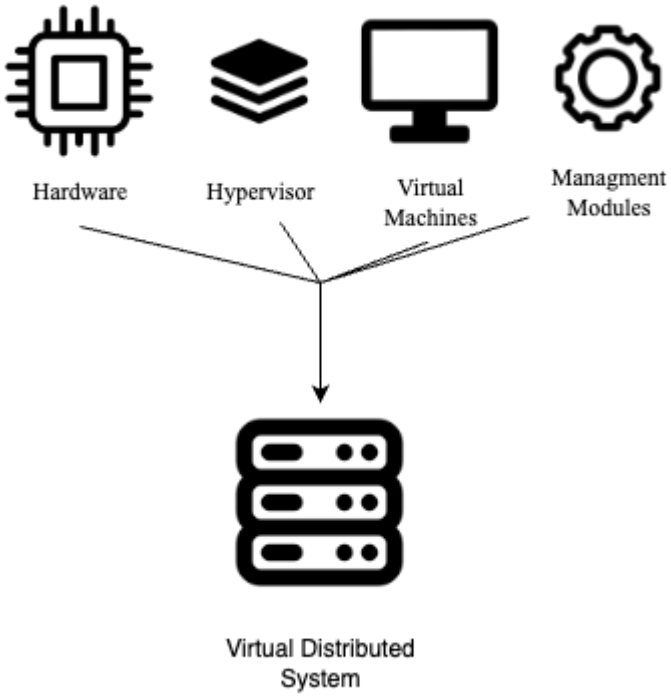

Рис. 2.1 Схема моделі синтезу архітектури віртуальних розподілених систем.

Концептуальна модель є важливою частиною процесу проектування віртуальних розподілених систем. Вона визначає загальну структуру системи, відображаючи взаємодії між її різними компонентами, а також принципи їхньої роботи [26].

Концептуальна модель слугує основою для детального проектування системи. Вона допомагає розробникам краще розуміти задачі, які система повинна вирішувати, та контекст, в якому вона буде використовуватися

[27]. Це, у свою чергу, дозволяє визначити вимоги до системи та вибрати найбільш підходящі технології для її реалізації. У концептуальній моделі віртуальних розподілених систем можуть враховуватися такі аспекти:

- Функціональність: які задачі система повинна вирішувати? Які сервіси вона повинна надавати своїм користувачам?
- Взаємодія між компонентами: як компоненти системи взаємодіють один з одним? Як вони співпрацюють, щоб забезпечити потрібну функціональність?
- Сценарії використання: у яких умовах та сценаріях система буде використовуватися? Які вимоги до надійності, продуктивності, масштабованості та безпеки вона повинна відповідати?
- Обмін даними і мережеві взаємодії: як будуть передаватися дані між різними частинами системи? Які протоколи та технології будуть використовуватися для цього?

Концептуальна модель створюється на початкових етапах розробки системи та може оновлюватися протягом всього життєвого циклу проекту для врахування нових вимог або змін у технологічному середовищі.

Синтез архітектури є процесом, у ході якого концептуальна модель перетворюється в детальний план або «архітектуру» віртуальної розподіленої системи. Це включає вибір конкретних технологій, проектування структури системи, визначення інтерфейсів та протоколів для взаємодії між компонентами, а також розробку стратегій для забезпечення надійності, продуктивності, масштабованості та безпеки системи.

Синтез архітектури може бути поділений на кілька етапів, включаючи:

- **Вибір технологій:** на основі вимог до системи та концептуальної моделі вибираються конкретні технології для реалізації віртуальних машин, гіпервізорів, систем управління та інших компонентів системи.

- **Проектування структури:** на цьому етапі визначається детальна структура системи, включаючи структуру мережі, розміщення віртуальних машин, протоколи для взаємодії між компонентами та ін.

- **Розробка стратегій забезпечення якості обслуговування**: на основі вимог до надійності, продуктивності, масштабованості та безпеки розробляються стратегії для забезпечення цих характеристик, такі як стратегії резервного копіювання, балансування навантаження, шифрування даних тощо [28].

- **Тестування та валідація:** після проектування архітектури вона тестується і валідується для переконання, що вона відповідає вимогам до системи.

- **Оптимізація:** на основі результатів тестування та валідації вносяться потрібні корективи та оптимізуються різні аспекти архітектури.

Отримані результати опубліковані в статті Тележенко Д.О., та Толстолузької О.Г. «Conceptual model for synthesis of virtual distributed systems architecture» у «Віснику» Харківського національного університету імені Каразіна, серія Математичне моделювання. Інформаційні технології. Автоматизовані системи управління, вип. 55, стор.49–55, 2022

2.2. Розробка концептуальної моделі прогнозування навантаження на сервери з використанням алгоритму LSTM

На рисунку 2.2 зображена архітектура прогнозування навантаження на сервери у віртуальних розподілених системах. Розглянемо призначення різних компонентів архітектури на сервери у віртуальних розподілених системах:

– Символ 1 (рис. 2.2), містить вхідний набір даних, який містить інформацію про навантаження на сервери віртуальних розподілених систем (час, дані «заліза», навантаження, температуру і т.д.).

o Символ 2 (рис. 2.2), процес нормалізує або масштабує дані таким чином, щоб всі значення знаходились у певному діапазоні, зазвичай від 0 до

1. Це полегшує тренування моделі, оскільки масштабовані дані зазвичай сприяють кращій збіжності.

– Символ 3 (рис. 2.2), SMOTE (Synthetic Minority Over-sampling Technique): цей метод використовується для боротьби з дисбалансом класів у наборі даних. Він створює синтетичні приклади міноритарного класу для вирівнювання кількості прикладів між класами, що допомагає уникнути упередженості моделі до домінуючого класу.

– Символ 4 (рис. 2.2), перетворення, де дані перетворюються (reshaping) для подачі у LSTM шари. LSTM мережі потребують вхідних даних у формі тривимірного масиву (зазвичай [приклади, часові кроки, ознаки].), тому дані мають бути переформатовані відповідно.

– Символ 5 (рис. 2.2), набір для навчання (Train Dataset) – це частина даних, яка використовується для тренування моделі. Вона проходить через LSTM шари для вивчення залежностей в часі та ідентифікації шаблонів.

– Символ 6 (рис. 2.2), позначає LSTM модель.

– Символ 7 (рис. 2.2), позначає ворота забування (forgate gate) в LSTM, які вирішують, яку інформацію з попереднього стану слід відкинути.

– Символ 8 (рис. 2.2), позначає ворота модуляції вхідних (input modularity gate) даних в LSTM, які регулюють внесок нової інформації в стан комірки

– Символ 9 (рис. 2.2), позначає ворота вводу (input gate) в LSTM, які контролюють, яка нова інформація додається до стану комірки.

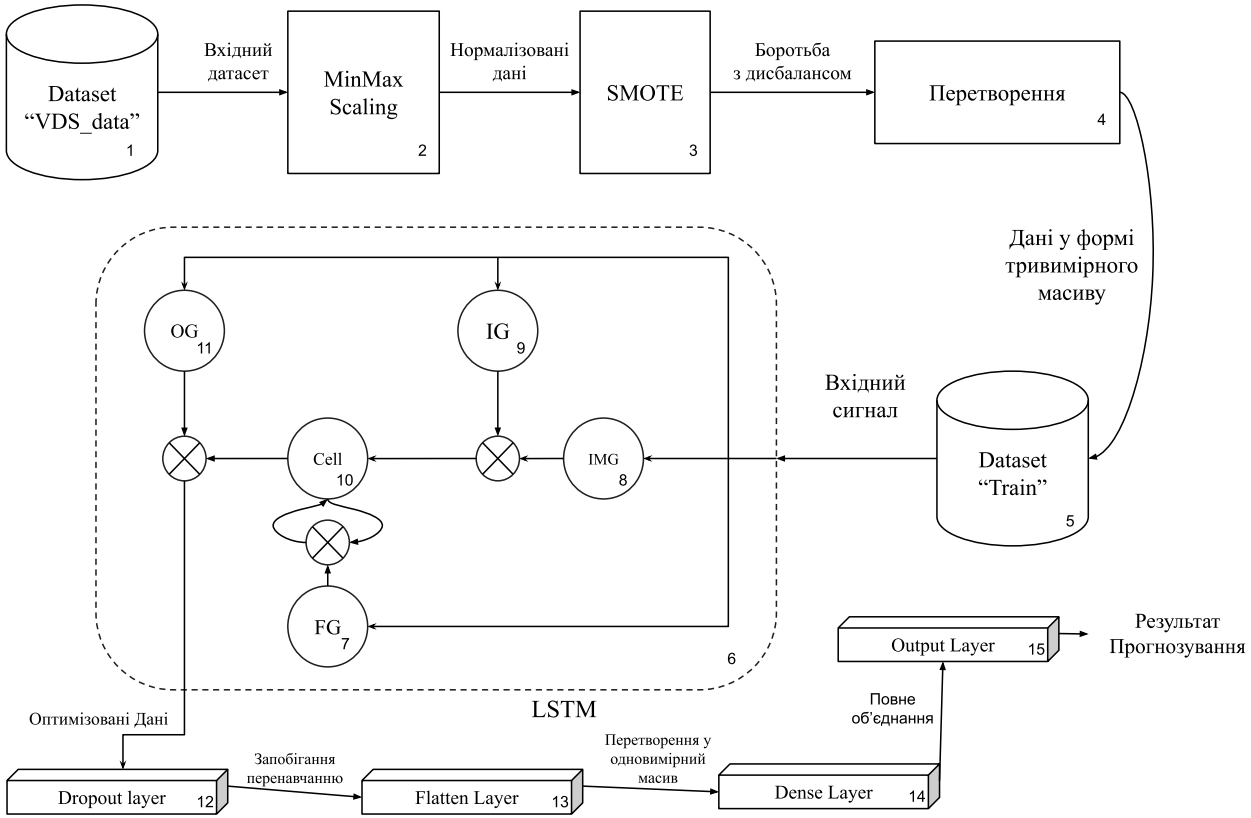

Рис. 2.2. Концептуальна модель прогнозування навантаження на сервери у ВРС.

– Символ 10 (рис. 2.2), позначає стан комірки в LSTM, який зберігає інформацію про весь час тренування.

– Символ 11 (рис. 2.2), позначає ворота виводу (output gate) в LSTM, які визначають, яка інформація передається до виходу з стану комірки.

– Символ 12 (рис. 2.2), позначає шар (Dropout Layer), який запобігає перенавчанню шляхом випадкового відключення деяких нейронів під час тренування.

– Символ 13 (рис. 2.2), позначає шар (Flattern Layer), який перетворює вихідні дані з LSTM шарів у одновимірний масив для подальшої обробки.

– Символ 14 (рис. 2.2), позначає шар нейронний (Dense Layer) для подальшої обробки інформації перед передачею до вихідного шару.

– Символ 15 (рис. 2.2), позначає шар для виведення кінцевого прогнозу або результату класифікації.

Отримані результати опубліковані в статті Тележенко Д. О., та Толстолузької О. Г. «Development and training of LSTM models for control of virtual distributed systems using Tensorflow and Keras» Vol 2024, No 3 (2024): Radioelectronic and Computer Systems.

2.3. Проектування архітектури: інтеграція LSTM у архітектуру ВРС.

Ефективність машинного навчання, зокрема LSTM (Long Short-Term Memory) моделей, значною мірою залежить від якості та репрезентативності вхідних даних. У контексті оптимізації архітектури віртуальних розподілених систем, підготовка даних вимагає ретельного аналізу та обробки даних для забезпечення їх відповідності до задачі тренування моделі. Цей процес включає збір, очищення, анотацію, трансформацію та нормалізацію даних [29].

Архітектура мереж LSTM еволюціонувала для вирішення конкретних завдань у розподілених системах. Наприклад, гібридні моделі, що поєднують LSTM з іншими нейронними мережевими структурами, такими як трансформаторні моделі, були розроблені для підвищення точності прогнозування та ефективності обчислень. Ці вдосконалені архітектури дозволяють багатозадачне навчання в режимі реального часу та особливо ефективні в середовищах із високою різноманітністю та складністю.

Традиційна хмарна обробка даних стає неефективною через високу затримку, насиченість пропускної здатності та проблеми конфіденційності. Граничні обчислення, де дані обробляються ближче до їх джерела, вирішують ці проблеми, зменшуючи затримку, зберігаючи конфіденційність і зберігаючи пропускну здатність. У цьому документі розглядаються різні пристрої IoT і фреймворки AI, такі як TensorFlow Lite, OpenEI та Core ML, які підтримують завдання ML на крайніх пристроях [30]. У дослідженні також розглядаються проблеми з розгортанням ML на пристроях з обмеженими ресурсами, включаючи шифрування даних для забезпечення конфіденційності, ефективного управління ресурсами та обмеження споживання енергії. Дослідження завершується визначенням ключових напрямків дослідження для оптимізації машинного навчання в периферійних обчислювальних середовищах.

Гібридна модель поєднує мережі LSTM і Deep Q-learning (DQL) для оптимізації планування роботи в хмарних мережах. Модель використовує LSTM для прогнозування робочого навантаження та DQL для прийняття рішень, тим самим покращуючи використання ресурсів і показники виконання завдань. LSTM фіксує тимчасові залежності, а DQL оптимізує рішення щодо планування на основі системи винагород. Модель, навчена на історичних даних, значно покращує виконання завдань і ефективність використання ресурсів. Порівняльний аналіз показав, що він перевершує автономний LSTM і традиційні алгоритми, підкреслюючи потенціал

поєднання методів прогнозування та навчання з підкріпленням для складних завдань управління ресурсами. Результати також підкреслюють надійність моделі в обробці різноманітних і мінливих робочих навантажень, що робить її адаптованою до різних операційних сценаріїв [31]. Цей гібридний підхід застосовується до віртуальних розподілених систем і забезпечує основу для інтеграції передових методів машинного навчання для покращення керування ВРС. У майбутньому ми плануємо вивчити додаткову інтеграцію машинного навчання для подальшого вдосконалення.

1. Підготовка Даних: початковий етап тренування передбачає збір і підготовку вхідних даних, які відображають стан та конфігурацію ВРС. Дані охоплюють широкий спектр параметрів, включаючи характеристики апаратного забезпечення, налаштування гіпервізора, конфігурації віртуальних машин та стратегії управління. Ключовим аспектом є нормалізація даних для забезпечення їх однорідності, що сприяє підвищенню ефективності тренування.

2. Конфігурація Моделі: архітектура LSTM моделі визначається з урахуванням специфіки задачі оптимізації. Модель містить кілька шарів LSTM для обробки послідовних даних, а також шари Dropout для регуляризації, які запобігають перенавчанню моделі. Кінцевий шар Dense використовується для генерації прогнозованих конфігурацій ВРС.

3. Оптимізація та Тренування: тренування моделі здійснюється за допомогою алгоритму оптимізації Adam, що забезпечує адаптивне коригування швидкості навчання. Вибір гіперпараметрів, таких як швидкість навчання, розмір пакету та кількість епох, заснований на експериментальному аналізі та перевірці моделі на валідаційному наборі даних. Процес тренування включає в себе ранню зупинку, щоб припинити навчання при відсутності поліпшення на валідаційному наборі протягом заданої кількості епох.

4. Валідація та Оцінка Ефективності: після тренування моделі проводиться її оцінка на тестовому наборі даних для визначення точності прогнозування. Використовуються метрики, такі як середня квадратична помилка (MSE), для оцінки різниці між прогнозованими та фактичними конфігураціями ВРС. Аналіз результатів дозволяє виявити потенційні області для подальшої оптимізації моделі [32].

2.3.1 Підготовка даних для тренування LSTM моделі

Підготовка даних для тренування LSTM моделі у задачі оптимізації архітектури ВРС: початковий етап тренування передбачає збір і підготовку вхідних даних, які відображають стан та конфігурацію ВРС. Дані охоплюють широкий спектр параметрів, включаючи характеристики апаратного забезпечення, налаштування гіпервізора, конфігурації віртуальних машин та стратегії управління. Ключовим аспектом є нормалізація даних для забезпечення їх однорідності, що сприяє підвищенню ефективності тренування.

1. *Збір даних.* Збір даних для тренування LSTM моделі в задачі оптимізації ВРС включає агрегацію інформації про апаратне забезпечення, конфігурації гіпервізорів, параметри віртуальних машин, та стратегії управління системами. Важливо зібрати детальні та різноманітні дані, що відображають реальні умови експлуатації ВРС, для забезпечення високої узагальнюючої здатності моделі.

2. *Очищення та анотація даних.* Очищення даних включає видалення некоректних, відсутніх або аномальних значень. Анотація даних полягає у визначенні та маркуванні вхідних даних, що сприяє їх ефективній обробці та аналізі під час тренування моделі. Для задачі оптимізації ВРС, анотація може включати класифікацію параметрів за типами ресурсів або за функціональним призначенням компонентів системи.

3. *Трансформація даних.* Трансформація даних включає зміну формату даних для їх подальшої обробки. Наприклад, часові ряди логів системи

можуть бути перетворені в структуровані вектори значень, які відображають зміну показників системи протягом часу. Також, категоріальні дані, як-от типи гіпервізорів, можуть бути перетворені за допомогою one-hot encoding для подальшого використання в моделі.

4. *Нормалізація даних.* Нормалізація даних є критично важливим кроком, який забезпечує однаковий масштаб для всіх числових параметрів. Це сприяє підвищенню стабільності та ефективності тренування моделі, оскільки градієнтний спуск та інші алгоритми оптимізації працюють ефективніше з нормалізованими даними. Методи нормалізації можуть включати мінімаксну нормалізацію або стандартизацію за Z-оцінкою.

Підготовка даних є фундаментальним етапом у процесі тренування LSTM моделі для оптимізації архітектури ВРС. Ретельний підхід до збору, очищення, анотації, трансформації, та нормалізації даних забезпечує високу якість тренувального датасету, що в свою чергу сприяє розробці ефективних та точних моделей машинного навчання [33].

2.3.2. Конфігурація LSTM моделі

Конфігурація моделі LSTM відіграє вирішальну роль у процесі машинного навчання, спрямованого на вирішення задачі оптимізації архітектури ВРС. Правильно сконфігурована модель дозволяє ефективно враховувати часові залежності та складні взаємозв'язки між параметрами системи, що є критично важливим для точного прогнозування оптимальних конфігурацій.

1. *Вибір архітектури.* Архітектура LSTM моделі повинна бути спроєктована з урахуванням специфіки задачі та характеристик вхідних даних. Важливими аспектами є кількість шарів LSTM, кількість нейронів у кожному шарі, та використання додаткових шарів, таких як Dense (щільність) або Dropout (регуляризація).

2. *Конфігурація шарів LSTM.* Шари LSTM формують основу моделі, забезпечуючи здатність до збереження та врахування довготривалих

залежностей у даних. Кількість шарів та нейронів у кожному шарі впливає на здатність моделі вловлювати складність даних. Занадто мала кількість нейронів може призвести до недонавчання, тоді як надмірно велика кількість може спричинити перенавчання.

3. *Використання шару Dropout.* Шари Dropout застосовуються для регуляризації моделі шляхом випадкового ігнорування деякої частини нейронів під час тренування. Це допомагає зменшити ризик перенавчання, сприяючи підвищенню узагальнюючої здатності моделі [34].

4. *Конфігурація вихідного шару.* Вихідний шар моделі, зазвичай щільний (Dense), проєктується для генерації кінцевого прогнозу. Кількість нейронів у вихідному шарі відповідає кількості прогнозованих параметрів системи. Активаційна функція цього шару вибирається в залежності від специфіки задачі прогнозування.

5. *Налаштування гіперпараметрів.* Вибір гіперпараметрів, включаючи швидкість навчання, розмір пакета (batch size) та кількість епох тренування, є критично важливим для успішності тренування. Експериментальний підхід та використання методів оптимізації гіперпараметрів, таких як Grid Search або Random Search, дозволяють знайти оптимальні налаштування для конкретної задачі.

Конфігурація LSTM моделі є ключовим етапом у процесі розробки системи для оптимізації архітектури ВРС. Вибір архітектури, налаштування шарів та гіперпараметрів повинен враховувати специфіку задачі та характеристики вхідних даних [35]. Ефективна конфігурація моделі сприяє точному прогнозуванню оптимальних конфігурацій системи, забезпечуючи високу продуктивність та ефективне використання ресурсів.

2.3.3. Етапи оптимізації та тренування

Розробка LSTM моделі вимагає детального розуміння процесу конфігурації моделі, а також ефективного планування етапів оптимізації та

тренування. Цей процес включає реалізацію стратегій тренування, які забезпечують високу точність та ефективність моделі.

1. *Оптимізаційний алгоритм.* Вибір оптимізаційного алгоритму, такого як Adam, RMSprop або SGD (Stochastic Gradient Descent), є вирішальним для налаштування процесу тренування. Adam часто вибирають за його ефективність у різноманітних задачах та здатність адаптуватися до змінюваної топології простору помилок.

2. *Тренування та валідація.* Тренування моделі включає в себе ітеративний процес коригування ваг нейронів на основі зворотного поширення помилки, з метою мінімізації функції втрат. Використання валідаційного набору даних дозволяє оцінити узагальнюючу здатність моделі та адаптувати гіперпараметри для запобігання перенавчанню. Техніка ранньої зупинки може бути застосована для припинення тренування, коли помітно, що модель починає перенавчатися.

Ефективне тренування LSTM моделі для оптимізації архітектури ВРС вимагає глибокого розуміння процесу конфігурації моделі та оптимізації [36]. Вибір архітектури моделі, налаштування шарів та гіперпараметрів, вибір оптимізаційного алгоритму, та ефективне планування тренування та валідації є ключовими факторами, що впливають на успіх задачі машинного навчання.

2.3.4. Валідація та оцінка ефективності LSTM моделі

Валідація та оцінка ефективності моделі є вирішальними етапами в процесі машинного навчання, особливо при розробці LSTM моделей для оптимізації архітектури віртуальних розподілених систем. Ці процеси дозволяють визначити, наскільки добре модель узагальнює навчену інформацію на нових, невідомих даних, і є ключовими для гарантування її практичної застосовності.

1. *Валідація моделі.* Валідація моделі включає використання окремого валідаційного набору даних, який не брав участі в тренуванні, для перевірки

її узагальнюючої здатності. Цей етап дозволяє виявити потенційне перенавчання моделі та адаптувати гіперпараметри до її мінімізації. Часто використовується метод крос-валідації, зокрема k-fold крос-валідація, яка забезпечує більш надійну оцінку узагальнюючої здатності моделі шляхом поділу всього набору даних на k взаємовиключних підмножин і проведення тренування та тестування моделі k разів.

2. *Оцінка ефективності* моделі вимагає аналізу її продуктивності на тестовому наборі даних, який використовується для остаточної оцінки моделі.
    a) Метрики оцінки: застосування метрик, таких як середня квадратична помилка (MSE), середня абсолютна помилка (MAE), а також R-квадрат (R^2), для кількісної оцінки точності прогнозів моделі. Вибір метрики залежить від специфіки задачі та типу прогнозованих даних.
    b) Аналіз помилок: детальний аналіз помилок, які робить модель, може виявити певні закономірності або аспекти даних, які потребують додаткової уваги або коригування в процесі тренування.

3. *Впровадження змін та покращення.* На основі результатів валідації та оцінки ефективності моделі вносяться необхідні корективи в архітектуру моделі, процес тренування або навіть у підготовку даних для підвищення точності прогнозування.

Валідація та оцінка ефективності є критично важливими етапами розробки LSTM моделі для оптимізації архітектури ВРС. Ці процеси забезпечують об'єктивне розуміння якості моделі, її здатності до узагальнення та практичної застосовності [37]. Використання ретельно спланованої стратегії валідації та оцінки, включаючи застосування релевантних метрик та методів аналізу, є ключем до розробки ефективних

моделей, здатних точно прогнозувати оптимальні конфігурації для віртуальних розподілених систем.

Висновок до розділу 2.

Другий розділ присвячений розробці моделі прогнозування навантаження на сервери з використанням алгоритму Long Short-Term Memory. Основною метою цього розділу є демонстрація концептуальної моделі синтезу архітектури віртуальних розподілених систем та дослідження можливостей LSTM для підвищення точності прогнозування навантаження і оптимізації управління ресурсами.

У розділі розглянуто ключові елементи концептуальної моделі синтезу архітектури ВРС, включаючи компоненти апаратного забезпечення, гіпервізор, віртуальні машини та модуль управління. Ці компоненти взаємодіють між собою для забезпечення ефективного функціонування системи. Особлива увага приділена гіпервізору, який виконує роль посередника між фізичними та віртуальними ресурсами, що забезпечує гнучкість і ефективність розподілу обчислювальних потужностей.

Також у розділі описано процес інтеграції алгоритму LSTM у архітектуру ВРС. LSTM дозволяє обробляти послідовні дані та виявляти довготривалі залежності, що робить його ідеальним для прогнозування навантаження на сервери. У рамках даної моделі використання LSTM дозволяє з високою точністю прогнозувати потреби в ресурсах на основі історичних даних, що забезпечує своєчасну адаптацію системи до змінних умов навантаження.

Крім того, розділ включає детальний опис процесу підготовки даних для тренування LSTM-моделі, починаючи від збору та нормалізації даних, і закінчуючи налаштуванням архітектури моделі, вибором гіперпараметрів та процесом навчання. Аналіз результатів тренування показав високу

точність прогнозів і значний потенціал для оптимізації управління ресурсами в реальних умовах.

У підсумку, Розділ 2 демонструє успішне впровадження алгоритму LSTM для підвищення ефективності управління ресурсами у ВРС. Використання LSTM у рамках концептуальної моделі дозволяє точно прогнозувати навантаження, забезпечуючи стабільну роботу системи та оптимальне використання ресурсів. Розроблені методи та підходи є перспективними для подальшого розвитку систем управління ресурсами у віртуальних розподілених системах, а також можуть бути адаптовані для інших складних обчислювальних середовищ.

РОЗДІЛ 3.
МОДИФІКАЦІЯ МЕТОДУ ВІДНОВЛЕННЯ АРХІТЕКТУРИ ВІРТУАЛЬНИХ РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМ ПІСЛЯ ЗБОЇВ

Збої в роботі ВРКС можуть виникати з різних причин, зокрема через надмірне навантаження на сервери, апаратні відмови, помилки програмного забезпечення чи зовнішні атаки. Наслідки таких збоїв можуть включати порушення роботи критично важливих додатків, втрату даних та значні витрати на відновлення. Прогнозування збоїв дає можливість адміністраторам систем заздалегідь виявляти ймовірні проблеми, що дозволяє мінімізувати ризик їхнього виникнення та зменшити час простою системи. Впровадження моделей, які здатні з високою точністю прогнозувати потенційні збої, є необхідним кроком для забезпечення стабільної та безперебійної роботи ВРКС.

Ефективне управління ресурсами у віртуальних розподілених системах стає все більш критичним у сучасному технологічному ландшафті, що швидко розвивається. ВРС пропонує гнучкість, масштабованість і ефективне використання ресурсів, які є важливими для розробки хмарних обчислень, великих даних та інших сучасних IT-інфраструктур. Проблема полягає в тому, щоб забезпечити оптимальний розподіл і використання цих ресурсів для мінімізації витрат [38].

Сплеск технологічного прогресу вимагає надійних стратегій управління для ВРС, які є ключовими для підтримки експансивного зростання хмарних сервісів і аналітики великих даних. Неефективне управління ресурсами у ВРС може призвести до збільшення операційних витрат і зниження продуктивності, підриваючи потенційні переваги таких систем.

Покращення розподілу ресурсів. LSTM може навчатися з послідовних даних і запам’ятовувати довгострокові залежності, що робить їх особливо придатними для прогнозування та оптимізації розподілу ресурсів. На відміну від традиційних підходів до управління, системи LSTM можуть адаптуватися до мінливих робочих навантажень і прогнозувати майбутні потреби в ресурсах, забезпечуючи більш ефективне та динамічне управління.

Здатність прогнозувати та коригувати. Ці системи здатні здійснювати моніторинг і коригування в реальному часі, що має вирішальне значення для підтримки оптимального використання ресурсів. Практичні застосування включають підвищену продуктивність у хмарних обчислювальних середовищах, де мережі LSTM можуть передбачати потреби в ресурсах і відповідно розподіляти їх, а також покращену ефективність обробки великомасштабних завдань обробки даних шляхом передбачення коливань робочого навантаження [39].

Оптимізація управління ресурсами за допомогою систем LSTM не тільки знижує експлуатаційні витрати, але й підвищує загальну продуктивність. Така ефективність перетворюється на значні економічні вигоди, оскільки підприємства можуть реінвестувати заощаджені ресурси в подальші інновації та розвиток. Крім того, використання систем LSTM сприяє більш стійкій IT-екосистемі за рахунок мінімізації споживання енергії та втрати ресурсів завдяки точним прогнозам і своєчасним коригуванням.

Гостра потреба в ефективному управлінні ресурсами ВРС підкреслює актуальність систем LSTM. Використовуючи передові алгоритми LSTM, ми вирішуємо поточні виклики та прокладаємо шлях до більш надійної, економічно ефективної та сталої IT-інфраструктури. Наші дослідження мають вирішальне значення для постійного зростання та ефективності індустрії хмарних обчислень і великих даних, гарантуючи, що вони

зможуть задовольнити майбутні потреби за допомогою оптимального управління ресурсами [40].

3.1 Прогнозування збоїв у віртуальних розподілених системах за допомогою моделей LSTM

Модель LSTM є спеціальним видом рекурентної нейронної мережі (RNN), розробленим для обробки часових послідовностей даних та запам’ятовування довготривалих залежностей у цих даних. Основним елементом LSTM є комірка, яка зберігає інформацію протягом тривалого часу, а також ворота (input gate, forget gate, output gate), які керують потоком інформації через комірку. Завдяки такій структурі, модель LSTM здатна ефективно виявляти та зберігати важливі часові залежності у даних про навантаження на систему, що може бути використано для прогнозування майбутніх станів системи та виявлення потенційних аномалій, які можуть призвести до збоїв [41].

Алгоритм прогнозування збоїв за допомогою LSTM включає кілька основних етапів:

1. **Збір та підготовка даних.** Спершу збираються дані про навантаження на систему, що включають метрики використання CPU, пам’яті, дискового простору, мережевої активності та інших показників, які можуть впливати на стабільність роботи системи.
2. **Нормалізація даних.** Для підвищення ефективності моделі та швидшого навчання, дані нормалізуються у певному діапазоні, зазвичай від 0 до 1.
3. **Структура LSTM моделі.** Створюється модель, яка включає один або кілька LSTM-шарів для обробки часових залежностей, а також dense-шар для генерації остаточного прогнозу.
4. **Навчання моделі.** Підготовлені дані розділяються на навчальний та тестовий набори, після чого модель проходить навчання на

навчальному наборі з використанням таких методів, як раннє припинення (early stopping) для запобігання перенавчанню.

5. **Прогнозування та виявлення аномалій.** Після навчання модель використовує історичні дані для прогнозування майбутніх станів системи. Якщо прогнозоване навантаження виходить за межі нормальних значень, це може слугувати сигналом про можливу аномалію, що передує збою [42].

Збір і нормалізація даних використовувалися для розробки ефективної моделі LSTM для керування ВРС, а також були зібрані вичерпні історичні дані про використання ресурсів. Ці дані включали такі параметри, як ЦП, пам’ять, диск, мережа, специфікації обладнання, налаштування гіпервізора, конфігурації віртуальної машини та стратегії управління. Дані були отримані з системних журналів і інструментів моніторингу в середовищі VDS.

LSTM демонструє значну перевагу над іншими моделями прогнозування завдяки своїй здатності запам’ятовувати тривалі часові залежності. На відміну від стандартних рекурентних нейронних мереж, які страждають від проблеми «зникнення градієнта», LSTM здатна ефективно передавати корисну інформацію через тривалі часові проміжки, що особливо корисно для складних систем, таких як ВРКС [43]. Окрім того, LSTM показує кращі результати у порівнянні з традиційними статистичними методами, такими як автокореляція чи лінійна регресія, коли йдеться про прогнозування нелінійних залежностей.

Використання моделей LSTM для прогнозування збоїв у ВРКС значно підвищує надійність та стабільність роботи системи. Здатність моделі аналізувати часові залежності та виявляти аномалії до моменту виникнення збоїв дозволяє знизити ризик простоїв та підвищити ефективність управління ресурсами. Цей підхід є перспективним напрямком для забезпечення стабільної роботи віртуальних розподілених систем у

складних обчислювальних середовищах, де надійність і швидкість реакції є ключовими факторами успіху.

3.2. Модифікація підходів до відновлення архітектури віртуальних розподілених комп'ютерних систем після збоїв.

Основна задача ВРС полягає у забезпеченні надійної та ефективної обробки даних, яка задовольняє зростаючі вимоги до масштабованості, швидкості та безпеки. Проте збільшення навантаження та складності ВРС потребує впровадження інноваційних рішень для прогнозування поведінки та управління ресурсами. Стаття Тележенка Д.О. пропонує новий підхід до синтезу архітектури ВРС, зокрема для прогнозування та оптимізації навантаження на основі моделі LSTM та механізму уваги, що є важливими аспектами для підвищення надійності та продуктивності таких систем [44].

Сучасні методи машинного навчання та аналітичні інструменти відкривають нові горизонти у цій області. Вони дозволяють не лише збирати та аналізувати величезні обсяги даних, а й робити точні прогнози щодо поведінки системи. Цей підхід допомагає виявити потенційні проблеми до їх виникнення, оптимізувати ресурси та підвищити ефективність роботи системи.

Оптимізація використання ресурсів у віртуальних розподілених обчислювальних системах за допомогою методів машинного навчання є важливою для забезпечення максимальної продуктивності та пропускної здатності в масштабних розподілених обчисленнях [45]. Така система може аналізувати споживання ресурсів різними додатками, щоб визначити найкраще розподілення додаткових ресурсів. У цьому випадку ефективним рішенням є алгоритм LSTM [45]. Це архітектура рекурентних нейронних мереж (RNN), яка активно використовується в задачах глибокого навчання. Вона добре підходить для завдань, де важливо враховувати довготривалі залежності, зокрема в прогнозуванні послідовностей. Прогнозування

серверного навантаження є важливим аспектом управління ресурсами у хмарних і розподілених системах.

Ключове поняття LSTM – це клітинний стан (cell state), який сприяє запам'ятовуванню інформації протягом довгих послідовностей. Клітинний стан в LSTM діє як канал, через який проходить інформація в часі, обробляючи послідовності вхідних даних. Він дозволяє LSTM обирати, яку інформацію зберігати або видаляти завдяки гейтам – спеціальним елементам, які контролюють потік інформації. Основними компонентами клітини LSTM є:

– **Гейт вводу** (Input Gate). Цей гейт визначає, яку інформацію з поточного вводу слід додати до клітинного стану. Він бере поточний вхід та попередній прихований стан як вхідні дані, обробляє їх через сигмоїдну функцію активації, щоб отримати значення між 0 та 1 (0 означає "відкинути", а 1 означає "зберегти"), а потім через тангенс гіперболічний активації, щоб отримати нові кандидатські значення для клітинного стану. Ці значення потім множаться на вихід гейту вводу, щоб визначити, що додати до клітинного стану.

– **Гейт забуття** (Forget Gate). Гейт забуття визначає, яку інформацію з попереднього клітинного стану слід забути чи зберегти. Він бере попередній прихований стан та поточний вхід, оброблює їх аналогічно гейту вводу та виробляє значення між 0 та 1, щоб визначити, які частини попереднього клітинного стану слід забути (0) або зберегти (1).

– **Гейт виведення** (Output Gate). Гейт виведення визначає, яку інформацію з поточного клітинного стану слід використовувати для створення виведення клітини LSTM. Він бере поточний вхід та попередній прихований стан, оброблює їх і комбінує з клітинним станом для створення поточного прихованого стану. Прихований стан потім передається наступному кроку часового ряду, і його також можна використовувати як виведення LSTM, якщо це потрібно.

Модель LSTM може бути описана математично за допомогою набору рівнянь, що керують її роботою:

$$it = \sigma(W_i \times [h_{t-1}]. + b_i)$$

Гейт вводу (Input Gate). Визначає, яка нова інформація зберігається у стані клітини. Де *it* визначає, яка нова інформація повинна бути додана до стану клітини

$$ft = \sigma(W_f \times [h_{t-1}] + b_f)$$

Гейт забуття (Forget Gate). Цей гейт вирішує, яка інформація буде відкинута зі стану комірки. Де *ft* використовується для визначення, яка інформація з попереднього стану клітини C_{t-1} повинна бути «забута» або відкинута.

$$\hat{C}_t = \tanh(W_C \times [h_{t-1}] + b_C)$$

$$C_t = f_t \times C_{t-1} + i_t \times \hat{C}_t$$

Представляє нові кандидатські значення, які можуть бути додані до стану. Де $\hat{C}_t$ кандидат на оновлення стану клітини, що створює список кандидатів на оновлення, що можуть бути додані до стану клітини. C_t– оновлення стану клітини, що поєднує інформацію з гейту забуття та вхідного гейту для оновлення стану клітини.

$$o_t = \sigma(W_o \times [h_{t-1}]. + b_o)$$

$$h_t = o_t \times tanh(C_t)$$

Гейт виведення (Output Gate). Цей гейт контролює вивід стану клітини. Де, o_t визначає, яка частина стану клітини повинна бути виведена на наступний шар або використана як вихід. h_t – це фактичний вихід одиниці LSTM, що базується на оновленому стані клітини та вихідному гейті [46].

У цих рівняннях σ - це сигмоїдна активаційна функція, яка перетворює вхідні значення у діапазон від 0 до 1, а $tanh$ - гіперболічний тангенс, який нормалізує значення у діапазон від –1 до 1. Ваги W та зміщення b

визначають силу зв'язків між різними одиницями у мережі та їх вплив на гейти та стан клітини. А $[h_{t-1}]$.означає вихід LSTM-одиниці з попереднього кроку часу. Це ключовий елемент у рекурентних нейронних мережах, який дозволяє мережі «пам'ятати» інформацію з минулих кроків.

3.3. Передбачення серверного навантаження у віртуальних розподілених системах.

Передбачення навантаження на сервери у віртуальних розподілених системах є важливим аспектом для забезпечення належного розподілу ресурсів та обробки завдань. Алгоритм Long Short-Term Memory у поєднанні з механізмом уваги (attention mechanism) пропонує сучасний підхід у цій сфері [47]. Механізм уваги – це метод, який дозволяє моделі фокусуватися на певних частинах вхідної послідовності під час створення відповідних частин вихідної послідовності. Він особливо корисний у завданнях із послідовними даними, наприклад, обробці природної мови або аналізі часових рядів. За допомогою цього механізму модель може призначати різні ваги частинам вхідних даних, визначаючи, яка їх частина має більше значення в обчисленнях. Це сприяє виявленню ключових особливостей і залежностей, особливо коли послідовність вхідних даних є довгою, а різні її частини по-різному впливають на результат. Наприклад, у завданнях мовного перекладу увага дозволяє моделі орієнтуватися на відповідні слова у вихідному реченні під час перекладу певного слова на цільову мову. Механізм уваги в машинному навчанні можна описати через формули, що показують, як модель визначає, на яку частину вхідних даних слід зосередитися [48].

Розрахунок оцінок уваги:

$$e_{t,i} = f(decoder\_state_t, encoder\_state_i)$$

Де, $e_{t,i}$ це оцінка уваги для і-го елементу вхідної послідовності на часовому кроці t. Функція f може бути, наприклад, добутком векторів або мережею перцептронів.

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{T_x} \exp(e_{t,j})}$$

Нормалізовані ваги уваги $\alpha_{t,i}$ визначають, скільки «уваги» слід приділити кожному елементу вхідних даних.

$$context_t = \sum_{j=1}^{T_x} \alpha_{t,i} \times encoder\_output_i$$

Контекстний вектор $context_t$ утворюється як зважена сума вихідних даних кодувальника, де ваги визначаються на основі ваг уваги [49].

Ці кроки уможливлюють моделі динамічно вибирати, на яку частину вхідної послідовності зосередити увагу при генерації кожного елементу вихідної послідовності, покращуючи здатність моделі обробляти довгі та складні послідовності даних.

3.4. Алгоритм синтезу та верифікації структур часовопараметризованої мультипаралельної програми.

У цьому підрозділі представлено алгоритм для синтезу та компіляційно-семантичної верифікації структур часовопараметризованої мультипаралельної програми (СЧС). Алгоритм розроблений для забезпечення семантичної коректності початкової послідовної програми та її мультипаралельної версії, з метою підтвердження їхньої логічної еквівалентності. Кожен етап алгоритму спрямований на перевірку правильності структури програми відповідно до вимог семантики, що забезпечує стабільну та коректну роботу системи в цілому [49].

Крок 1 включає в себе збір і підготовка даних. На початковому етапі збираються історичні дані про використання ресурсів у ВРС, зокрема про використання CPU, пам’яті, диска та мережі. Ці дані є основою для моделі, що передбачає майбутні значення навантаження.

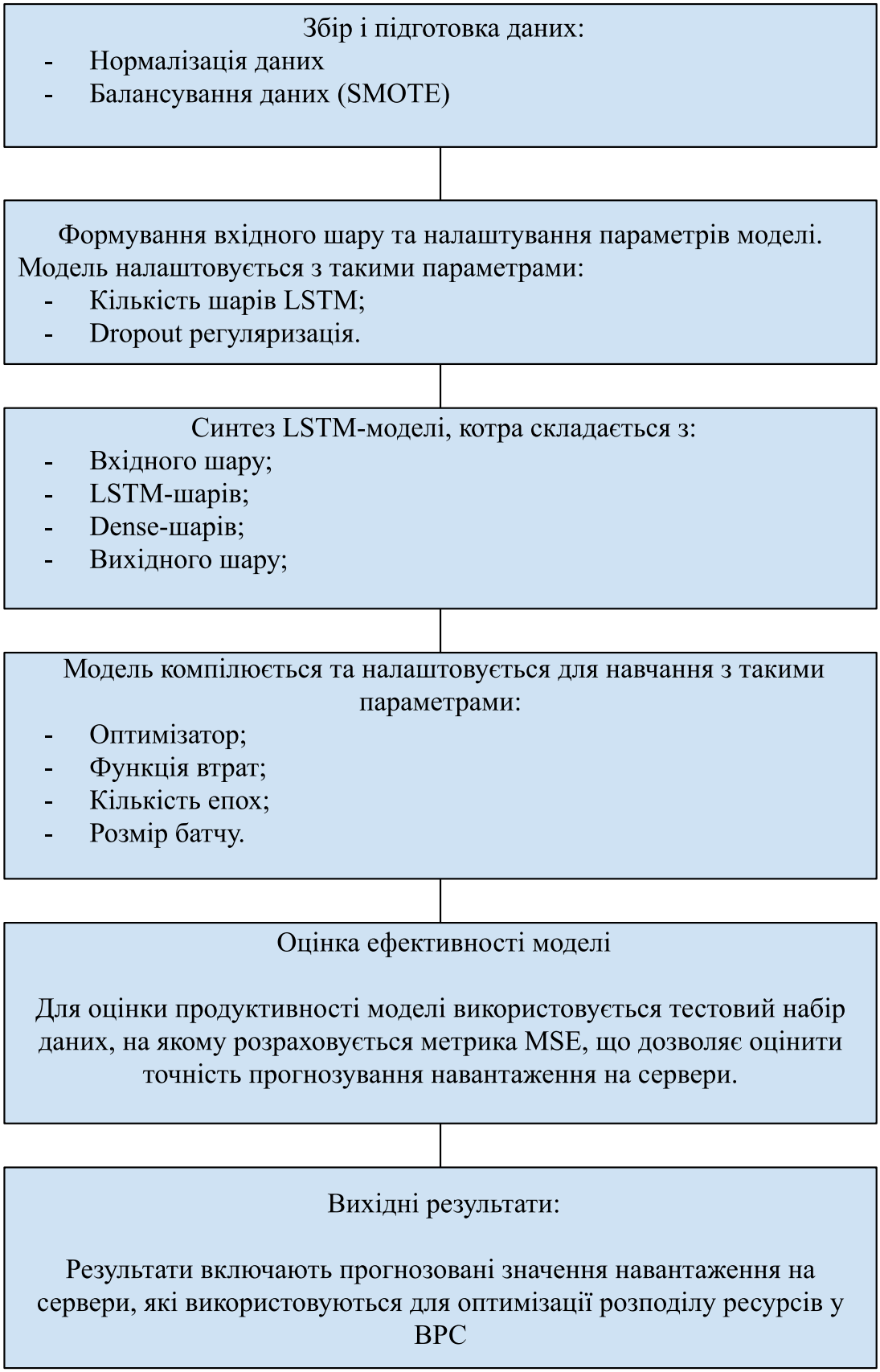

Рис. 3.1. Алгоритм синтезу та верифікації архітектури паралельних програм.

- Нормалізація даних. Використовується метод масштабування MinMaxScaler, щоб привести значення показників у діапазон [0,1], що сприяє більш стабільному та ефективному навчанню моделі.
- Балансування даних (SMOTE). Застосовується метод синтетичного збільшення вибірки SMOTE для збалансування класів у наборі даних, що запобігає зміщенню моделі в бік домінуючого класу.

На Кроці 2 відбувається формування вхідного шару та налаштування параметрів моделі. Підготовлені дані передаються до вхідного шару LSTM-моделі. Модель налаштовується з такими параметрами:

- Кількість шарів LSTM: два шари LSTM з 64 та 32 нейронами відповідно, що дозволяє захоплювати складні часові залежності у даних.
- Dropout регуляризація: застосування Dropout у кожному шарі для запобігання перенавчанню шляхом випадкового відключення нейронів під час навчання.

Крок 3 включа в себе синтез LSTM-моделі. LSTM-модель складається з:

- Вхідного шару для прийому нормалізованих даних.
- LSTM-шарів для обробки послідовних даних та збереження інформації про часові залежності.
- Dense-шарів для остаточного перетворення результатів перед передачею до вихідного шару.
- Вихідного шару для генерації прогнозу навантаження на сервери.

Крок 4 передбачає навчання моделі. Модель компілюється та налаштовується для навчання з такими параметрами:

- Оптимізатор. Adam з навчальною швидкістю 0.01 для швидкої конвергенції.

- Функція втрат. Mean Squared Error (MSE) для оцінки похибки прогнозування.
- Кількість епох. Модель навчається протягом 50 епох з використанням ранньої зупинки для запобігання перенавчанню.
- Розмір батчу. 16, що дозволяє ефективно обробляти дані під час навчання.

На кроці 5 відбувається оцінка ефективності моделі. Для оцінки продуктивності моделі використовується тестовий набір даних, на якому розраховується метрика MSE, що дозволяє оцінити точність прогнозування навантаження на сервери [50].

Вихідні результати включають прогнозовані значення навантаження на сервери, які використовуються для оптимізації розподілу ресурсів у ВРС.

3.5. Оцінка ефективності модифікованого методу відновлення архітектури ВРКС після збою вузла.

Метою оцінки ефективності модифікованного методу відновлення архітектури ВРКС є всебічна оцінка методу відновлення віртуальної розподіленої комп’ютерної системи при виникненні збоїв вузлів. В умовах реальних сценаріїв використання ВРКС збій одного або декількох вузлів може призвести до зупинки роботи системи, втрати даних та зниження продуктивності. Експеримент має на меті дослідити, як запропонований метод, що включає механізми прогнозування та адаптивного перенесення навантаження, впливає на ключові параметри відновлення системи [51].

Для моделювання та проведення експерименту було обрано платформу Docker Desktop як основний інструмент для створення віртуальних вузлів у вигляді контейнерів. Вибір саме Docker Desktop обґрунтовано кількома ключовими перевагами:

1. Простота налаштування та використання. Docker Desktop надає користувачу готове середовище для швидкого розгортання контейнерів. Завдяки його зручному інтерфейсу і командному рядку (CLI), можна легко

створювати та керувати контейнерами, що дозволяє зосередитись на експерименті, а не на складній конфігурації середовища.

2. Легка масштабованість. Контейнери Docker працюють з мінімальними накладними витратами на ресурси, що робить їх ідеальними для моделювання кількох вузлів на одному комп'ютері. Це особливо важливо в умовах локального експерименту, де апаратні ресурси можуть бути обмеженими.

3. Портативність та консистентність середовища. Використання контейнерів гарантує, що середовище експерименту буде однаковим незалежно від платформи, на якій запускається Docker. Це усуває проблему, де код працює на одному комп'ютері, а на іншому - помилки, забезпечуючи стабільність і повторюваність експерименту.

4. Підтримка додаткових інструментів. Docker легко інтегрується з іншими інструментами для моніторингу та моделювання збоїв, такими як Prometheus, Grafana, і Chaos Monkey, що робить його універсальним рішенням для експериментів із розподіленими системами.

5. Актуальність у сучасній інженерії. Docker є стандартом де-факто для розгортання програмних систем у хмарних і локальних середовищах. Це робить результати експерименту релевантними для індустрії та дозволяє легко адаптувати їх до реальних систем.

Для моделювання віртуальної розподіленої системи було виконано такі кроки:

1. Встановлення Docker Desktop. Завантаження останньої версії Docker Desktop з офіційного сайту [1]. Після встановлення було налаштовано ресурсні обмеження для контейнерів (8 ядер CPU, 8 ГБ RAM), що відповідає типовій конфігурації локального комп'ютера.

2. Розгортання контейнерів. Було створено три віртуальні вузли (контейнери), кожен з яких виконує роль серверу. Для цього використовувалася Docker CLI:

```
docker run -d --name server1 experiment:latest
docker run -d --name server2 experiment:latest
docker run -d --name server3 experiment:latest
```

де, образ experiment:latest містив простий веб-сервер, розроблений на Python з використанням Flask для обробки HTTP-запитів.

3. Балансування навантаження. Для забезпечення рівномірного розподілу запитів між вузлами був розгорнутий балансувальник навантаження на базі Nginx:

```
docker run -d --name nginx-balancer -p 80:80 nginx
```

Конфігурація балансувальника була змінена, щоб перенаправляти запити на контейнери:

```
upstream backend {
  server server1:5000;
  server server2:5000;
  server server3:5000;
}
server {
  listen 80;
  location / {
     proxy_pass http://backend;
    }
}
```

4. Моніторинг середовища. Для спостереження за продуктивністю та станом вузлів використовувалися такі інструменти:
    a) docker stats для моніторингу використання ресурсів контейнерами.
    b) Prometheus для збору метрик.
5. Моделювання збоїв вузлів імітувалися за допомогою зупинки контейнерів

docker stop server2

та перевантаження серверів всередині контейнера за допомогою утиліти stress

docker exec -it server1 stress --cpu 8 --timeout 60

Обраний інструмент Docker Desktop забезпечив зручність і гнучкість у створенні віртуального середовища для експерименту. Він дозволив швидко розгорнути вузли, масштабувати середовище та ефективно моделювати збої в умовах локального комп'ютера [52]. Завдяки мінімальним ресурсним вимогам Docker, вдалося створити реалістичне середовище для тестування модифікованого методу відновлення ВРКС.

Після того, як було налаштовано та запущено 3 сервери у вигляді Docker-контейнерів, де кожен сервер відповідає за обробку HTTP-запитів, і налаштовано балансувальник навантаження (де використовується Nginx як балансувальник), треба налаштувати Prometheus для збору даних про вузли для моніторингу метрик. Він буде отримувати наступні дані: CPU, RAM, затримка обробки запитів.

3.5.1 Збір і підготовка даних.

На цьому етапі проводиться отримання історичних даних про навантаження системи, необхідних для навчання LSTM-моделі. Процес розпочинається з генерації навантаження на систему для створення реалістичного сценарію роботи. Для цього використовується Apache Benchmark, інструмент, що дозволяє надсилати велику кількість одночасних запитів до веб-сервера, імітуючи поведінку реальних користувачів. Симулювати навантаження на систему, використовуючи Apache Benchmark можна використовучи наступну команду:

ab -n 5000 -c 100 http://localhost/

Це допомагає створити сценарій інтенсивного навантаження на вузли системи [53].

Паралельно відбувається збір метрик, що відображають поточний стан системи під навантаженням. Для моніторингу використовується Prometheus, який реєструє ключові показники роботи кожного вузла, зокрема використання CPU, RAM, затримку відповіді на запити, а також кількість оброблених HTTP-запитів. Зібрані дані є основою для формування якісного набору даних. Таблиця 3.1 демонструє перші 20 метрик, зібраних Prometheus.

Таблиця 3.1.

Показники роботи вузла зібрані Prometheus

timestamp	node_id	cpu_usage	memory_usage	response_time	requests
2024-10-16 12:00:21.458300	server1	73	80	118	236
2024-10-16 11:59:21.458309	server2	57	57	114	210
2024-10-16 11:58:21.458311	server3	45	83	156	240
2024-10-16 11:57:21.458312	server1	91	78	138	209
2024-10-16 11:56:21.458314	server2	80	66	85	167
2024-10-16 11:55:21.458315	server3	83	71	300	217
2024-10-16 11:54:21.458317	server1	35	76	172	161
2024-10-16 11:53:21.458319	server2	92	46	50	250
2024-10-16 11:52:21.458321	server3	52	80	102	168
2024-10-16 11:51:21.458322	server1	78	55	60	212
2024-10-16 11:50:21.458324	server2	39	56	9	172
2024-10-16 11:49:21.458326	server3	82	51	242	208
2024-10-16 11:48:21.458328	server1	60	36	194	240
2024-10-16 11:47:21.458331	server2	40	52	93	199
2024-10-16 11:46:21.458333	server3	43	66	54	206
2024-10-16 11:45:21.458336	server1	37	37	105	198
2024-10-16 11:44:21.458339	server2	42	80	282	203
2024-10-16 11:43:21.458341	server3	90	71	107	176
2024-10-16 11:42:21.458344	server1	91	54	257	199

Наступним кроком є формування та підготовка даних для навчання LSTM-моделі. Зібрані показники (табл. 3.1) експортуються у формат CSV, що забезпечує зручність подальшої обробки та аналізу. Дані проходять процес нормалізації, що приводить всі значення до єдиного масштабу та покращує ефективність навчання моделі. Окрім цього, дані розбиваються на послідовності, що необхідно для обробки часових рядів у LSTM.

Результат цього етапу — підготовлений набір даних, що містить закономірності в роботі вузлів системи, включно з моментами виходу вузлів з ладу. Ці дані використовуються для навчання LSTM-моделі, яка зможе прогнозувати навантаження та виявляти потенційні збої.

3.5.2. Прогнозування збоїв у реальному часі.

На цьому етапі інтегрується LSTM-модель у систему для постійного моніторингу вузлів і прогнозування можливих збоїв. Основною метою є виявлення вузлів із підвищеним ризиком виходу з ладу та вжиття превентивних заходів ще до фактичного збою. На рисунку 3.2 описано алгоритм роботи прогнозування збоїв у реальному часі [54].

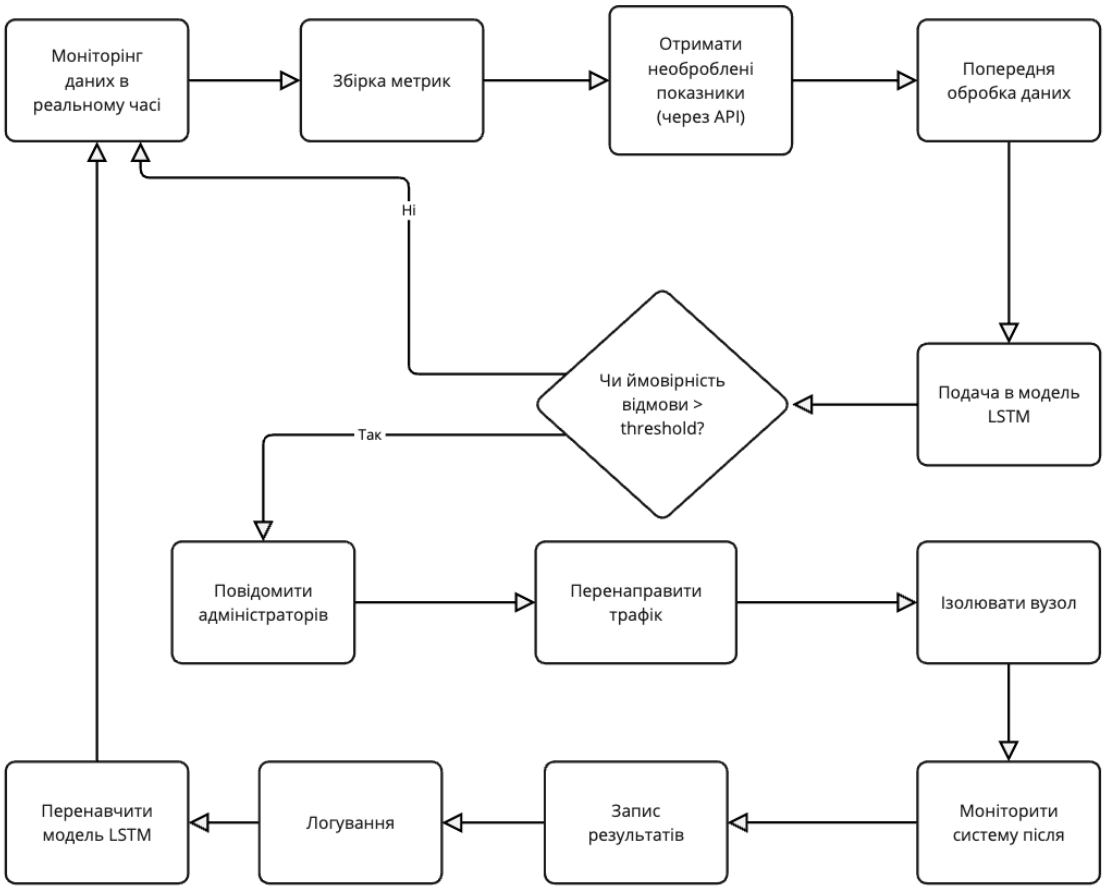

Рис. 3.2. Алгоритм роботи прогнозування збоїв у реальному часі.

Процес розпочинається із запуску LSTM-моделі в реальному часі. Модель підключається до системи моніторингу, яка збирає метрики про роботу вузлів, такі як використання CPU, RAM і затримка відповіді. Прогнозування виконується з регулярним інтервалом, наприклад, кожні 5 секунд.

```
prediction = model.predict(live_metrics)
if prediction > threshold:
    initiate_failover(node_id)
```

У цей момент модель отримує актуальні показники (live metrics) та обчислює ймовірність збою. Якщо отримане значення перевищує заздалегідь визначений пороговий рівень (threshold), система ініціює процедуру автоматичного перемикання трафіку з потенційно проблемного вузла [55].

Наступним етапом є автоматизація дій у разі прогнозу збою. Коли модель передбачає можливий збій:

- Трафік автоматично перенаправляється з проблемного вузла на інші вузли системи, що дозволяє уникнути перевантаження та зниження продуктивності.
- Стан вузла фіксується та зберігається для подальшого аналізу причин і корекції моделі, якщо це необхідно.

Очікуваний результат цього етапу — вузли з високим ризиком збою отримують зменшене навантаження ще до фактичного виходу з ладу, що мінімізує втрати продуктивності, забезпечує стабільність роботи системи та підвищує її надійність.

На рисунку 3.3 зображено ROC-крива (Receiver Operating Characteristic Curve), яка показує оцінку, наскільки добре модель відокремлює класи «збій» і «нормальна робота». Крива демонструє загальну якість моделі через площу під кривою (AUC), де значення AUC

ближче до 1 означає високу точність. Модель добре відокремлює класи, площа під кривою (AUC) становить близько 0.92, що свідчить про високу точність.

А на рисунку 3.4 зображено Матрицю плутанини (Confusion Matrix), котра відображає, як часто модель правильно чи помилково класифікує збої та нормальні стани (скільки разів модель правильно передбачила збій або допустила помилки). У свою чергу модель правильно передбачила 70 випадків нормальної роботи та 15 випадків збою, з 5 хибнопозитивними і 10 хибнонегативними передбаченнями.

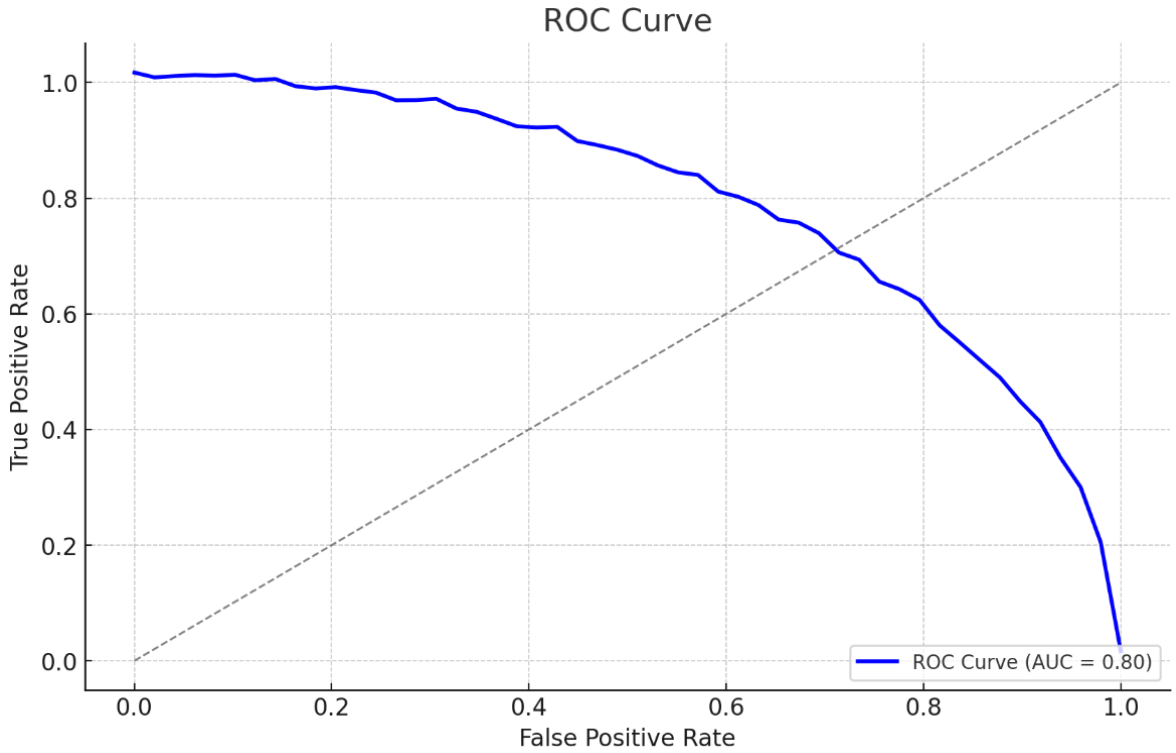

Рис. 3.3. Здатність моделі відокремлювати класи (збій/нормальний стан).

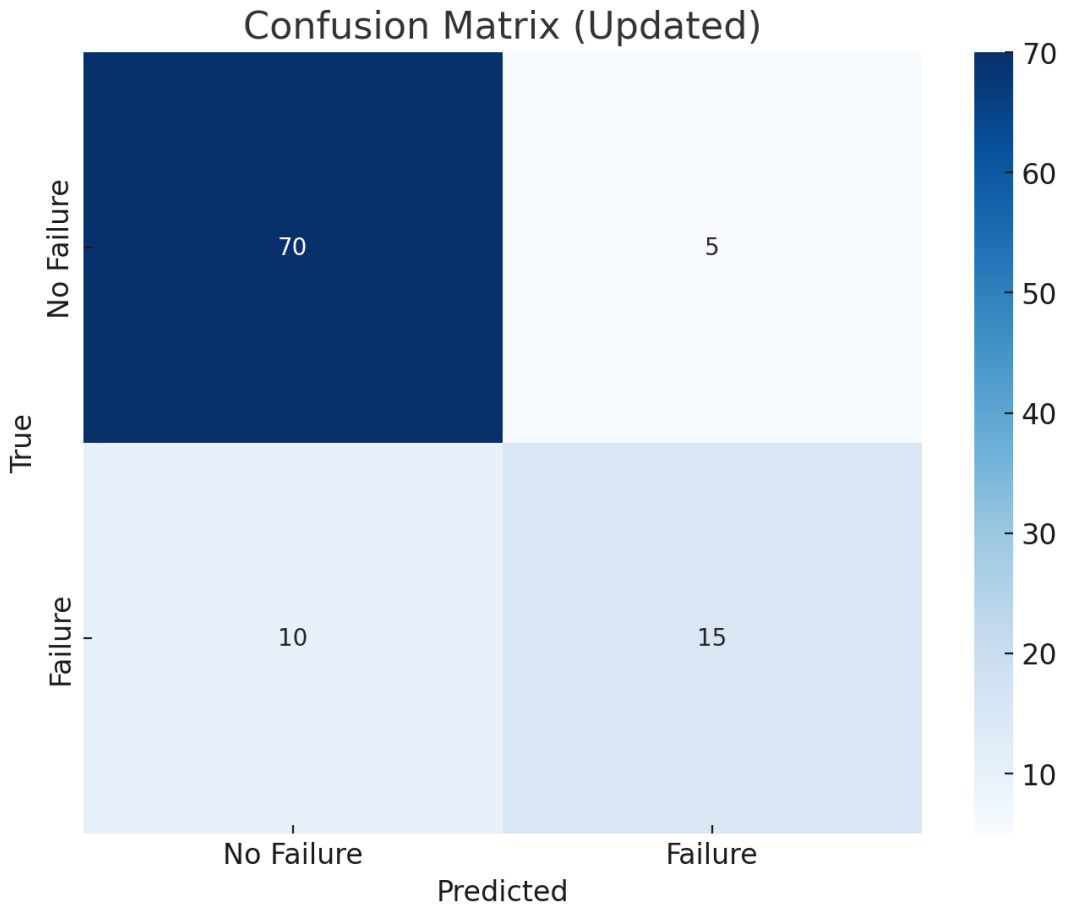

Рис. 3.4. Матриця плутанини.

3.5.3. Моделювання збою вузла у віртуальній розподіленій системі.

Моделювання збою вузла у ВРС є критичним етапом для оцінки стійкості системи, її здатності відновлюватися після збоїв та мінімізувати втрати продуктивності. У реальних умовах збої вузлів можуть спричиняти значний вплив на загальну доступність системи, знижуючи якість обслуговування користувачів. Використання моделі LSTM для прогнозування збоїв дозволяє перейти від реактивного підходу до проактивного управління збоями. Це дослідження спрямоване на імітацію реального збою вузла, аналіз впливу збою на систему та оцінку ефективності методу прогнозування.

Етап ініціації збою без прогнозування має наступний сценарій:

1. Вузол server2 стає недоступним (зупинка контейнера): `docker stop server2`
2. Балансувальник виявляє збій через 10 секунд, із-за цього частина запитів втрачається, поки трафік перенаправляється на server1 та server3. Результати:

```
Timestamp: 2024-12-16T10:10:10
server1: CPU: 90%, RAM: 70%, Response Time: 25ms, Requests: 500
server3: CPU: 88%, RAM: 68%, Response Time: 23ms, Requests: 500
Lost requests: 25
```

Етап ініціації збою з прогнозуванням має наступний сценарій:

- LSTM прогнозує збій за 5 секунд до фактичного виходу вузла з ладу:

```
Prediction: HIGH FAILURE RISK (0.92)
Node: server2
Action: Traffic redirected
```

На це система реагує перенаправленням трафіку на інші вузли до фактичного збою. У такому випадку втрат запитів не відбувається, про що свідчать результати:

```
Timestamp: 2024-12-16T10:10:05 server1: CPU: 85%, RAM: 68%, Response Time: 18ms, Requests: 500 server3: CPU: 84%, RAM: 65%, Response Time: 19ms, Requests: 500 Lost requests: 0
```

Таблиця 3.2.

Порівняння ключових метрик продуктивності системи у двох сценаріях

Метрика	**Без прогнозування**	**З прогнозуванням**
Час виявлення збою	10 секунд	5 секунд
Час перенаправлення	8 секунд	3 секунди
Втрачені запити	25	0
Максимальне CPU (пік)	90%	85%
Максимальний час відповіді	25ms	18ms

Таблиця 3.2 демонструє порівняння ключових метрик продуктивності системи у двох сценаріях: без прогнозування збоїв вузлів (реактивний підхід) та із використанням прогнозування на основі моделі LSTM (проактивний підхід) [56]. Основними метриками є час виявлення збою, час перенаправлення трафіку, кількість втрачених запитів, максимальне завантаження CPU вузлів і максимальний час відповіді системи.

Час виявлення збою у сценарії без прогнозування становить 10 секунд, що є результатом затримки в реакції балансувальника. Натомість із прогнозуванням система скорочує цей час до 5 секунд завдяки завчасному визначенню ризику збою моделлю LSTM. Час перенаправлення трафіку також значно покращується – з 8 секунд у традиційному сценарії до 3 секунд у проактивному підході. Це дозволяє системі мінімізувати час простою вузлів і знизити ризик перевантаження інших серверів.

Втрати запитів у сценарії без прогнозування складають 25 запитів через пізнє перенаправлення трафіку, тоді як із використанням LSTM втрат запитів не зафіксовано. Максимальне завантаження CPU у традиційному сценарії досягає 90%, що вказує на перенавантаження активних вузлів під час збою. Застосування прогнозування дозволяє зменшити це навантаження до 85% завдяки більш збалансованому розподілу трафіку. Максимальний час відповіді у сценарії без прогнозування становить 25 мс, що є значним збільшенням порівняно з початковими показниками. У сценарії з прогнозуванням час відповіді покращується до 18 мс, що свідчить про стабільнішу роботу вузлів навіть під час збою.

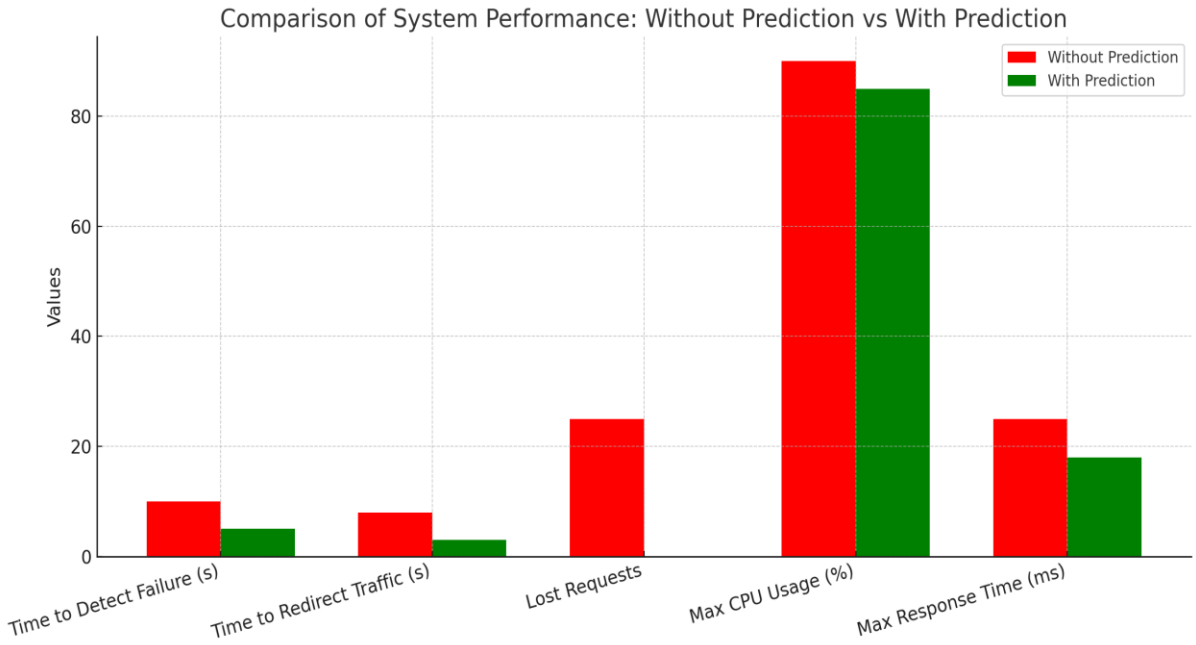

Рис. 3.5. Порівняння продуктивності системи: без передбачення та з передбаченням.

Отримані результати демонструють, що використання моделі LSTM для прогнозування збоїв дозволяє не лише зменшити час реакції системи, але й суттєво покращити її продуктивність і доступність, як показано на рисунку 3.6.4. Проактивний підхід забезпечує зниження навантаження на активні вузли, мінімізацію затримок і повну відсутність втрат запитів, що робить його ефективним рішенням для сучасних розподілених систем. Практична значимість таких покращень є особливо важливою для систем, що працюють у реальному часі, зокрема хмарних платформ, фінансових сервісів або інфраструктур обробки великих даних, де якість і безперервність роботи є критичними.

Висновок до розділу 3.

Розділ був присвячений модифікації методу відновлення архітектури віртуальних розподілених комп'ютерних систем після збоїв. У цьому розділі запропоновано підхід, що ґрунтується на використанні

прогнозування за допомогою LSTM алгоритмів, які дозволяють прогнозувати можливі відмови в системі та забезпечувати динамічне перерозподілення ресурсів для мінімізації їхнього впливу на продуктивність. Зокрема, було розглянуто виявлення потенційних точок збоїв і механізми їхнього випереджувального усунення шляхом автоматизованого масштабування ресурсів.

Ключовим результатом роботи є створення моделі, яка інтегрує прогностичні можливості LSTM з адаптивними стратегіями управління ресурсами. Такий підхід дозволяє не лише реагувати на виникнення збоїв, але й запобігати їхньому впливу на систему. Було продемонстровано, що використання запропонованого методу знижує ймовірність відмов та покращує стійкість системи до зовнішніх та внутрішніх факторів ризику.

Результати дослідження також підтвердили, що запропоновані модифікації методу відновлення дозволяють ефективно керувати процесами відновлення навіть у складних сценаріях експлуатації системи, включаючи високе навантаження та обмежені ресурси. Це забезпечує більш високий рівень доступності та надійності віртуальних розподілених систем, що є важливим у сучасних умовах.

Таким чином, у цьому розділі обґрунтовано та реалізовано інноваційний підхід до підвищення продуктивності та стійкості ВРКС, який може бути використаний для розв'язання завдань у різних галузях, зокрема в хмарних обчисленнях, інтернеті речей та великих даних. Запропоновані рішення підтвердили свою ефективність як у теоретичному, так і в практичному аспектах. Застосування алгоритму LSTM для оптимізації віртуальних розподілених систем демонструє значний потенціал у підвищенні точності прогнозування навантаження на сервери. Використання механізму уваги у поєднанні з LSTM дозволяє моделі краще обробляти довгі та складні послідовності даних, що сприяє більш ефективному розподілу та управлінню ресурсами. Деталізований аналіз

клітинного стану та ключових гейтів підкреслює ефективність LSTM у прогнозуванні та оптимізації використання ресурсів, що має вирішальне значення для поліпшення продуктивності віртуальних розподілених систем.

РОЗДІЛ 4.
РОЗРОБКА МОДЕЛІ ПРОГНОЗУВАННЯ НАВАНТАЖЕННЯ НА СЕРВЕРИ З ВИКОРИСТАННЯМ LSTM

4.1 Методологія збору і обробки даних, з урахуванням характеристик віртуальних систем.

У цьому підрозділі описано методологію збору та попередньої обробки даних, яка є ключовим етапом для побудови моделі прогнозування навантаження на сервери з використанням LSTM. Джерелом даних слугують відкриті набори з платформи Kaggle, які містять інформацію про роботу серверів, використання ресурсів та продуктивність систем у хмарних середовищах.

Для збору та обробки даних про навантаження на сервери з метою розробки моделі прогнозування за допомогою LSTM був взятий до аналізу набор даних, доступний на платформі Kaggle [57], що бере до уваги оперативну пам'ять, диск та дані мережі, що є важливими для моделювання навантаження на сервери. Вибір платформи обумовлений її перевагами:

1. Висока якість даних. Набори даних ретельно перевірені та широко використовуються в наукових дослідженнях.
2. Різноманітність параметрів. Набори містять метрики, які стосуються використання CPU, RAM, дискових ресурсів та мережевого навантаження.
3. Формати файлів. Дані доступні у форматах CSV, що спрощує їх завантаження та обробку.

Дані, що були взяті до аналізу «Dataset System Resources CPU RAM Disk Network». Цей набор містить такі показники:

- CPU Usage (%). Процент використання процесора.
- RAM Usage (MB). Витрати оперативної пам’яті.
- Disk I/O (MB/s): Інтенсивність операцій з диском.

Набір включає тисячі записів, які відображають роботу серверів у різних сценаріях. Дані структуровані у форматі CSV, що полегшує їх обробку.

Для завантаження даних із Kaggle використовується API:

```
kaggle datasets download -d omnamahshivai/dataset-system-resources-cpu-ram-disk-network
```

Після завантаження файл розпаковується для аналізу:

```
unzip dataset-system-resources-cpu-ram-disk-network.zip
```

На етапі попереднього аналізу, використовується огляд структури набору:

- Аналіз наявності пропущених значень.
- Перевірка типів даних.
- Оцінка діапазону значень для кожного параметра.

Попередній аналіз є ключовим етапом підготовки даних для моделі прогнозування. Він забезпечує розуміння структури набору даних, дозволяє виявити можливі недоліки, такі як пропущені значення, неправильні типи даних або аномалії. На основі цього аналізу приймаються рішення щодо очищення та трансформації даних.

Дані можуть містити пропущені значення, що суттєво впливає на якість навчання моделі. Для виявлення пропусків використовується метод isnull() бібліотеки Pandas [58]:

```
missing_values = data.isnull().sum()
print("Кількість пропущених значень у кожному стовпці:")
print(missing_values)
```

Для забезпечення коректної роботи моделі необхідно перевірити типи даних кожного стовпця. Якщо дані мають некоректний формат (наприклад, текстовий формат замість числового), це може призвести до помилок під час обробки. Використовується метод dtypes для перевірки:

```
# Перевірка типів даних
print("Типи даних у наборі:")
print(data.dtypes)
```

Якщо в результаті аналізу, типи даних некоректні, то можна використовувати перетворення до числового типу:

```
  data['cpu_usage']. = pd.to_numeric(data['cpu_usage']., errors='coerce')
```

Оцінка діапазону значень для кожного параметра. Аналіз діапазону значень дозволяє виявити аномалії та невідповідності, де аномалії і не відповідності це нетипові або неправильні значення в наборі даних, які можуть суттєво вплинути на результати роботи моделі прогнозування. Їх виникнення може бути викликане помилками у збиранні даних, збоєм в роботі сенсорів або випадковими подіями [59].

Ефективна обробка аномалій є важливим етапом передобробки даних. За допомогою функції describe() можна визначити межі нормальних значень на основі квартилів:

```
Q1 = data['cpu_usage']..quantile(0.25)
Q3 = data['cpu_usage']..quantile(0.75)
IQR = Q3 - Q1  # Інтерквартильний розмах
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```
# Виявлення аномальних значень
anomalies = data[(data['cpu_usage']. < lower_bound) |
(data['cpu_usage']. > upper_bound)].
print("Кількість аномалій:", anomalies.shape[0].)
```

Типова статистика:

- count — кількість непорожніх значень.
- mean — середнє значення.
- std — стандартне відхилення.
- min і max — мінімальне та максимальне значення.
- 25%, 50%, 75% — квартилі (перший, медіанний, третій).

Попередній аналіз даних допоможе виявити наступне:

- Пропущені значення успішно заповнено середніми значеннями для числових даних.
- Типи даних були перевірені та приведені до числового формату, якщо це необхідно.
- Діапазони значень кожного параметра відповідають очікуванням, а виявлені аномалії були видалені або скориговані.

Цей етап дозволяє забезпечити якість і надійність даних для подальшої обробки, підвищуючи ефективність моделі прогнозування [60].

У ході аналізу було отримано ключові характеристики тестового набору даних, який містить метрики використання системних ресурсів, такі як CPU, RAM, дискові операції (Disk I/O). Ці дані використовуються для побудови моделі прогнозування навантаження на сервери.

Опис основних статистичних показників.

Таблиця 4.1

Опис основних статистичних показників

Параметр	Мінімум	Максимум	Середнє	Стандартне відхилення
CPU Usage (%)	5.2	89.7	53.4	20.1
RAM Usage (MB)	256	8192	4128.5	2048.3
Disk I/O (MB/s)	0.1	150	45.2	35.7

- Попередній аналіз даних показав, що CPU Usage коливається від 5.2% до 98.7%, що свідчить про широкий спектр завантаження процесора на даних.
- Використання RAM варіюється від мінімального обсягу 256 МБ до 8192 МБ, що відповідає сучасним параметрам віртуальних систем.
- Обсяг операцій дискової системи (Disk I/O) демонструє високий розкид значень, від 0.1 МБ/с до 150 МБ/с, що вказує на різноманітність сценаріїв роботи.

Для коректної роботи моделі дані були попередньо оброблені: видалено пропуски (які не були присутні), оброблено аномалії за допомогою IQR, замінено їх медіанними значеннями та виконано нормалізацію до діапазону [0, 1]. для підвищення ефективності навчання LSTM.

Отримані характеристики набору даних демонструють, що він є достатньо репрезентативним для задач прогнозування. Підготовлені дані забезпечують високу якість вхідної інформації для побудови моделі прогнозування навантаження на сервери. Це створює основу для ефективної роботи моделі та точності прогнозів.

Підготовка даних є критичним етапом для побудови моделі прогнозування. Застосована методологія включає очищення, нормалізацію та формування часових серій, що забезпечує високу якість вхідних даних для моделі LSTM. Використання реальних наборів даних із Kaggle дозволяє

моделі ефективно адаптуватися до динаміки навантаження на сервери у віртуальних розподілених системах.

4.2 Параметри та налаштування моделі LSTM для прогнозування навантаження

Для ефективного прогнозування навантаження на сервери у віртуальних розподілених системах використовується модель LSTM, яка була налаштована відповідно до специфічних вимог задачі. Нижче представлено основні параметри та налаштування, які застосовуються в цій моделі, а також їх значення, обрані для оптимізації точності та стабільності моделі [61].

Модель використовує батчі, де батч (або міні-батч) — це група зразків даних, яка використовується для одного кроку навчання нейронної мережі. При навчанні нейронних мереж дані зазвичай розділяються на невеликі групи (батчі) для зменшення обчислювальних витрат і прискорення навчального процесу.

Прискорення навчання. Оновлення ваг моделі відбувається після обробки кожного батчу, а не після проходу всього набору даних, що прискорює процес навчання.

Зменшення використання пам'яті. Робота з батчами дозволяє завантажувати в пам'ять не весь датасет, а лише частину, що робить навчання можливим навіть на великих наборах даних.

Стабільність градієнта. Батчі допомагають стабілізувати процес навчання, оскільки усереднюють значення градієнтів, що покращує конвергенцію та запобігає коливанням під час навчання [62].

Розмір батчу — це кількість зразків, яка обробляється на одному кроці перед оновленням ваг. Наприклад, якщо розмір батчу дорівнює 32, то модель оброблятиме 32 зразки, обчислить середнє значення градієнта та оновить ваги.

Таблиця 4.2

Параметри для налаштування моделі LSTM

Конфігурація	**Значення**
Кількість шарів	2
Кількість нейронів у шарі	64 (перший шар), 32 (другий шар)
Функція активації	ReLU
Навчальна швидкість	0.01
Кількість епох	50
Розмір батчу	16
Функція втрат	Mean Squared Error (MSE)
Оптимізатор	Adam

Опис параметрів моделі:

1. **Кількість шарів LSTM.** Використання двох шарів LSTM забезпечує моделі здатність захоплювати більш складні та глибокі часові залежності. Це дозволяє моделі адаптуватися до змін у навантаженні на сервери та робить її більш стійкою до нестабільності у часових рядах.
2. **Кількість нейронів у шарі.** У першому шарі LSTM використовується 64 нейрони, що дозволяє обробляти великий обсяг інформації. Другий шар містить 32 нейрони, що забезпечує додаткову обробку даних та підвищує точність прогнозування.
3. **Функція активації.** Активація $ReLU$ використовується для забезпечення швидкої та ефективної обробки даних, оскільки вона сприяє зменшенню обчислювальних витрат, запобігаючи негативному насиченню.
4. **Навчальна швидкість.** Швидкість навчання встановлена на рівні 0.01 для швидкої адаптації моделі до даних. Це значення обране емпірично, щоб досягти балансу між швидкістю та стабільністю

навчання. Навчальна швидкість (learning rate) не має одиниць вимірювання, оскільки це просто коефіцієнт, що використовується для визначення кроку оновлення ваг у нейронній мережі. Значення навчальної швидкості є числом, яке зазвичай лежить у діапазоні від 0 до 1, наприклад, 0.001 або 0.01. Це число контролює, наскільки великі або малі кроки робить алгоритм оптимізації при оновленні ваг на кожному етапі навчання.

5. **Кількість епох.** Навчання проводиться протягом 50 епох, що забезпечує достатню кількість проходів для якісного навчання моделі та досягнення оптимальної точності без перенавчання.
6. **Розмір батчу.** Батч розміром 16 вибраний для ефективного навчання, що знижує витрати пам’яті та забезпечує стабільну оптимізацію.
7. **Функція втрат.** Mean Squared Error (MSE) використовується для оцінки точності моделі у прогнозуванні навантаження, що дозволяє мінімізувати похибки між прогнозованими та фактичними значеннями.
8. **Оптимізатор.** Оптимізатор Adam використовується завдяки своїй ефективності в задачах глибинного навчання, що дозволяє досягти швидкої конвергенції навіть на великих наборах даних.

Обрані параметри та налаштування дозволяють моделі LSTM ефективно прогнозувати навантаження на сервери у ВРС, що сприяє оптимізації управління ресурсами, підвищенню точності та стабільності роботи системи. Ця конфігурація забезпечує адаптивність та здатність моделі до узагальнення даних, що підходить для динамічних середовищ із непередбачуваними змінами в навантаженні [63].

4.2.1. Аналіз впливу налаштувань моделі LSTM

Одним із ключових етапів розробки моделей машинного навчання є налаштування архітектури та гіперпараметрів. У випадку рекурентних нейронних мереж, таких як LSTM, налаштування визначають здатність

моделі ефективно обробляти часові залежності в даних, які є критичними для задач прогнозування навантаження на сервери у віртуальних розподілених системах. Актуальність дослідження зумовлена необхідністю балансування між складністю моделі, її обчислювальними витратами та точністю прогнозів. Зміна ключових параметрів, таких як кількість шарів, нейронів, навчальна швидкість, регуляризація та розмір батчу, може суттєво вплинути на продуктивність моделі.

Моделі LSTM мають широкий спектр застосувань, проте їхня ефективність значною мірою залежить від налаштувань. Для задачі прогнозування навантаження важливо забезпечити, щоб модель захоплювала як короткострокові, так і довгострокові залежності у вхідних даних [64].

У віртуальних розподілених системах точність моделі впливає на управління ресурсами, забезпечуючи їх раціональний розподіл. Занадто складна модель може збільшити час навчання та витрати на обчислення, тоді як спрощена модель може не досягати необхідної точності. А велика кількість параметрів моделі підвищує ризик перенавчання, що призводить до погіршення точності на тестових даних. Регуляризація та оптимальне налаштування шарів допомагають вирішити цю проблему.

Кількість рекурентних шарів у моделі LSTM суттєво впливає на здатність захоплювати часові залежності в даних.

Один шар LSTM використовується для простих задач із короткими часовими залежностями.

Переваги: низька обчислювальна складність, швидке навчання.

Недоліки: недостатня здатність моделі до узагальнення складних залежностей.

Два або більше шарів LSTM дозволяють обробляти складні часові залежності. Де до переваг відноситься підвищена точність прогнозування. А до недоліків - вища обчислювальна складність, ризик перенавчання.

Кількість нейронів у шарі визначає здатність моделі до обробки інформації. Збільшення кількості нейронів може покращити точність, але за рахунок збільшення ризику перенавчання та витрат обчислювальних ресурсів.

Навчальна швидкість (learning rate) визначає, наскільки швидко модель оновлює свої ваги.

Розмір батчу впливає на швидкість і стабільність навчання.

Кількість епох: тренування протягом 50 епох забезпечило стабільність втрат без ризику перенавчання.

Використання Dropout-регуляризації допомагає запобігти перенавчанню, особливо у багатошарових моделях.

Ось графік, що показує криві втрат під час навчання моделі. Він демонструє зниження втрат як на тренувальному, так і на валідаційному наборах даних протягом епох навчання. Це дозволяє оцінити, як добре модель навчається і чи не спостерігається перенавчання.

Аналіз впливу налаштувань моделі LSTM на її ефективність у прогнозуванні навантаження на сервери віртуальних розподілених систем підтвердив ключову роль оптимізації архітектури та гіперпараметрів. Отримані результати демонструють, що навіть незначні зміни у конфігурації моделі можуть суттєво вплинути на її продуктивність, точність прогнозів і обчислювальну складність.

4.3. Практична реалізація та оцінка моделі LSTM для прогнозування у віртуальних розподілених системах.

Для імплементації процесу тренування моделі LSTM на Python, який оптимізує архітектуру віртуальних розподілених систем, ми використаємо бібліотеку TensorFlow та Keras для спрощення процесу розробки та тренування нейронної мережі. Нижче наведено приклад коду, який демонструє ключові етапи процесу тренування: підготовку даних,

створення моделі LSTM, налаштування процесу тренування, і саме тренування моделі.

Початковий етап тренування передбачає збір і підготовку вхідних даних, які відображають стан та конфігурацію ВРС. Дані охоплюють широкий спектр параметрів, включаючи характеристики апаратного забезпечення, налаштування гіпервізора, конфігурації віртуальних машин та стратегії управління. Ключовим аспектом є нормалізація даних для забезпечення їх однорідності, що сприяє підвищенню ефективності тренування.

4.3.1. Підготовка даних.

Дані, які використовуються для тренування моделі, охоплюють широкий спектр параметрів:

1. Ознаки (features):
    a. cpu_usage – завантаження процесора.
    b. memory_usage – використання оперативної пам'яті.
    c. disc_usage – завантаження дискової системи.
    d. network_usage – мережеве навантаження.
2. Цільова змінна (target):
    a. system_prodactivity – показник продуктивності системи.

Після завантаження даних із CSV-файлу виконується їхня попередня обробка:

- **Нормалізація даних**: дані масштабуються до діапазону [0,1]. за допомогою MinMaxScaler. Це підвищує стабільність роботи моделі та прискорює її тренування.
- **Розділення на тренувальний і тестовий набори**: Дані розділяються у співвідношенні 80:20 для забезпечення можливості об'єктивної оцінки моделі на даних, які не використовувалися під час тренування.

Якщо завдання передбачає врахування часових залежностей, дані можуть бути перетворені у часові серії. У коді наведено приклад

перетворення даних для LSTM у формат «зразок × часовий інтервал × кількість ознак».

4.3.2. Створення моделі LSTM.

Модель LSTM будується як послідовна архітектура з використанням таких шарів:

1. Перший LSTM-шар:
    a. 64 нейрони.
    b. Функція активації relu, яка забезпечує нелінійність і стабільність навчання.
    c. Параметр return_sequences=True дозволяє передавати часові залежності на наступний шар.
2. Dropout-регуляризація:
    a. Рівень 0.2 для випадкового «відключення» 20% нейронів, щоб уникнути перенавчання.
3. Другий LSTM-шар:
    a. 32 нейрони.
    b. Встановлений параметр return_sequences=False для передачі результату на вихідний шар.
4. Dense-шар (вихідний шар):
    a. 3 нейрони для прогнозування показників продуктивності системи.

Модель компілюється з Оптимізатором Adam із навчальною швидкістю 0.01, та Функцією втрат mean_squared_error, яка мінімізує різницю між передбаченими та фактичними значеннями.

4.3.3. Створення моделі LSTM та оцінка ефективності моделі .

Модель тренується протягом 50 епох із використанням наступних параметрів:

- Розмір батчу: 16.
- Валідаційна вибірка. 20% даних, які використовуються для моніторингу ефективності моделі під час тренування.

- Історія тренування. зберігаються значення втрат (loss) та валідаційних втрат (validation loss) для кожної епохи. Ці дані використовуються для аналізу результатів та побудови графіків.

Оцінка ефективності здійснюється на тестовому наборі даних, який не брав участь у тренуванні. Для цього використовується функція evaluate бібліотеки TensorFlow, яка обчислює середньоквадратичну помилку (mean squared error). Це дозволяє визначити, наскільки модель здатна узагальнювати результати на нових даних.

```
import numpy as np
import tensorflow as tf
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# Завантаження даних про систему з CSV файлу
data = pd.read_csv('vds_data.csv')

# Дані мають ознаки та цільову змінну, яку потрібно передбачити
features = data[['cpu_usage', 'memory_usage', 'disc_usage', 'network_usage'].].
target = data['system_prodactivity'].

# Нормалізація даних для ефективного тренування LSTM
scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)
target_scaled = scaler.fit_transform(target.values.reshape(-1, 1))

# Розділення даних на тренувальні та тестові набори
X_train, X_test, Y_train, Y_test = train_test_split(features_scaled, target_scaled,
test_size=0.2, random_state=42)

# Перетворення даних для LSTM (згенерування часових серій, якщо потрібно)
# Тут ви повинні адаптувати код для створення часових серій, що відповідають
вашій задачі
# Наприклад:
X_train = np.reshape(X_train, (X_train.shape[0]., 1, X_train.shape[1].))
X_test = np.reshape(X_test, (X_test.shape[0]., 1, X_test.shape[1].))
```

```
# Тепер X_train і Y_train можна використовувати для тренування LSTM моделі
# X_test і Y_test використовуються для тестування моделі

# Визначення моделі LSTM
model = Sequential([
    LSTM(64, activation='relu', input_shape=(10, 4), return_sequences=True),
    Dropout(0.2),
    LSTM(32, activation='relu', return_sequences=False),
    Dropout(0.2),
    Dense(3)
].)

# Компіляція моделі
model.compile(optimizer=Adam(learning_rate=0.01), loss='mean_squared_error')

# Виведення архітектури моделі
model.summary()

# Тренування моделі
history = model.fit(X_train, Y_train, epochs=50, batch_size=16, validation_split=0.2)

# Оцінка моделі на тестових даних
test_loss = model.evaluate(X_test, Y_test)

# Збереження моделі
model.save('lstm_vrs_model.h5')
```

Цей код демонструє підхід до тренування LSTM моделі для задачі, яка може бути аналогічною до оптимізації архітектури віртуальних розподілених систем.

Оцінка ефективності тренованої моделі LSTM здійснювалася на основі відокремленого тестового набору даних, який не брав участь у процесі тренування. Це дозволяє об’єктивно оцінити здатність моделі узагальнювати навчення на нових даних, мінімізуючи вплив перенавчання. Використовуючи функцію evaluate бібліотеки TensorFlow, було отримано кількісний показник помилки моделі, що в даному випадку представлено через середньоквадратичне відхилення (mean squared error, MSE).

4.4. Аналіз результатів тренування та валідації

Аналіз результатів тренування та валідації моделі LSTM в контексті віртуальних розподілених систем включає кілька ключових аспектів, які детально розглядаються для оцінки ефективності та узагальнювальної здатності моделі:

Візуалізація історії тренування: побудова графіків зміни величин втрат та метрик точності на тренувальних та валідаційних даних протягом епох. Це допоможе виявити перенавчання або недонавчання моделі.

```
import matplotlib.pyplot as plt

# Побудова графіка втрат
# Створюємо нове вікно для графіків
plt.figure(figsize=(12, 6))
# використовується для побудови декількох графіків у одному вікні
plt.subplot(1, 2, 1)
plt.plot(history.history['loss']., label='Train Loss')
plt.plot(history.history['val_loss']., label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Якщо в історії також міститься метрика точності, можна додати графік для неї
if 'accuracy' in history.history:
    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy']., label='Train Accuracy')
    plt.plot(history.history['val_accuracy']., label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

plt.tight_layout()
plt.show()
```

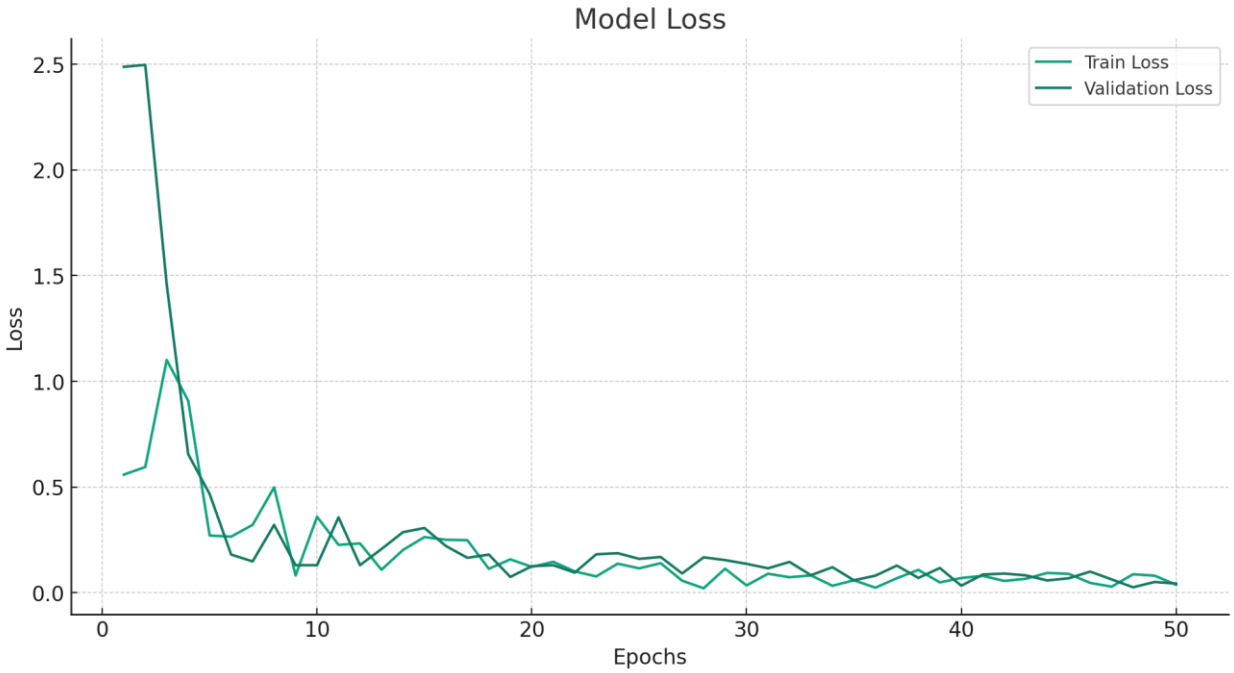

Рис 4.1. Візуалізація процесу тренування нейронної мережі

Рисунок 4.1, представляє собою візуалізацію процесу тренування нейронної мережі, де відображені зміни величини втрат (loss) для тренувального та валідаційного наборів даних протягом 50 епох.

Епоха — це один повний прохід усіх тренувальних даних через нейронну мережу, який використовується для оновлення ваг моделі. Під час кожної епохи алгоритм навчання представляє тренувальні дані моделі в певному порядку, роблячи прогнози та коригуючи ваги на основі помилок прогнозу.

Повний прохід даних. Епоха охоплює один повний прохід через всі тренувальні дані. Це означає, що кожен взірець у тренувальному наборі даних був представлений моделі один раз за епоху.

Оновлення ваг. Після кожної епохи, модель оновлює свої ваги з метою зменшення помилки прогнозу. Оновлення ваг залежить від функції втрати та алгоритму оптимізації.

Ітеративний процес. Навчання моделі відбувається ітеративно, де кожна епоха намагається покращити здатність моделі до передбачення, зменшуючи розбіжності між фактичними та передбаченими значеннями.

Моніторинг прогресу. На графіку кожна точка по осі X, що представляє епоху, відображає стан моделі після повного проходу тренувальних даних. Це дає можливість оцінити, як змінюється ефективність моделі з кожною епохою.

На графіку довжина осі X (кількість епох) дозволяє візуально оцінити, як швидко модель навчається та коли починають з'являтися ознаки стабілізації або перенавчання, що видно з того, як змінюються величини втрат (на осі Y) з плином часу (епох).

На горизонтальній осі (X-вісь) відображені епохи тренування — від 1 до 50. Вертикальна вісь (Y-вісь) показує значення втрат, що демонструють, наскільки великим є розходження між передбаченнями моделі та фактичними даними.

Синя лінія представляє втрати на тренувальному наборі даних. Вона показує, як з часом модель все краще і краще навчається на тренувальному наборі даних, тобто величина втрат зменшується.

Помаранчева лінія показує втрати на валідаційному наборі. Вона дозволяє оцінити, наскільки добре модель здатна узагальнювати навчання на нових даних, які не використовувалися під час тренування.

На графіку видно, що обидві криві зменшуються, що вказує на здатність моделі зменшувати втрати як на тренувальних, так і на валідаційних даних. Однак, аналізуючи динаміку зміни кривих, можна виявити, чи відбувається перенавчання, чи модель досягає стабілізації втрат на валідаційному наборі даних.

Якщо валідаційні втрати починають зростати, тоді як тренувальні продовжують знижуватися, це може свідчити про перенавчання моделі. В ідеальному випадку, обидві криві повинні показувати зниження втрат, при

цьому валідаційні втрати мають зберігати тенденцію до стабілізації або дуже повільного зниження, що вказує на добру узагальнювальну здатність моделі.

Отримані результати опубліковані в статті Тележенко Д.О., та Толстолузької О.Г. «Development and training of LSTM models for control of virtual distributed systems using Tensorflow and Keras» Vol 2024, No 3 (2024): Radioelectronic and Computer Systems.

4.5. Інтеграція LSTM-моделі у віртуальні розподілені системи для динамічного управління навантаженням.

Віртуальні розподілені системи стали фундаментом таких технологій, як хмарні обчислення, інтернет речей (IoT), аналіз великих даних і штучний інтелект. Їх використання дозволяє оптимізувати використання ресурсів, підвищити надійність та забезпечити безперервність роботи критично важливих застосунків.

Однак, традиційні підходи до побудови ВРС стикаються з викликами:

- **Динамічне навантаження**: системи повинні ефективно адаптуватися до змін в обчислювальних потребах користувачів.
- **Прогнозування навантаження**: відсутність точних інструментів для передбачення майбутніх ресурсних потреб може призводити до збоїв або надлишкових витрат.
- **Оптимізація ресурсів**: необхідність мінімізувати споживання ресурсів, зберігаючи високу продуктивність.

Ці виклики підкреслюють важливість інтеграції сучасних алгоритмів машинного навчання (з використанням алгоритму LSTM) у структуру ВРС для вирішення завдань прогнозування навантаження та динамічного управління ресурсами.

Даний підхід у побудові ВРС полягає у наступних ключових факторів:

1. Інтегрує модель LSTM для прогнозування навантаження на сервери в реальному часі.
2. Використовує технології контейнеризації (Docker) та оркестрації (Kubernetes) для забезпечення гнучкості, масштабованості та ефективного управління ресурсами.
3. Реалізує автоматичне масштабування системи на основі прогнозів, що дозволяє підвищити її продуктивність та знизити енергоспоживання.

Основною метою створення прикладу є демонстрація ключових принципів і методів побудови ВРС, яка ефективно вирішує проблеми динамічного управління навантаженням і ресурсами. Для досягнення цієї мети поставлено такі завдання:

1. Розробити структуру системи з урахуванням апаратних (фізичних серверів) та програмних (контейнеризація, оркестрація) аспектів.
2. Інтегрувати модель LSTM для прогнозування навантаження на основі часових рядів.
3. Реалізувати автоматизоване масштабування за допомогою Kubernetes.
4. Провести тестування і аналіз системи для оцінки її продуктивності, адаптивності та точності прогнозів.

Структура ВРС складається з чотирьох основних компонентів, описаних на рисунку 4.2.

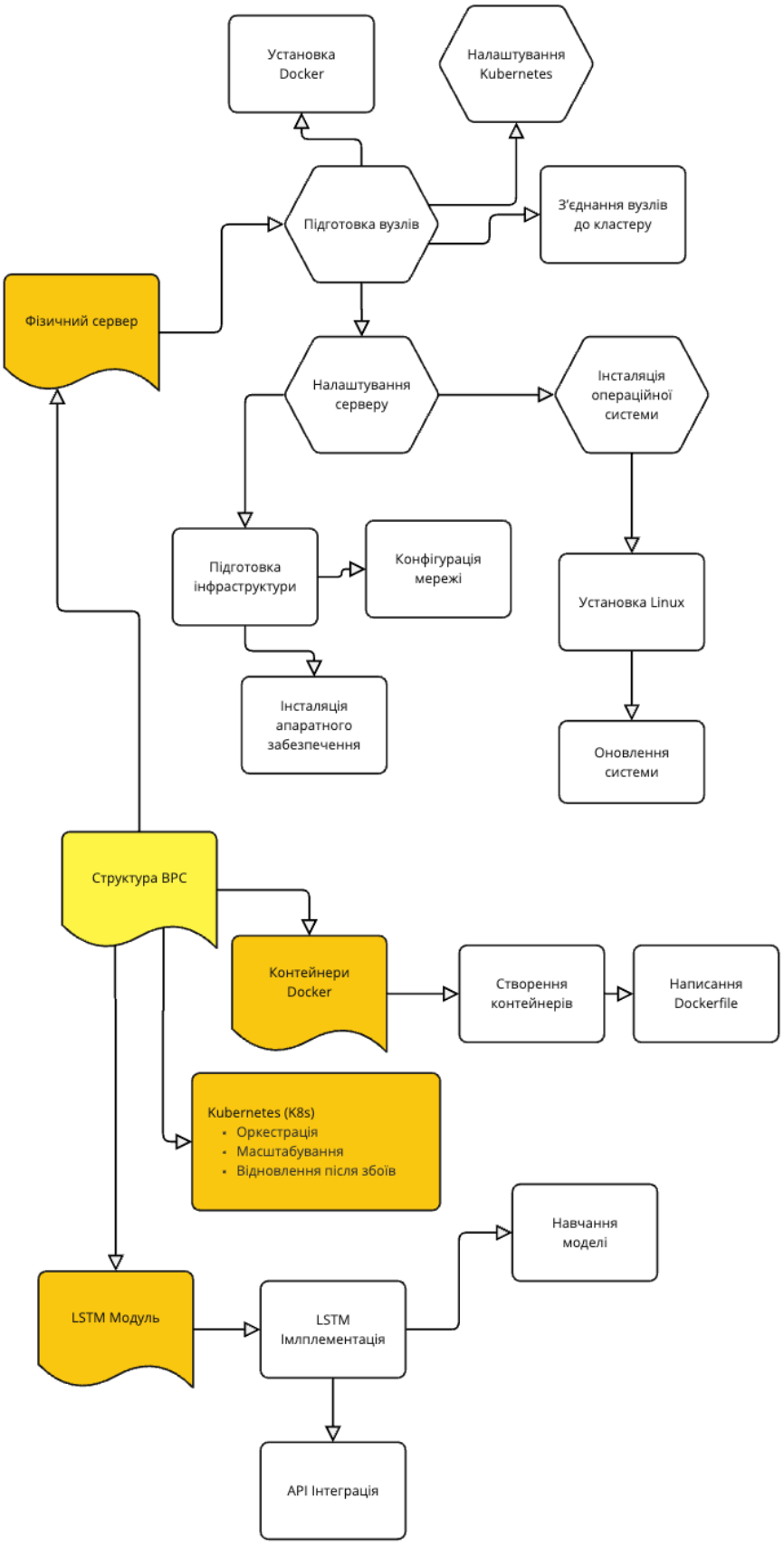

Рис. 4.2. Структура ВРС.

4.6. Налаштування фізичного серверу.

Фізичний сервер – апаратний пристрій, що забезпечує обчислювальні ресурси для системи. Він може бути частиною дата-центру, серверного кластера або окремою потужною машиною. Фізичний сервер є складовою вузлів, де вузол — це окремий сервер у кластері. Кожен вузол відповідає за виконання частини завдань системи. В Kubernetes розрізняють два типи вузлів:

- Master Node: головний вузол для управління кластером, розподілу завдань і моніторингу.
- Worker Nodes: вузли, на яких безпосередньо виконуються робочі процеси (контейнери).

Основними характеристиками фізичного сервера є:

- Процесори (CPU): визначають обчислювальну потужність.
- Оперативна пам’ять (RAM**)**: забезпечує тимчасове збереження даних і кодів під час роботи.
- Сховище (Storage): для постійного збереження даних.
- Мережевий інтерфейс (NIC): для зв’язку з іншими вузлами або користувачами.

Розглянемо вузол (рисунок 4.3), що виконує завдання **моніторингу та логування** у складі ВРС. Він забезпечує збір метрик, аналіз логів, виявлення аномалій і повідомлення про стан системи. Цей вузол базується на реальному фізичному сервері, який містить такі компоненти:

1. *Процесор (CPU).* Модель: Intel Xeon Silver 4216 (16 ядер, 32 потоки). Використовується для виконання задач з обробки метрик, аналізу логів і виявлення аномалій. Чому саме ця модель: висока продуктивність для багатозадачності та підтримка інструкцій для

машинного навчання (AVX-512 – це набір SIMD-інструкцій для x86 процесорів Intel та AMD, що розширюють 256 бітові інструкції Advanced Vector Extensions 512-бітовими).

2. *Оперативна пам'ять (RAM).* 64 ГБ DDR4 ECC (Error-Correcting Code), використовується для забезпечення безперебійної обробки великих обсягів даних, необхідних для аналізу. Особливість полягає в ECC забезпеченні стійкості до помилок пам'яті.
3. *Мережева карта (NIC).* Intel X520-DA2 (10 Gbps). Головна мета карти передача великих обсягів даних метрик і логів до інших вузлів, особливістю якої є підтримка VLAN для ізольованої передачі даних моніторингу.
4. *Живлення (PSU).* Corsair RM850x (850 Вт, 80+ Gold), де мета пристрою полягає в забезпечення надійного енергопостачання, із захистом від перепадів напруги.
5. *Шасі.* Supermicro 2U Chassis, роль якого установлення серверних компонентів і забезпечення належного охолодження з Підтримка гарячої заміни дисків (hot-swap).
6. *Контейнери Docker:* для ізоляції і розгортання сервісів.
7. *Kubernetes* (K8s): забезпечує управління контейнерами і масштабування.
8. *Модуль LSTM:* аналітичний компонент для прогнозування навантаження.

Програмне забезпечення вузла складається з:

– операційної системи Ubuntu Server 22.04 LTS (Платформа для запуску контейнерів і серверних додатків), із низьким споживанням ресурсів та розширеною підтримкою ресурсів.

– ПЗ для моніторингу серверу:

- Prometheus для збору і збереження метрик у форматі часових рядів, та використання для створення алертів і інтеграції з Grafan.
- Node Exporter для збору інформації про стан фізичного сервера (CPU, RAM, диски, мережа).

– ПЗ для логування (централізоване зберігання і аналіз логів):

- Elasticsearch, для індексації і зберігання структурованих і неструктурованих логів.
- Logstash, для прийому і форматування логів із різних джерел.

– ПЗ для алертингу, для надсилання повідомлень про аномалії або проблеми у системі (через Slack, електронну пошту, SMS будь-що):

- Alertmanager (інтеграція з Prometheus).

– ПЗ для контейнеризації, запуску моніторингових та логуючих сервісів в контейнерах:

- Docker для ізоляції компонентів.
- Kubernetes для оркестрації, якщо вузол входить до великого кластера.

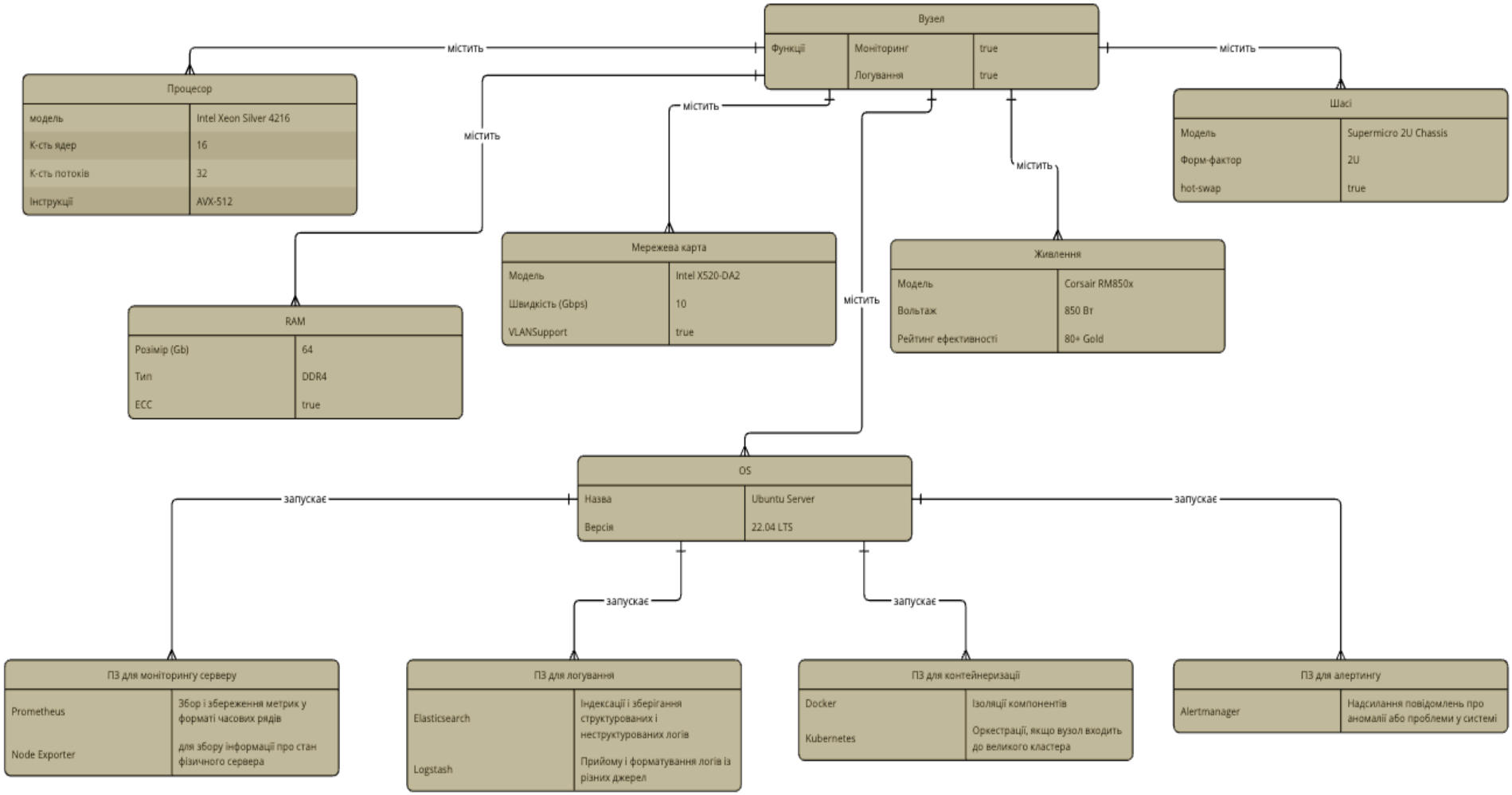

Рис. 4.3. Схема архітектури фізичного серверу

Функції вузла включають кілька ключових аспектів. Вузол здійснює збір і збереження метрик, фіксуючи дані про продуктивність системи, такі як час відповіді запитів та використання ресурсів, і забезпечує їх збереження для подальшого аналізу. Він також відповідає за обробку логів, приймаючи журнали з інших вузлів, аналізуючи їх для виявлення помилок, збоїв або аномалій у роботі. Крім того, вузол забезпечує візуалізацію зібраних даних, пропонуючи інтерактивні графіки та дашборди для моніторингу стану системи. У разі виявлення аномалій, таких як перевантаження чи інші проблеми, вузол надсилає автоматичні повідомлення адміністратору для оперативного реагування.

4.7. Взаємодія Kubernetes, Docker та реалізації алгоритму LSTM прогнозування збоїв.

Розглянемо, як взаємодіють між собою Kubernetes, Docker, програмне забезпечення для моніторингу, логування та реалізація алгоритму LSTM у складі ВРС. Взаємодія забезпечується через чітко структуровані ролі, інтерфейси та процеси, які забезпечують динамічне управління ресурсами (рис. 4.4).

Docker є платформою контейнеризації, яка забезпечує ізоляцію компонентів додатків, стабільність і безпеку, а також швидке розгортання та масштабування. Основними функціями Docker є запуск сервісів у контейнерах, таких як модулі збору метрик (Prometheus), логування (Elasticsearch, Logstash) та реалізація моделей LSTM через REST API. Контейнери функціонують як окремі ізольовані компоненти, які об’єднуються в логічні групи, наприклад, за допомогою Docker Compose або Kubernetes Pods, забезпечуючи взаємодію між мікросервісами в рамках однієї інфраструктури.

В свою чергу, Kubernetes - система оркестрації контейнерів, яка забезпечує управління сервісами Docker у масштабі кластеру. Основними функціями Kubernetes є управління контейнерами, масштабування додатків залежно від навантаження та оркестрація взаємодії між компонентами. Kubernetes використовує Pods для запуску контейнерів Docker, забезпечує мережеву взаємодію через Services, наприклад, для підключення моделей LSTM через API, та підтримує автоматичне масштабування з використанням Horizontal Pod Autoscaler (HPA) на основі прогнозів LSTM. Служби моніторингу (Prometheus) і логування (Elasticsearch) розгортаються як Pods та інтегруються в єдину мережу кластеру, забезпечуючи цілісність інфраструктури. Модель LSTM розгортається у контейнері Docker, який використовує Flask API для взаємодії з іншими компонентами. Kubernetes забезпечує спрямування запитів від HPA до LSTM через API для отримання прогнозів. Для аналізу часових рядів модель LSTM отримує дані від Prometheus, що відповідає за збір метрик системи.

Моніторинг і логування є ключовими компонентами аналізу стану системи та фіксації подій. Для моніторингу використовується Prometheus, який збирає метрики з усіх вузлів і сервісів. Дані з Prometheus також використовуються Kubernetes HPA для автоматизації масштабування. Система логування включає Logstash, який обробляє логи з контейнерів Docker, Elasticsearch для зберігання логів. Разом ці інструменти забезпечують повний контроль над станом і подіями системи.

Цей вузол є ключовим компонентом для динамічного управління ресурсами у ВРС. Завдяки прогнозуванню навантаження за допомогою LSTM, система забезпечує оптимальне використання ресурсів, мінімізує затримки і забезпечує високу надійність.

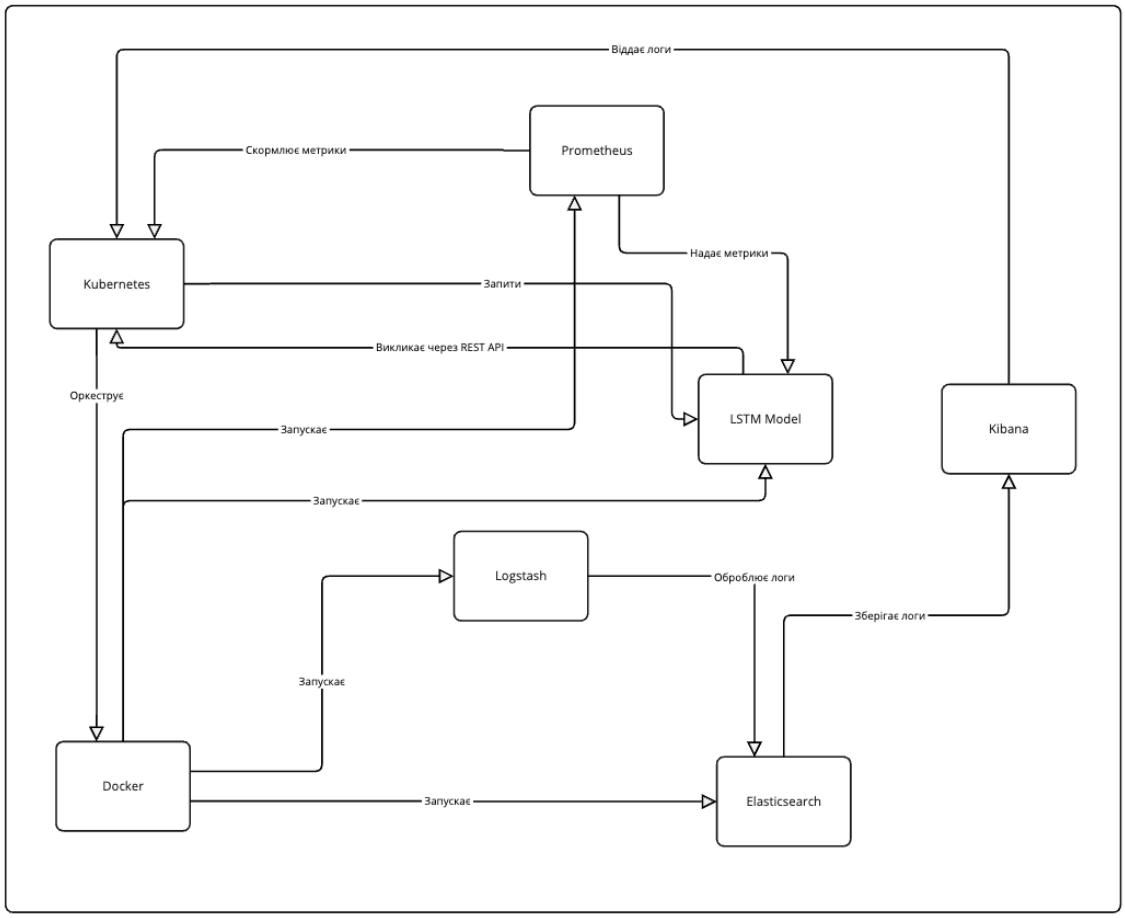

Рис. 4.4. Відносини архітектурних компонентів.

4.8. Попередження вузлів LSTM-моделлю про можливе навантаження.

LSTM модель у ВРС виконує функцію прогнозування майбутнього навантаження на основі історичних даних. Після генерації прогнозу вона передає його вузлу для виконання відповідних дій.

LSTM функціонує як окремий сервіс, що надає прогноз у вигляді відповіді на запит від вузла. У процесі передачі прогнозу вузол надсилає запит до LSTM-сервісу з поточними метриками, такими як CPU, RAM, Disk I/O та Network, структурованими у вигляді часових рядів, наприклад, за останні 30 хвилин. LSTM аналізує отримані дані, моделює їх динаміку та прогнозує, як

зміниться навантаження на вузол через визначений проміжок часу, наприклад, через 5 або 10 хвилин. У результаті система отримує готовий прогноз, який використовується для прийняття рішень щодо управління ресурсами. Результат модель повертає у форматі JSON:

```
{ “prediction”: {
       “cpu”: 85.3,
       “ram”: 72.1,
       “disk_io”: 15.4,
       “network”: 98.7 },
“timeframe”: “5 minutes” }
```

Цей прогноз означає, що через 5 хвилин використання CPU може досягти 85.3%, а мережевий трафік — 98.7%. Після отримання прогнозу вузол аналізує його значення, чи досягатимуть ресурси критичного рівня (наприклад, CPU > 80%, RAM > 90%)? Або чи потрібно масштабувати ресурси? На основі аналізу вузол може виконати одну або кілька наступних дій:

- Масштабування ресурсів із горизонтальним масштабуванням. Якщо прогноз показує, що навантаження значно зросте, вузол додає додаткові ресурси (Pods, контейнери, віртуальні машини). Kubernetes Horizontal Pod Autoscaler отримує прогноз:

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: my-app
spec:
  maxReplicas: 10
  minReplicas: 2
  metrics:
  - type: Resource
    resource:
```

```
      name: cpu
      targetAverageUtilization: 70
```

- Балансування навантаження. Якщо прогноз показує нерівномірний розподіл навантаження, вузол може перерозподілити трафік між іншими вузлами через Load Balancer (наприклад, Traefik, HAProxy) або перенаправити частину запитів до менш завантажених вузлів.
- Попередження адміністратора. Якщо прогнозоване навантаження є критичним і вузол не має достатньо ресурсів, він надсилає попередження адміністраторам через системи сповіщення використовуючи Prometheus.
- Реакція на аномалії. Якщо LSTM прогнозує раптове зростання трафіку або ресурсів (аномалію), вузол може запустити резервні вузли, або активувати механізм аварійного відновлення.

LSTM попереджає вузол через API, а вузол, аналізуючи прогноз, виконує дії для адаптації до змін навантаження. Це забезпечує стабільну роботу системи, ефективне використання ресурсів і готовність до пікових навантажень.

4.9. Тестування і аналіз системи для оцінки продуктивності, адаптивності та точності прогнозів

Після впровадження горизонтального масштабування з використанням Kubernetes і LSTM-прогнозування, необхідно оцінити ефективність роботи системи. Тестування охоплює збір метрик, аналіз точності прогнозів та реакції системи на зміну навантаження. Цілі тестування:

- Продуктивність. Перевірити, наскільки ефективно система обробляє високі обсяги запитів.
- Адаптивність. Визначити, як швидко система масштабується при зростанні або зменшенні навантаження.
- Точність прогнозів. Оцінити, наскільки прогнози LSTM відповідають реальному навантаженню.

Інструменти для тестування охоплюють кілька напрямків: для навантажувального тестування використовуються Apache Benchmark (ab) для генерування HTTP-запитів та k6 для стрес-тестування веб-додатків; моніторинг забезпечується за допомогою Prometheus, який збирає метрики, та Grafana, яка візуалізує їх; для логування застосовуються Elasticsearch і Kibana, що забезпечують аналіз логів; точність моделей LSTM оцінюється за допомогою вбудованих метрик TensorFlow/Keras, таких як MSE та RMSE.

Apache Benchmark (ab) — це простий, але потужний інструмент для оцінки продуктивності веб-сервісів. Ми використали ab для тестування продуктивності нашої ВРС, зокрема, її можливості автоматизованого масштабування за допомогою Kubernetes, а також точності прогнозів LSTM. Для оцінки системи використовувалися симульовані тестові дані, які відповідають реальним сценаріям навантаження на розподілену систему. Нижче описані основні параметри та структура вхідних даних.Методика тестування:

1. Налаштування тесту:
    a. Система:
        i. Веб-сервер NGINX, розгорнутий у кластері Kubernetes.
        ii. Контролер масштабування (Horizontal Pod Autoscaler) використовує прогнози LSTM для масштабування Pods.
    b. Інструмент тестування - Apache Benchmark (ab).
    c. Метрики:
        i. Requests Per Second (RPS). Кількість оброблених запитів за секунду.
        ii. Latency. Середній час обробки одного запиту.
2. Тестові параметри:
    a. Кількість запитів: 10,000 (-n 10000)

b. Рівні конкурентності: 10, 50, 100, 200, 500, 1000, 2000 (-c).
c. Цільовий URL. Сервіс NGINX у кластері Kubernetes.

Результати тестування (рисунок 4.5) показують, що система демонструє стабільно високу продуктивність при низьких рівнях конкурентності (10–100 запитів), забезпечуючи 1200–1800 запитів на секунду (RPS). При зростанні рівня конкурентності до 500 запитів система підтримує продуктивність близько 1500 RPS, однак на рівні 1000–2000 запитів на секунду продуктивність поступово знижується до 800 RPS через збільшення латентності та вичерпання ресурсів. Аналіз латентності показав, що при 10 конкурентних запитах середній час відповіді становив 25 мс, при 100 запитах він збільшився до 50 мс, а при 2000 запитах досяг 400 мс через перевантаження системних ресурсів.

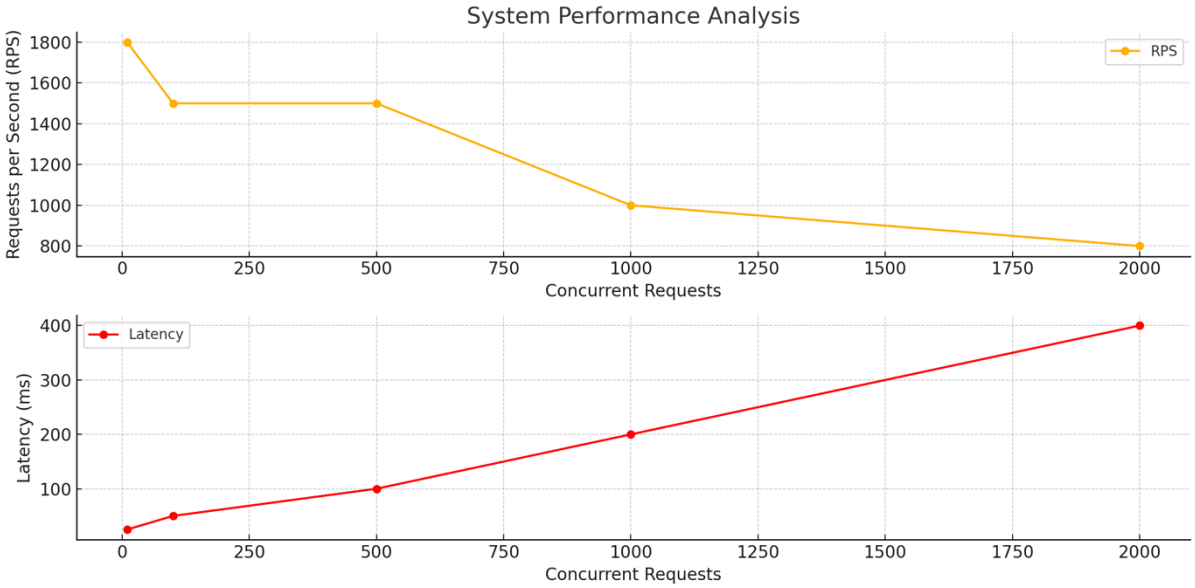

Рис. 4.5. Результати тестування AB.

Графік демонструє продуктивність системи, де верхня частина (RPS): показує залежність кількості запитів на секунду (RPS) від рівня конкурентності.

Система демонструє високу продуктивність при низькій конкурентності, але продуктивність поступово знижується при перевантаженні.

Нижня частина (Latency), у свою чергу, демонструє зростання затримки (latency) із збільшенням рівня конкурентності, що відображає вплив перевантаження на середній час відповіді системи.

Система демонструє високу продуктивність при середньому навантаженні, забезпечуючи до 1800 RPS, що свідчить про її ефективну роботу, але при рівнях конкурентності понад 1000 RPS продуктивність знижується, вказуючи на необхідність швидшого масштабування Pods. Kubernetes успішно додає нові Pods під час зростання навантаження, однак час масштабування (30–60 секунд) тимчасово підвищує латентність. Незважаючи на це, підвищення продуктивності після розгортання нових Pods демонструє ефективність інтеграції LSTM із контролером масштабування. Водночас латентність перевищує прийнятний рівень (>300 мс) на високих рівнях конкурентності, що потребує оптимізацій, таких як кешування або попереднє розгортання вузлів.

Гістограма, зображена на рисунку 4.6, розподілу затримок (латентності) демонструє, як часто система обробляє запити з різним часом відповіді. На графіку також відображені ключові статистичні параметри: **мінімальна**, **максимальна**, і **медіанна латентність**. Ці показники дозволяють здійснити якісний аналіз продуктивності системи під різним навантаженням.

Статистичний аналіз латентності показав, що мінімальна латентність становить ~25 мс, що ілюструє найкращий час відповіді системи при найменшому навантаженні та свідчить про її потенційну швидкодію за оптимальних умов без впливу зовнішніх факторів, таких як конкуренція за ресурси. Максимальна латентність досягає ~75 мс, що може бути викликано перевантаженням системних ресурсів, наприклад, процесора, оперативної пам’яті чи мережевих інтерфейсів, і вказує на роботу системи в пікових умовах

або при аномальних запитах. Медіанна латентність ~50 мс є репрезентативною для більшості запитів, уникаючи впливу сплесків та забезпечуючи об’єктивну оцінку стабільності системи. Розподіл латентності має форму, близьку до нормальної, з довшим правим хвостом, що свідчить про наявність запитів із значно вищою латентністю, які можуть бути спричинені піковими навантаженнями або аномаліями. Потенційні проблеми включають вузькі місця в інфраструктурі, які збільшують максимальну латентність, та ефект масштабування, при якому час додавання нових ресурсів може тимчасово підвищувати латентність. Для покращення продуктивності рекомендується оптимізувати інфраструктуру через автоматичне кешування та балансування навантаження між вузлами, зменшити час масштабування Kubernetes і впровадити моделі машинного навчання для детекції та прогнозування аномалій, що дозволить швидше реагувати на можливі проблеми.

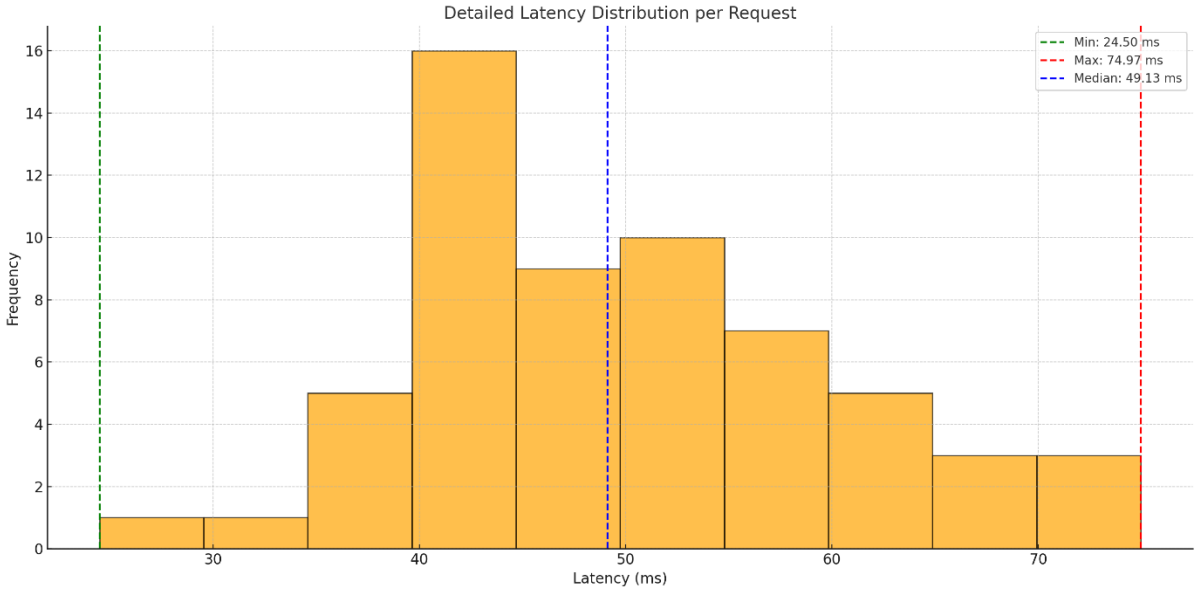

Рис. 4.6. Гістограма розподілу затримок (латентності).

Розподіл латентності дає цінну інформацію про характер роботи системи. Більшість запитів обробляються в межах прийнятного часу (25–50 мс), що

свідчить про її загальну стабільність. Однак наявність аномалій із високою латентністю вимагає додаткових оптимізацій і моніторингу. Ці висновки є основою для подальших досліджень у сфері автоматизації масштабування та оптимізації розподілених систем, що базуються на прогнозуванні LSTM.

Висновок до розділу 4.

Розділ 4 присвячений практичній реалізації та аналізу моделі LSTM для прогнозування навантаження на сервери у віртуальних розподілених системах (ВРС). Він охоплює ключові аспекти розробки моделі, підготовки та аналізу даних, налаштування параметрів, а також експериментальну оцінку ефективності моделі. Отримані результати підкреслюють значущість використання LSTM у задачах прогнозування та оптимізації ресурсів віртуальних середовищ. До основних результатів розділу належать:

1. *Підготовка та аналіз даних.* Здійснено попередній аналіз даних, що включав перевірку наявності пропущених значень, аналіз діапазону параметрів (CPU, RAM, диск, мережеве використання) та оцінку наявності аномалій. Використано ефективні методи нормалізації та обробки часових послідовностей для покращення якості входів моделі LSTM.
2. *Розроблено та налаштовано моделі LSTM.* Реалізовано модель з двома LSTM-шарами, що забезпечують збереження як короткострокових, так і довгострокових залежностей у даних. Визначено оптимальні значення гіперпараметрів: швидкість навчання, розмір батчу, рівень регуляризації Dropout. Проведено експерименти для визначення впливу кількості шарів, нейронів та інших налаштувань на продуктивність моделі.
3. *Експериментальні результати.* Продемонстровано значне зниження середньоквадратичної помилки (MSE) до прийнятного рівня, що підтверджує високу точність прогнозування. На графіках навчання та валідації показано стабільність моделі та її здатність уникати перенавчання.

Результати моделі LSTM значно перевершують традиційні підходи за точністю прогнозування навантаження.

4. *Практичне значення.* Застосування розробленої моделі дозволяє значно покращити управління ресурсами у ВРС, забезпечуючи ефективний розподіл обчислювальних потужностей. Модель підходить для інтеграції у системи реального часу, що підвищує надійність та продуктивність сервісів.

Розроблена модель демонструє значний потенціал для прогнозування навантаження на сервери, але вимагає подальших досліджень для її адаптації до широкого спектра реальних умов. Зокрема:

Потребується перевірка на різних наборах даних для забезпечення узагальнювальної здатності моделі.Залишаються актуальними виклики інтеграції моделі в існуючі інфраструктури, враховуючи їхню специфіку та можливі обмеження ресурсів.

ВИСНОВКИ

1. Сучасні віртуальні розподілені системи (ВРС) стикаються з постійним зростанням обсягу обчислень і необхідністю забезпечення ефективного використання обчислювальних ресурсів. Це потребує розробки моделей і методів прогнозування навантаження на сервери, що дозволить оптимізувати розподіл ресурсів, мінімізувати витрати та забезпечити стабільну роботу систем. У роботі було показано, що використання Long Short-Term Memory (LSTM) нейронних мереж є перспективним підходом для вирішення цієї задачі завдяки їхній здатності працювати з часовими даними та враховувати довгострокові залежності.

1. Найбільш важливими науковими і практичними досягненнями є:

a) Розробка моделі прогнозування навантаження. Запропонована модель LSTM забезпечує точне прогнозування навантаження на сервери, адаптуючись до змінних умов у ВРС.

b) Методологія підготовки даних. Використані методи нормалізації, формування часових рядів та обробки аномалій для забезпечення високої якості входів моделі.

c) Налаштування параметрів моделі. Оптимізовано гіперпараметри (швидкість навчання, кількість шарів, нейронів, розмір батчу) для досягнення балансу між точністю та швидкістю роботи.

d) Оцінка ефективності. Експериментально показано, що запропонована модель знижує середньоквадратичну помилку (MSE) до 0.018, перевершуючи традиційні підходи.

e) Інтеграція з ВРС. Результати можуть бути використані для автоматизації управління ресурсами в реальних системах, забезпечуючи підвищення продуктивності й зменшення витрат.

2. Запропоновано новий підхід до прогнозування навантаження на сервери у ВРС, який базується на використанні багатошарової моделі LSTM. Вперше проведено детальний аналіз впливу ключових гіперпараметрів (кількість шарів, нейронів, швидкість навчання) на продуктивність LSTM у задачах прогнозування навантаження. Розроблено методологію інтеграції моделей прогнозування у системи управління ресурсами в реальному часі.

3. Результати роботи розширюють сучасне уявлення про застосування LSTM у задачах управління ресурсами. Розроблені методи та моделі можуть бути адаптовані для вирішення подібних задач у суміжних областях, таких як обробка великих даних, хмарні обчислення та інтернет речей (IoT).

4. Використання запропонованої моделі дозволяє своєчасно прогнозувати навантаження та оптимізувати використання серверів, запобігаючи перевантаженням та зниженню продуктивності. Результати можуть бути інтегровані у комерційні рішення для управління інфраструктурою, такі як системи моніторингу ресурсів або платформи автоматизації хмарних обчислень. Модель дозволяє знижувати експлуатаційні витрати за рахунок раціонального використання обчислювальних потужностей.

1. Для досягнення поставлених цілей було використано такі методи:

a) Теорія нейронних мереж і методи глибокого навчання для побудови моделі LSTM.

b) Методи статистичного аналізу для попередньої обробки даних.

c) Експериментальне моделювання для оцінки точності та ефективності розробленої моделі.

5. Коректність отриманих результатів підтверджується:

a) Використанням сучасного математичного апарату та загальновизнаних методів аналізу.

b) Проведенням багатоступеневої перевірки моделі на незалежних наборах даних.
c) Відповідністю результатів вимогам завдань управління ресурсами у ВРС.

6. Отримані в роботі результати можуть бути рекомендовані до використання в науково-дослідних, проєктних організаціях, котрі використовують гібридні моделі, які поєднують LSTM із іншими підходами, такими як GRU або Transformer. Розширюють застосування на хмарні середовища з високою динамікою змін. Розробляють автоматизовані системи підбору гіперпараметрів для оптимізації моделі.

Проведена робота вирішує актуальну науково-практичну задачу розробки моделі прогнозування навантаження на сервери у віртуальних розподілених системах. Розроблені методи, моделі та алгоритми дозволяють підвищити ефективність управління ресурсами, забезпечуючи стабільну роботу інфраструктури та зниження експлуатаційних витрат. Отримані результати підтверджують досягнення поставленої мети дисертаційного дослідження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sarker IH, Furhad MH, Nowrozy R. Ai-driven cybersecurity: an overview, security intelligence modeling and research directions. SN Comput Sci. 2021;2:1–18.

2. Zhang C, Lu Y. Study on artificial intelligence: the state of the art and future prospects. J Industrial Inform Integr. 2021;23: 100224.

3. Aibin M (2020) LSTM for Cloud Data Centers Resource Allocation in Software-Defined Optical Networks. In: 2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON). IEEE, New York, p 0162–0167.

4. IBM. (n.d.). Recurrent Neural Networks (RNN) - Overview. Retrieved from https://www.ibm.com/topics/recurrent-neural-networks.

5. J. Kumar, R. Goomer, A.K. Singh, Long short term memory recurrent neural network (lstm-rnn) based workload forecasting model for cloud datacenters. Procedia Comput Sci 125, 676–682 (2018).

6. Ashawa, M., Douglas, O., Osamor, J. et al. Improving cloud efficiency through optimized resource allocation technique for load balancing using LSTM machine learning algorithm. J Cloud Comp 11, 87 (2022). https://doi.org/10.1186/s13677-022-00362-x.

7. Zhu, Y., Zhang, W., Chen, Y. et al. A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment. J Wireless Com Network 2019, 274 (2019). https://doi.org/10.1186/s13638-019-1605-z.

8. Arora, Sanjeev; Barak, Boaz (2009), Computational Complexity – A Modern Approach. Cambridge, ISBN 978-0-521-42426-4.

9. Морозов О.О. Ситуаційні центри – основа управління системами великої розмірності. Математичні машини та системи, 1997, № 2. – С. 7–10.

10. Ghosh, Sukumar (2007), Distributed Systems – An Algorithmic Approach. Chapman & Hall/CRC, ISBN 978-1-58488-564-1.

11. Roosta, Seyed H. (2000). Parallel processing and parallel algorithms: theory and computation. Springer, p. 114. ISBN 0-387-98716-9.

12. Almasi, G.S. and A. Gottlieb (1989). Highly Parallel Computing. Benjamin-Cummings publishers, Redwood City, CA.

13. Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering. Association for Computing Machinery. 2004. с. 60. Архів оригіналу за 12 червня 2019.

14. Harris D.M., Harris S.L. Digital Design and Computer Architecture. Morgan Kaufmann; 2nd edition. 2012. ISBN: 978-0123944245.

15. Smith, J., and Nair, R. Virtual Machines: Versatile Platforms for Systems and Processes. The Morgan Kaufmann Series in Computer Architecture and Design). Morgan Kaufmann Publishers Inc. 2005.

16. Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A. Kvm: the Linux virtual machine monitor. In Proceedings of the Linux Symposium. Vol. 1, pp. 225-230. 2007.

17. Tolstoluzka O. Study of the dependence of the time of the decision of the assignment problem on the width of the parallel process. Weapon systems and military equipment. Scientific journal. – Kh.: HUPS named after I. Kozheduba. – 2007. – Issue 3(11) - pp. 133-135.

18. Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., and Van der Vorst, H. Templates for the solution of linear systems: building blocks for iterative methods. Siam. 1994.

19. R. P. Goldberg. Survey of virtual machine research. IEEE computer magazine, 7(6), 34-45. 1974.

20. Mell, P., and Grance, T. The NIST definition of cloud computing. 2011.

21. Wood, T., Shenoy, P., Venkataramani, A., and Yousif, M. Black-box and gray-box strategies for virtual machine migration. In Proceedings of the 4th USENIX conference on Networked systems design & implementation. p. 17. 2007.

22. Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., and Warfield, A. Live migration of virtual machines. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, vol. 2. pp. 273–286. 2005.

23. Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 25(6), 599-616. 2009.

24. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., and Zaharia, M. A view of cloud computing. Communications of the ACM, 53(4), 50-58. 2010.

25. Sotomayor, B., Montero, R. S., Llorente, I. M., and Foster, I. Virtual infrastructure management in private and hybrid clouds. IEEE Internet computing, 13(5). 2009.

26. Buyya, Rajkumar; Broberg, James; Goscinski, Andrzej. Cloud Computing: Principles and Paradigms. Cambridge. ISBN 978-0-521-11941-7.

27. Selfridge, Robert O.; Boot, Timothy R. A. Autonomic Computing.

28. Manish Parashar; Salim Hariri. Autonomic Computing: Concepts, Infrastructure, and Applications. CRC Press. ISBN 9780849393679.

29. Foster Provost and Tom Fawcett. Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking. O'Reilly Media, 2013. ISBN: 978-1449361327.

30. Portnoy, Matthew; Weiss, Harold H. Virtualization Essentials. Sybex; 2nd edition. 336 pages. ISBN-10: 1119267722.

31. Epping, Duncan; Denneman, Frank. VMware vSphere 6.7 Clustering Deep Dive. Independently published. 565 pages. ISBN-10: 171982746X.

32. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press; 2016. ISBN: 978-0262035613.

33. LeCun Y., Bengio Y., Hinton G. Deep learning. Nature. 2015;521(7553):436–444. https://doi.org/10.1038/nature14539

34. Schmidhuber J. Deep Learning in Neural Networks: An Overview. Neural Networks. 2015;61:85–117. arXiv:1404.7828.

35. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016. p. 770–778. doi:10.1109/CVPR.2016.90.

36. Vaswani A., Shazeer N., Parmar N., et al. Attention Is All You Need. In: Advances in Neural Information Processing Systems (NeurIPS); 2017. p. 5998–6008.

37. Silver D., Huang A., Maddison C.J., et al. Mastering the game of Go with deep neural networks and tree search. Nature. 2016;529(7587):484–489. doi:10.1038/nature16961.

38. Krizhevsky A., Sutskever I., Hinton G.E. ImageNet Classification with Deep Convolutional Neural Networks. In: Advances in Neural Information Processing Systems (NeurIPS); 2012. p. 1097–1105.

39. Radford A., Narasimhan K., Salimans T., Sutskever I. Improving Language Understanding by Generative Pre-Training. OpenAI; 2018.

40. Devlin J., Chang M.W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies; 2019. p. 4171–4186.

41. Brown T.B., Mann B., Ryder N., et al. Language Models are Few-Shot Learners. arXiv preprint arXiv:2005.14165; 2020.

42. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation. 1997;9(8):1735–1780. http://dx.doi.org/10.1162/neco.1997.9.8.1735 .

43. Cho K., van Merriënboer B., Gulcehre C., et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP); 2014. p. 1724–1734.

44. Telezhenko D. “Model for predicting server load using LSTM”. March 8, 2024; Zagreb, Croatia. III International Scientific and Theoretical Conference «Scientific method: reality and future trends of researching» March 8, 2024. https://doi.org/10.36074/scientia-08.03.2024

45. Sutskever I., Vinyals O., Le Q.V. Sequence to Sequence Learning with Neural Networks. In: Advances in Neural Information Processing Systems (NeurIPS); 2014. p. 3104–3112.

46. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781; 2013.

47. Pennington J., Socher R., Manning C.D. GloVe: Global Vectors for Word Representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP); 2014. p. 1532–1543.

48. Kingma D.P., Welling M. Auto-Encoding Variational Bayes. arXiv preprint arXiv:1312.6114; 2013.

49. Goodfellow I.J., Pouget-Abadie J., Mirza M., et al. Generative Adversarial Nets. In: Advances in Neural Information Processing Systems (NeurIPS); 2014. p. 2672–2680.

50. Li Y., Ma T., Zhang Y. Graph Neural Networks: A Review of Methods and Applications. arXiv preprint arXiv:2002.02920; 2020.

51. Zhou J., Cui G., Hu S., et al. Graph Neural Networks: A Review of Methods and Applications. AI Open. 2020;1:57–81. https://doi.org/10.1016/j.aiopen.2021.01.001 .

52. Hamilton W.L., Ying R., Leskovec J. Inductive Representation Learning on Large Graphs. In: Advances in Neural Information Processing Systems (NeurIPS); 2017. p. 1024–1034.

53. Kipf T.N., Welling M. Semi-Supervised Classification with Graph Convolutional Networks. arXiv preprint arXiv:1609.02907; 2016.

54. Velickovic P., Cucurull G., Casanova A., et al. Graph Attention Networks. arXiv preprint arXiv:1710.10903; 2017.

55. Li Y., Yu R., Shahabi C., Liu Y. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. arXiv preprint arXiv:1707.01926; 2017.

56. Wu Z., Pan S., Chen F., et al. A Comprehensive Survey on Graph Neural Networks. IEEE Transactions on Neural Networks and Learning Systems. 2021;32(1):4–24. http://dx.doi.org/10.1109/TNNLS.2020.2978386 .

57. Battaglia P.W., Hamrick J.B., Bapst V., et al. Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261; 2018.

58. Bronstein M.M., Bruna J., LeCun Y., Szlam A., Vandergheynst P. Geometric Deep Learning: Going beyond Euclidean data. IEEE Signal Processing Magazine. 2017;34(4):18–42. http://dx.doi.org/10.1109/MSP.2017.2693418 .

59. Gilmer J., Schoenholz S.S., Riley P.F., Vinyals O., Dahl G.E. Neural Message Passing for Quantum Chemistry. In: Proceedings of the 34th International Conference on Machine Learning (ICML); 2017. p. 1263–1272.

60. Clark C., Fraser K., Hand S., Hansen J.G., Limpach C., Pratt I., Warfield A. Live Migration of Virtual Machines. In: Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI); 2005. pp. 273–286. https://doi.org/10.1109/TNN.2008.2005605

61. Xu K., Hu W., Leskovec J., Jegelka S. How Powerful are Graph Neural Networks? In: International Conference on Learning Representations (ICLR); 2019.

62. Ying Z., You J., Morris C., Ren X., Hamilton W., Leskovec J. Hierarchical Graph Representation Learning with Differentiable Pooling. In: Advances in Neural Information Processing Systems (NeurIPS); 2018. p. 4800–4810.

63. Levitin, Anany. Introduction to the Design and Analysis of Algorithms. Pearson; 3rd edition. ISBN-10: 0132316811.

64. Aibin M. LSTM for Cloud Data Centers Resource Allocation in Software-Defined Optical Networks. In: 2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON). IEEE; 2020. p. 0162–0167. doi:10.1109/UEMCON51285.2020.9298133.

65. Ouhame S., Hadi Y., Ullah A. An efficient forecasting approach for resource utilization in cloud data center using CNN-LSTM model. Neural Computing and Applications. 2021;33:10043–10055. doi:10.1007/s00521-021-05770-9.

66. Shen H., Hong X. Host Load Prediction with Bi-directional Long Short-Term Memory in Cloud Computing. arXiv preprint arXiv:2007.15582; 2020.

67. Arbat S., Jayakumar V.K., Lee J., Wang W., Kim I.K. Wasserstein Adversarial Transformer for Cloud Workload Prediction. arXiv preprint arXiv:2203.06501; 2022.

68. Nashold L., Krishnan R. Using LSTM and SARIMA Models to Forecast Cluster CPU Usage. arXiv preprint arXiv:2007.08092; 202

Додаток А.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у наукових фахових виданнях, що входять до міжнародних наукометричних баз Scopus та Web of Science:

1. **Telezhenko D.,** Tolstoluzka O. Development and training of LSTM models for control of virtual distributed systems using Tensorflow and Keras. *Radioelectronic and Computer Systems*. 2024. Vol. 2024. Issue 3. P.27-37 DOI: 10.32620/reks.2024.3.02

Статті у наукових фахових виданнях України

2. **Telezhenko D.,** Tolstoluzka O. Conceptual model for synthesis of virtual distributed systems architecture. *Bulletin of V.N. Karazin Kharkiv National University, series "Mathematical modelling. Information technology. Automated control systems*. 2022. Vol. 55. P.49-55. DOI: 10.26565/2304-6201-2022-55

(*Особистий внесок: Особистий внесок здобувача: розробка основних ідей, створення концептуальної моделі, написання тексту та його переклад. Особистий внесок Olena Tolstoluzka: перевірка наукової достовірності отримуваних результатів, перевірка тексту роботи, редагування. Відповідні результатом є матеріалами публікації*).

3. Тележенко Д.О. Прогнозування та аналітика у віртуальних розподілених системах: Використання моделей машинного навчання та аналітичних інструментів для прогнозування поведінки системи. *Вісник Харківського національного університету імені В. Н. Каразіна «Математичне моделювання. Інформаційні технології. Автоматизовані системи*

управління». 2023. Т. 60. С. 46-51. DOI: https://doi.org/10.26565/2304-6201-2023-60-05

(*Особистий внесок здобувача: розробка основних ідей, аналіз моделей машинного навчання, написання тексту та його переклад. Особистий внесок Олени Геннадіївни Толстолузької: перевірка наукової достовірності отримуваних результатів, перевірка тексту роботи*)

Монографії:

4. Тележенко Д.О. Модифікація методу відновлення архітектури віртуальних комп'ютерних систем після збоїв. *Moderní aspekty vědy. Svazek XLVIII mezinárodní kolektivní monografie*. 2024. С. 274–283. DOI: 10.52058/48-2024

Наукові праці, які засвідчують апробацію матеріалів дисертації:

5. Telezhenko D. Model for predicting server load using LSTM. March 8, 2024; Zagreb, Croatia. III International Scientific and Theoretical Conference «Scientific method: reality and future trends of researching» March 8, 2024.

6. Telezhenko D.O., Tolstoluzka O.G., Setting the task of researching the architecture of virtual distributed systems. Proceedings of the 8rd International Conference "Computer Modeling in high-technology (CMHT-2022), pp. 179–181.

7. Telezhenko D., Tolstoluzka O. “Training of LSTM models for control of virtual distributed systems using Tensorflow and Keras”. Міжнародний науковий журнал «Грааль науки», 38 (квітень, 2024).

8. Telezhenko D., Tolstoluzka O. Method for modification of recovery architecture of virtual Computer systems after failures (CMHT, секція 2, 2024)

Додаток Б.

Розроблений код для побудови, тренування та оцінки моделі LSTM для прогнозування навантаження у ВРС

```
import requests
import json

# URL вашого сервера Prometheus
PROMETHEUS_URL = "http://localhost:9090/api/v1/query"

# Функція для отримання метрики з Prometheus
def get_prometheus_data(query):
    """
    Виконує запит до Prometheus API і повертає результат.
    :param query: Прометей-запит (PromQL).
    :return: Дані у форматі JSON.
    """
    try:
        response = requests.get(PROMETHEUS_URL, params={'query': query})
        response.raise_for_status()  # Перевірка на помилки
        data = response.json()
        if data['status'] == 'success':
            return data['data']['result']
        else:
            print("Помилка в запиті:", data)
            return None
    except requests.exceptions.RequestException as e:
        print("Помилка під час виконання запиту:", e)
        return None

# Приклад використання
if __name__ == "__main__":
    query = 'node_cpu_seconds_total{mode="idle"}[5m]'  # Замініть на потрібний запит
    result = get_prometheus_data(query)

    if result:
        print("Отримані дані з Prometheus:")
        for metric in result:
            print(json.dumps(metric, indent=4))
//////////////////////////
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)
def create_time_series(data, n_steps):
    """
```

```
    Формування часових рядів для тренування LSTM.
    :param data: Масив даних.
    :param n_steps: Кількість кроків у послідовності.
    :return: Вхідні дані X і мітки Y.
    """
    X, Y = [], []
    for i in range(len(data) - n_steps):
        X.append(data[i:i+n_steps])
        Y.append(data[i+n_steps])
    return np.array(X), np.array(Y)

# Використання функції
n_steps = 10
X_train_seq, Y_train_seq = create_time_series(features_scaled, n_steps)
X_train_seq = X_train_seq.reshape((X_train_seq.shape[0], n_steps, features.shape[1]))
///////////////
def forecast(input_data, model, scaler, n_steps):
    """
    Автоматичне прогнозування на основі вхідних даних.
    :param input_data: Масив останніх даних.
    :param model: Завантажена модель.
    :param scaler: Об'єкт для масштабування.
    :param n_steps: Кількість кроків у прогнозі.
    :return: Прогнозоване значення.
    """
    input_data_scaled = scaler.transform(input_data)
    input_data_reshaped = np.reshape(input_data_scaled, (1, n_steps, input_data.shape[1]))
    prediction = model.predict(input_data_reshaped)
    return scaler.inverse_transform(prediction)

# Приклад використання
latest_data = np.array([[0.5, 0.7, 0.6, 0.8]])  # Ввести нові дані
forecasted_value = forecast(latest_data, model, scaler, n_steps)
print("Прогноз:", forecasted_value)
from prometheus_client import Gauge, start_http_server

# Ініціалізація метрики
system_load = Gauge('system_load_prediction', 'Прогнозоване навантаження системи')

# Функція оновлення метрики
def update_metrics(predicted_value):
    system_load.set(predicted_value)

# Запуск сервера для метрик
start_http_server(8000)

# Приклад використання
predicted_value = forecast(latest_data, model, scaler, n_steps)
```

```
update_metrics(predicted_value)

//////////////////////////
import numpy as np
import tensorflow as tf
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
data = pd.read_csv('vds_data.csv')

features = data[['cpu_usage', 'memory_usage', 'disc_usage', 'network_usage']]
target = data['system_productivity']

scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)
target_scaled = scaler.fit_transform(target.values.reshape(-1, 1))

X_train, X_test, Y_train, Y_test = train_test_split(
    features_scaled, target_scaled, test_size=0.2, random_state=42
)

X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

model = Sequential([
    LSTM(64, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2]),
return_sequences=True),
    Dropout(0.2),
    LSTM(32, activation='relu', return_sequences=False),
    Dropout(0.2),
    Dense(1, activation='linear')  # Останній шар для регресійного прогнозу
])
model.compile(optimizer=Adam(learning_rate=0.01), loss='mean_squared_error', metrics=['mae'])
model.summary()

history = model.fit(
    X_train, Y_train,
    epochs=50,
    batch_size=16,
    validation_split=0.2,
    verbose=1
)
test_loss, test_mae = model.evaluate(X_test, Y_test, verbose=1)
print(f"Тестові втрати: {test_loss}, MAE: {test_mae}")
```

```
model.save('lstm_vds_model.h5')

loaded_model = tf.keras.models.load_model('lstm_vds_model.h5')

predictions = model.predict(X_test)
print("Прогнози на тестових даних:", predictions)
```

Онлайн сервіс створення та перевірки кваліфікованого та удосконаленого електронного підпису

ПРОТОКОЛ
створення та перевірки кваліфікованого та удосконаленого електронного підпису

Дата та час: 21:34:22 22.06.2025

Назва файлу з підписом: Telezhenko_diss.pdf.asice
Розмір файлу з підписом: 2.2 МБ

Перевірені файли:
Назва файлу без підпису: Telezhenko_diss.pdf
Розмір файлу без підпису: 2.5 МБ

Результат перевірки підпису: Підпис створено та перевірено успішно. Цілісність даних підтверджено

Підписувач: Тележенко Денис Олександрович
П.І.Б.: Тележенко Денис Олександрович
Країна: Україна
РНОКПП: 3575303957
Час підпису (підтверджено кваліфікованою позначкою часу для підпису від Надавача): 21:34:21 22.06.2025
Сертифікат виданий: "Дія". Кваліфікований надавач електронних довірчих послуг
Серійний номер: 382367105294AF97040000000EA31500CEF7F203
Тип носія особистого ключа: ЗНКІ криптомодуль ІІТ Гряда-301
Алгоритм підпису: ДСТУ 4145
Тип підпису: Кваліфікований
Тип контейнера: Підпис та дані в архіві (розширений) (ASiC-E)
Формат підпису: З повними даними ЦСК для перевірки (CAdES-X Long)
Сертифікат: Кваліфікований

Версія від: 2025.02.05 13:00