

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ
СІКОРСЬКОГО»**

**Навчально-науковий інститут телекомунікаційних систем
Кафедра інформаційних технологій в телекомунікаціях**

До захисту допущено:

В.О. завідувача кафедрою

_____ **Валерій ПРАВИЛО**

«_____» _____ 2026 р.

ДИПЛОМНА РОБОТА

**на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційно-комунікаційні
технології» спеціальності 172 Телекомунікації та радіотехніка
на тему: Підхід до управління обчислювальними ресурсами і робочим
навантаженням у гетерогенних Kubernetes-середовищах на основі
штучного інтелекту**

Виконав: здобувач вищої освіти 4 курсу, групи ТІ-22

Ільченко Михайло Володимирович _____

Науковий керівник: доцент кафедри ІТТ НН ІТС,
кандидат технічних наук, с.н.с.

Алексеев Микола Олександрович _____

Рецензент:

Доцент кафедри телекомунікацій НН ІТС кандидат технічних наук, доцент
Міночкін Дмитро Анатолійович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Здобувач вищої освіти _____

Київ – 2026 року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

**Навчально-науковий інститут телекомунікаційних систем
Кафедра інформаційних технологій в телекомунікаціях**

Рівень вищої освіти – перший (бакалаврський)

Освітньо-професійна програма «Інформаційно-комунікаційні технології»
спеціальності 172 Телекомунікації та радіотехніка

ЗАТВЕДЖУЮ

В.О. завідувача кафедрою

_____ Валерій ПРАВИЛО

«_____» _____ 2026 р.

ЗАВДАННЯ

на дипломну роботу студенту

Ільченку Михайлу Володимировичу

1. Тема дипломної роботи: «Підхід до управління обчислювальними ресурсами і робочим навантаженням у гетерогенних Kubernetes середовищах на основі штучного інтелекту» і науковий керівник роботи доцент кафедри ІТТ НН ІТС, кандидат технічних наук, с.н.с. Алексєєв Микола Олександрович, затверджені наказом по університету від 26.05.2026р. № 1912-с.

2. Строк подання студентом дипломної роботи: 06.06.2026 р.

3. Об'єкт дослідження: є процес керування інформаційною інфраструктурою та розподіленням робочого навантаження кластерів Kubernetes в умовах гетерогенних систем з динамічною архітектурою.

4. Вихідні дані до роботи: Автономні, гетерогенні, фізичні Kubernetes-середовища, використання штучного інтелекту та проактивного автомасштабування.

5. Перелік завдань, які потрібно розробити:

1) здійснити аналітичний огляд наявних підходів до управління обчислювальними ресурсами і робочим навантаженням у гетерогенних Kubernetes середовищах на основі штучного інтелекту, провести систематизацію та порівняльний аналіз основних напрямків застосування штучного інтелекту в управлінні Kubernetes;

2) розробити підхід до управління гетерогенним динамічним середовищем Kubernetes кластерів для прогнозування навантаження, ризиків та проактивного предиктивного розподілу обчислювальних ресурсів;

3) провести експериментальну перевірку запропонованого підходу, оцінити практичну ефективність рішення.

6. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація – 28 слайдів, 3 таблиці, 8 рисунків

7. Дата видачі завдання: 15.12.2025 р.

Календарний план

з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Узгодження організаційних питань з науковим керівником	2.09.25 - 9.09.25	Виконано
2	Узгодження теми роботи та початок роботи над першим розділом	15.12.25 - 17.12.25	Виконано
3	Робота над першим розділом	7.01.26 - 3.03.26	Виконано
4	Робота над другим розділом.	3.03.26 - 17.03.26	Виконано
5	Робота над третім розділом.	17.03.26 - 24.03.26	Виконано
6	Висновки про ефективність рішення	24.03.26 - 31.03.26	Виконано
7	Оформлення дипломної роботи	5.04.26 - 1.06.26	Виконано
8	Підготовка презентації до захисту	25.05.26 - 12.05.26	Виконано

Здобувач вищої освіти _____ Михайло ІЛЬЧЕНКО

Науковий керівник роботи _____ Микола АЛЕКСЄЄВ

РЕФЕРАТ

Дипломна робота містить 83 сторінки тексту, 3 таблиці, 8 рисунків, а також використовує 64 літературних джерел посилань.

Актуальність теми:

Контейнеризовані застосунки та платформи їх оркестрації нині є одним із найпоширеніших підходів до розгортання прикладних сервісів у розподілених інформаційних системах. Одночасно з цим обчислювальні середовища, сформовані за принципом волонтерських обчислень, здатні забезпечувати значні обчислювальні ресурси, співставні з можливостями високопродуктивних суперкомп'ютерних систем. Використання таких середовищ для виконання контейнеризованих програм могло б мати очевидні переваги. Однак традиційні системи оркестрації, такі як Kubernetes, орієнтовані переважно на стабільні та однорідні інфраструктури. Натомість у гетерогенних системах з динамічною архітектурою (ГСДА), в яких реалізований підхід BYON (Bring Your Own Node), обчислювальні вузли можуть у будь-який момент змінювати доступність ресурсів, підключатися або виходити з кластера, що суттєво ускладнює, а інколи й унеможливорює ефективне застосування класичних засобів оркестрації.

На сьогодні існують рішення на основі технологій штучного інтелекту, спрямовані на підвищення передбачуваності та стабільності використання обчислювальних ресурсів шляхом проактивного масштабування кількості вузлів і реплік контейнеризованих сервісів. Проте більшість таких підходів орієнтовані на використання хмарних сервісів та однорідних високопродуктивних ресурсів, що зумовлює їх високу обчислювальну складність, економічну витратність і залежність від постійного інтернет-з'єднання. Через це вони лише частково придатні для автономних гетерогенних систем з динамічною архітектурою на основі BYON і особливо для систем з обчислювальними вузлами на основі автономних пристроїв із

відносно невеликою обчислювальною потужністю. Тож тема роботи, присвяченої розробці гібридного проактивного підходу до управління обчислювальними ресурсами і робочим навантаженням у гетерогенних Kubernetes середовищах з динамічною архітектурою і автономними вузлами на основі штучного інтелекту є актуальною.

Метою роботи є підвищення ефективності планування інфраструктури гетерогенних середовищ Kubernetes з динамічною архітектурою за рахунок застосування ШІ для прогнозування навантаження та проактивного розподілу обчислювальних ресурсів.

Об'єктом дослідження є процес керування інформаційною інфраструктурою та розподіленням робочого навантаження кластерів Kubernetes в умовах гетерогенних систем з динамічною архітектурою.

Предметом дослідження є створення підходу для підвищення ефективності планування інфраструктури за рахунок застосування можливостей ШІ для прогнозування навантаження та проактивного розподілу обчислювальних ресурсів.

Методами дослідження є аналіз наукових джерел, порівняльний аналіз існуючих підходів, проектування гібридної архітектури, програмна реалізація прототипу та експериментальне дослідження в тестовому гетерогенному середовищі.

Отримані результати. Проведено аналітичний огляд дистрибутивів Kubernetes за критерієм застосовності для кластерів BYON, а також основних напрямків застосування ШІ в управлінні Kubernetes. Обґрунтовано та розроблено гібридний інтелектуальний агент, що поєднує модель машинного навчання Prophet для прогнозування навантаження, евристичний розрахунок реплік та контекстний аналіз за допомогою локальної великої мовної моделі Llama 2:7B. Розгорнуто тестовий гетерогенний Kubernetes кластер та проведено експериментальну перевірку підходу в сценаріях поступового зростання, різкого пікового навантаження та зниження навантаження.

Підтверджено працездатність, проактивність та інтерпретованість запропонованого рішення в умовах обмежених ресурсів.

Апробація дослідження. Результати роботи було апробовано на 21-й Міжнародній науково-технічній конференції «Перспективи телекомунікацій» (2025, 2026 рр.) та опубліковано у вигляді наукової статті в журналі Information and Telecommunication Sciences (2025).

Ключові слова: ВОЛОНТЕРСЬКІ ОБЧИСЛЕННЯ, KUBERNETES, ГЕТЕРОГЕННІ СЕРЕДОВИЩА, ПРОАКТИВНЕ МАСШТАБУВАННЯ, PRORNET, LLM, ПРЕДИКТИВНЕ АВТОМАСШТАБУВАННЯ, ГІБРИДНИЙ ШІ-АГЕНТ, КРАЙОВІ ОБЧИСЛЕННЯ.

ABSTRACT

The Bachelor's thesis contains 83 pages, 3 tables, 8 figures, and references 64 literary sources.

The relevance of the topic.

Containerized applications and their orchestration platforms are currently one of the most widespread approaches to deploying application services in distributed information systems. At the same time, computing environments built on the principles of volunteer computing can provide significant computational resources comparable to those of high-performance supercomputer systems. Using such environments to run containerized workloads could offer clear advantages. However, traditional orchestration systems such as Kubernetes are designed primarily for stable and homogeneous infrastructures. In contrast, in heterogeneous systems with dynamic architecture (HSDA) that implement the BYON (Bring Your Own Node) approach, compute nodes can change resource availability, join, or leave the cluster at any moment. This significantly complicates, and sometimes renders impossible, the effective use of classical orchestration mechanisms. To date, artificial intelligence-based solutions exist that aim to improve the predictability and stability of resource usage through proactive scaling of nodes and container replicas. However, most of these approaches target cloud services and homogeneous high-performance resources. As a result, they are characterized by high computational complexity, significant economic costs, and dependence on a constant internet connection. Consequently, they are only partially suitable for autonomous heterogeneous systems with dynamic architecture based on BYON, especially those whose nodes are autonomous devices with relatively modest computing power. So the topic is highly relevant, as it deals with developing a hybrid proactive approach to managing computing resources and workload in heterogeneous Kubernetes environments with dynamic architecture and autonomous nodes, based on artificial intelligence.

The goal of the study is to increase the efficiency of infrastructure planning for heterogeneous Kubernetes environments with dynamic architecture by applying AI for load forecasting and proactive allocation of computing resources.

The object of research is the process of managing information infrastructure and distributing workload in Kubernetes clusters operating under heterogeneous systems with dynamic architecture.

The subject of research is the development of an approach to improve infrastructure planning efficiency through the use of AI capabilities for load forecasting and proactive distribution of computing resources.

Research methods include analysis of scientific sources, comparative analysis of existing approaches, design of a hybrid architecture, software implementation of a prototype, and experimental evaluation in a test heterogeneous environment.

Obtained results. An analytical review of Kubernetes distributions was conducted with respect to their applicability to BYON clusters, along with the main directions of AI application in Kubernetes management. A hybrid intelligent agent was substantiated and developed that combines the Prophet machine learning model for load forecasting, heuristic replica calculation, and contextual analysis using the local large language model Llama 2:7B. An experimental heterogeneous Kubernetes cluster was deployed, and the proposed approach was experimentally validated under scenarios of gradual load growth, sharp peak load, and load reduction. The operability, proactivity, and interpretability of the proposed solution under resource-constrained conditions were confirmed.

Presentation of research findings. The results of the work were presented at the 21st International Scientific and Technical Conference “Prospects of Telecommunications” (2025, 2026) and published as a scientific article in the journal *Information and Telecommunication Sciences* (2025).

Keywords: VOLUNTEER COMPUTING, KUBERNETES, HETEROGENEOUS ENVIRONMENTS, PROACTIVE SCALING, PROPHET,

LLM, PREDICTIVE AUTOSCALING, HYBRID AI-AGENT, EDGE
COMPUTING.

ЗМІСТ

ВСТУП	17
РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО УПРАВЛІННЯ РОБОЧИМ НАВАНТАЖЕННЯМ У ГЕТЕРОГЕННИХ KUBERNETES СЕРЕДОВИЩАХ ІЗ ЗАСТОСУВАННЯМ ШІ.....	22
1.1 Вступ до розділу	22
1.2 Аналіз наукових джерел у сфері ШІ-орієнтованого управління Kubernetes.....	22
1.3 Порівняльний аналіз легковагових Kubernetes- дистрибутивів	24
1.4 Машинне та глибоке навчання в управлінні ресурсами Kubernetes.....	26
1.5 Прогнозування навантаження за допомогою методів часових рядів і нейронних мереж.....	31
1.6 Використання великих мовних моделей (LLM) в управлінні Kubernetes.....	33
1.7 Порівняльний аналіз існуючих підходів та їх обмеження в гетерогенних середовищах, постановка задачі дослідження	37
1.8 Висновки до розділу 1	40
РОЗДІЛ 2. РОЗРОБКА ПІДХОДУ ДО УПРАВЛІННЯ ОБЧИСЛЮВАЛЬНИМИ РЕСУРСАМИ У ГЕТЕРОГЕННИХ KUBERNETES СЕРЕДОВИЩАХ НА ОСНОВІ ПРЕДИКТИВНОГО МАСШТАБУВАННЯ	42
2.1 Вступ до розділу	42
2.2 Теоретичне обґрунтування гібридного підходу до управління ресурсами.....	42
2.3 Аналіз підходу на основі предиктивного автоскейлінгу	43

2.4	Аналіз LLM-орієнтованого підходу до автоскейлінгу на основі підходу інтелектуального автомасштабувальника подів.....	47
2.5	Порівняльний аналіз розглянутих підходів	48
2.6	Теоретичне обґрунтування вибору компонентів гібридного підходу для малих гетерогенних дослідницьких кластерів.....	50
2.7	Висновки до розділу 2.....	53
РОЗДІЛ 3. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ГІБРИДНОГО ПІДХОДУ ДО ПРЕДИКТИВНОГО МАСШТАБУВАННЯ		55
3.1	Вступ до розділу	55
3.2	Створення тестового стенду та розгортання Kubernetes кластера	55
3.3	Підготовка та налаштування LLM компонента до інтеграції в гібридний підхід.....	57
3.4	Інтеграція ML в поєднанні з LLM в єдиний гібридний агент.....	58
3.5	Експериментальне дослідження ефективності підходу	61
3.5.1	Сценарій поступового зростання навантаження.....	63
3.5.2	Сценарій різкого зростання навантаження	65
3.5.3	Сценарій зниження навантаження	68
3.6	Аналіз результатів експериментального дослідження.....	69
3.7	Адаптація підходу до хмарного середовища AWS	71
3.8	Висновки до розділу 3.....	72
ЗАГАЛЬНІ ВИСНОВКИ.....		74
СПИСОК ЛІТЕРАТУРИ.....		76

СПИСОК ТЕРМІНІВ І СКОРОЧЕНЬ

AIOps	Artificial Intelligence for IT Operations Штучний інтелект для IT-операцій (автоматизація управління інфраструктурою)
ANN	Artificial Neural Networks Штучні нейронні мережі
ANN-HS	Horizontal Scaling based on Artificial Neural Networks Горизонтальне масштабування за допомогою штучних нейронних мереж
API	Application Programming Interface Інтерфейс програмування додатків
AWS	Amazon Web Services Хмарна платформа від компанії Amazon
Bridge	Мережевий міст (тип віртуального або фізичного мережевого інтерфейсу)
BYON	Bring Your Own Node «Принеси свій власний вузол» (концепція підключення власних потужностей до кластера)
Cloud-native	Хмарно-нативний (підхід до створення та запуску додатків у хмарних середовищах)
CNI	Container Network Interface Інтерфейс мережевих контейнерів (плагін для налаштування мережі в Kubernetes)
CPU	Central Processing Unit Центральний процесор
CRD	Custom Resource Definition Власне (кастомне) визначення ресурсу в Kubernetes
DeepCTS	Deep Learning-based Cloud Telemetry Scheduling сучасний науковий фреймворк та архітектурний підхід на основі глибокого навчання, створений для оптимізації планування, прогнозування та управління ресурсами у хмарних і периферійних інфраструктурах на основі аналізу великих

	масивів телеметричних даних
DQN	Deep Q-Network Глибока Q-мережа
DRL	Deep Reinforcement Learning Глибоке навчання з підкріпленням
EC-RL	Energy-Consumption Reinforcement Learning Навчання з підкріпленням з урахуванням енергоспоживання
Edge	концепція обчислень на межі (краю) мережі, ближче до джерела даних
EKS	Amazon Elastic Kubernetes Service Керований сервіс Kubernetes від AWS
Flannel	CNI-плагін для налаштування overlay-мережі в Kubernetes.
GenAI	Generative AI Генеративний ШІ (штучний інтелект)
GPU	Graphics Processing Unit Графічний процесор
GRU	Gated Recurrent Unit Рекурентна мережа з вентилями (тип рекурентних нейронних мереж, який добре захоплює довгострокові залежності та забезпечує вищу точність прогнозу)
HPA	Horizontal Pod Autoscaler Горизонтальний автомасштабувальник подів
Hypervisor	монітор віртуальних машин (Virtual Machine Monitor / VMM) програмне забезпечення, мікропрограма (firmware) або апаратний компонент, який створює, запускає та контролює віртуальні машини (VM)
Intnet	Internal Network Внутрішня мережа
IPA	Intelligent Pod Autoscaler Інтелектуальний автомасштабувальник подів
KAM	Kubernetes Autoscaling Mechanism Механізм автомасштабування Kubernetes
KEDA	Kubernetes Event-Driven Autoscaling

	Подієво-орієнтоване автомасштабування Kubernetes
KDDQN	K-Double Deep Q-Network К-подвійна глибока Q-мережа
	Kubeadm офіційний інструмент для швидкого розгортання та адміністрування кластерів Kubernetes
LLM	Large Language Model Велика мовна модель
LSTM	Long Short-Term Memory Довга короткочасна пам'ять (тип рекурентної нейронної мережі)
MetalLB	балансувальник навантаження для фізичних Kubernetes кластерів
ML	Machine Learning Машинне навчання
MLP	Multilayer Perceptron Багатошаровий персептрон
MSE	Mean Squared Error Середньоквадратична помилка
MTTR	Mean Time To Recovery Середній час відновлення
NGINX	Ingress Controller контролер вхідного трафіку на базі вебсервера NGINX для Kubernetes.
NLP	Natural Language Processing Обробка природної мови
ORACL	фреймворк для оптимізованого міркування при автоскейлінгу за допомогою ланцюга думок (Chain of Thought) LLM у мікросервісних архітектурах
Overlay-мережа	оверлейна або накладена віртуальна мережа, побудована поверх іншої мережі
QoS	Quality of Service Якість обслуговування
RAG	Retrieval-Augmented Generation Генерація з пошуком (метод розширення знань LLM за допомогою зовнішніх баз даних)
RBAC	Role-Based Access Control Контроль доступу на основі ролей
RCA	Root Cause Analysis

Аналіз кореневих причин (визначення першопричини збою/проблеми)

RL Reinforcement Learning Навчання з підкріпленням

SLO Service Level Objective Цільовий рівень обслуговування

SmartNIC Smart Network Interface Card

Програмована інтелектуальна мережева карта (апаратний прискорювач мережевих функцій)

Volunteer Computing Волонтерські обчислення

VPA Vertical Pod Autoscaler

Вертикальний автомасштабувальник подів

ВСТУП

У сучасних умовах стрімкого розвитку хмарних обчислень та мікросервісної архітектури контейнерна оркестрація стала ключовим елементом побудови надійної та масштабовної інфраструктури. Kubernetes (K8s), спочатку розроблений компанією Google, а нині підтримуваний Cloud Native Computing Foundation (CNCF), є фактичним стандартом оркестрації контейнерів. За даними на 2025 рік, понад 80% усіх виробничих контейнеризованих навантажень у світі працюють саме на базі Kubernetes [14]. Завдяки модульній архітектурі та розвиненій екосистемі, Kubernetes дозволяє організаціям ефективно керувати тисячами мікросервісів, забезпечуючи високу доступність, відмовостійкість і масштабованість. У той же час, стрімке зростання складності сучасних навантажень ставить перед Kubernetes нові виклики.

Використання волонтерських обчислень і BYON-середовищ, де вузли можуть динамічно з'являтися, зникати та змінювати доступність ресурсів, демонструють вагомий потенціал для створення потужних розподілених систем. Зокрема, залучення невикористаних обчислювальних ресурсів персональних пристроїв дозволяє досягати продуктивності, співставної з продуктивністю суперкомп'ютерів, за мінімальних витрат. [9].

Традиційні механізми планування та масштабування, засновані на жорстких правилах, такі як стандартні горизонтальний та вертикальний автомасштабувальники (Horizontal Pod Autoscaler (HPA), Vertical Pod Autoscaler) [35][36][37], добре працюють у відносно статичних і однорідних середовищах. Однак вони демонструють суттєві обмеження в умовах динамічних і гетерогенних навантажень. Зокрема, реактивний характер HPA та VPA призводить до того, що масштабування починається лише після виникнення перевантаження або простою, що спричиняє тимчасове зниження якості обслуговування (Quality of Service – QoS) і неефективне використання

ресурсів. Особливо гостро ці проблеми проявляються в гетерогенних обчислювальних середовищах (Гетерогенне Середовище з Динамічною Архітектурою – ГСДА). На відміну від класичних хмарних кластерів, де вузли мають порівняно однакові характеристики, у ГСДА обчислювальні ресурси представлені різноманітними пристроями: потужними хмарними серверами, робочими станціями, персональними комп'ютерами користувачів, локальними крайовими (edge) серверами. Такі вузли суттєво відрізняються за обчислювальною потужністю, обсягом оперативної пам'яті, характеристиками сховища, мережевими можливостями, наявністю апаратних прискорювачів (GPU, SmartNIC) та рівнем доступності. Що навіть більше критично, у ГСДА вузли можуть динамічно додаватися з певного пулу пристроїв та самотійно зникати, що робить середовище високо динамічним і непередбачуваним, а на додавання вузла з пулу готових пристроїв потребується значний додатковий час. У таких умовах традиційна парадигма оркестрації, побудована на припущенні відносної однорідності та стабільності ресурсів, перестає бути ефективною.

Інтеграцію методів штучного інтелекту в процеси планування та масштабування Kubernetes можна вважати такою, що має певні ознаки зміни парадигми у цій сфері, а саме, перехід від реактивного автомасштабування до проактивної предиктивної оптимізації. Саме такий перехід є ключовим для забезпечення ефективної роботи в сучасних гетерогенних динамічних обчислювальних середовищах.

Тож актуальність дослідження зумовлена великим обчислювальним потенціалом гетерогенних динамічних обчислювальних середовищ (ГСДА), у яких обчислювальні вузли представлені не лише традиційними хмарними серверами, а й персональними комп'ютерами користувачів, офісними комп'ютерами, робочими станціями та локальними серверами. Такі середовища характеризуються високою динамічністю навантаження, нестабільністю доступності вузлів та значною гетерогенністю ресурсів.

У таких умовах особливо актуальним стає перехід від реактивного до проактивного управління робочим навантаженням за допомогою методів штучного інтелекту, які дозволяють прогнозувати навантаження, адаптивно планувати розміщення подів та оптимізувати використання обмежених обчислювальних ресурсів.

З огляду на це, **метою роботи** є підвищення ефективності планування інфраструктури гетерогенних середовищ Kubernetes з динамічною архітектурою за рахунок застосування ШІ для прогнозування навантаження та проактивного розподілу обчислювальних ресурсів.

З поставленої мети випливає ряд задач:

1. Провести аналіз сучасних підходів до управління навантаженням у динамічних гетерогенних Kubernetes середовищах.
2. Створити підхід для управління обчислювальними ресурсами і робочим навантаженням у гетерогенних Kubernetes середовищах на основі ШІ.
3. Провести експериментальне дослідження розробленого підходу.

Також доцільно вирішити додаткові задачі – провести оглядовий аналіз дистрибутивів Kubernetes з точки зору вимогливості до апаратних характеристик обчислювальних вузлів кластерів, оскільки BYON-ноди можуть мати обмежені обчислювальні ресурси та/або залежати від споживання акумуляторної батареї. А для експериментального дослідження доцільно використовувати підходи GitOps, Infrastructure-as-Code, централізоване управління конфігураціями, оскільки це зменшить можливі фінансові втрати на створення та утримання експериментального стенду.

Об'єкт дослідження – процес керування інформаційною інфраструктурою та розподіленням робочого навантаження кластерів Kubernetes в умовах гетерогенних систем з динамічною архітектурою.

Предмет дослідження – створення підходу для підвищення ефективності планування інфраструктури за рахунок застосування

можливостей ШІ для прогнозування навантаження та проактивного розподілу обчислювальних ресурсів.

Наукова новизна дослідження полягає у розробці та обґрунтуванні гібридного проактивного підходу, придатного для застосування в умовах обмежених ресурсів і динамічної архітектури прогнозування навантаження, ризиків та проактивного предиктивного розподілу обчислювальних ресурсів.

Завдання дослідження: провести систематизацію та порівняльний аналіз чотирьох основних напрямків застосування штучного інтелекту в управлінні Kubernetes: класичного машинного навчання, навчання з підкріпленням (RL/DRL), прогнозування навантаження на основі часових рядів та великих мовних моделей (LLM).

Обґрунтувати доцільність застосування гібридного підходу для гетерогенних динамічних середовищ (ГСДА), який поєднує точність прогнозування навантаження методами машинного навчання, а саме прогнозуванням часових рядів моделлю Prophet [41] з контекстним аналізом і інтерпретацією рішень за допомогою великих мовних моделей.

Розробити архітектуру інтелектуального агента, що реалізує проактивне масштабування в умовах обмежених обчислювальних ресурсів і нестабільності вузлів, притаманних ГСДА. Запропонувати механізм забезпечення стабільності системи через поєднання детермінованого ML-компонента з LLM-аналізом.

Практична значущість роботи полягає в можливості застосування розробленого гібридного підходу для проактивного управління навантаження в реальних гетерогенних Kubernetes кластерах з обмеженими ресурсами (edge-системи, розподілені обчислення на базі персональних пристроїв, автономні системи тощо), де використання важких хмарних рішень є недоцільним або неможливим.

Апробація роботи. Основні результати дослідження апробовано на міжнародній науково-технічній конференції «Перспективи телекомунікацій» (ПТ-2025 та ПТ-2026) [2][3]. Зокрема, матеріали роботи представлено у

вигляді тез доповідей на конференціях ПТ-2025 і ПТ-2026, а також опубліковано у вигляді наукової статті в журналі *Information and Telecommunication Sciences* (2025) [28].

РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО УПРАВЛІННЯ РОБОЧИМ НАВАНТАЖЕННЯМ У ГЕТЕРОГЕННИХ KUBERNETES СЕРЕДОВИЩАХ ІЗ ЗАСТОСУВАННЯМ ШІ

1.1 Вступ до розділу

У першому розділі роботи проведено системний аналіз сучасних підходів до управління обчислювальними ресурсами та робочим навантаженням у гетерогенних Kubernetes-середовищах з динамічною архітектурою. Також приділено увагу особливостям волонтерських обчислень і BYON-середовищ, де вузли можуть динамічно з'являтися, зникати та змінювати доступність ресурсів. Проведено порівняльний аналіз основних напрямків застосування ШІ, та виявлено обмеження існуючих рішень.

1.2 Аналіз наукових джерел у сфері ШІ-орієнтованого управління Kubernetes

Отже, насамперед зазначимо важливі методологічні моменти щодо огляду літератури з питання. У нашому дослідженні чітко розрізняються три типи джерел наукової літератури, а саме: зрілі рецензовані публікації, препринти, кваліфікаційні роботи, наприклад, дисертація доктора філософії Омельченко В. В. [5], магістерська [40] та бакалаврська [6] роботи, оглядова стаття [20], а також актуальні наукові блоги, значимість яких полягає у наявності відкритих матеріалів, на відміну від закритих дослідницьких проєктів з обмеженим доступом. Таке розрізнення є особливо важливим у сфері цього дослідження, бо проблематика управління робочим навантаженням у Kubernetes є надзвичайно динамічною галуззю.

Корпус проаналізованих наукових праць налічує понад 30 джерел, які були опубліковані у період з 2023 по 2026 рік. Початкову вибірку

сформовано за допомогою ключових слів Kubernetes, workload management, LLM. Після цього, було вирішено доповнити її суміжними темами, які суттєво впливають на практику управління навантаженням, такі як: предиктивне масштабування [18] [64], мережева оптимізація [17] [23], самовідновлення (self-healing) [43], спільне розміщення подів (pod co-location) [46] та адаптивне планування [63] [59].

Пошук проводився в таких базах даних і платформах: SpringerLink, MDPI, IEEE Xplore, ACM Digital Library, arXiv, Semantic Scholar та Google Scholar. Корпус є певною мірою неоднорідним за типом публікацій і включає журнальні статті в журналах Electronics, Cluster Computing, IEEE Transactions on Cloud Computing, IEEE Access, CCF Transactions on High Performance Computing, матеріали міжнародних конференцій ACM Middleware, SpaCCS, SACML, одну магістерську роботу, кілька препринтів arXiv та два науково-практичні онлайн-ресурси – наукові блоги OneUpTime та Medium. Стосовно останніх двох зазначимо таке. Сучасна наука все частіше спирається на відкриті дослідницькі платформи. Багато вчених і розробників публікують аналітичні матеріали на платформі Medium, щоб зробити свої результати відкритими та зрозумілими для широкого кола фахівців, не чекаючи на традиційне довготривале рецензування. Такий підхід сприяє швидкому поширенню ідей, отриманню оперативного відгуку від наукового загалу, а також популяризації наукових результатів.

Кожне джерело перевірялося на бібліографічну коректність, репутацію майданчика публікації та зрілість заявлених результатів. У випадках, коли одна й та сама робота існує і як препринт, і як конференційна чи журнальна публікація, бібліографічний опис наводився за останнім варіантом, водночас електронні посилання наводилися на всі наявні онлайн-версії з огляду на забезпечення безперебійного доступу до відповідного джерела. Препринти наводилися переважно тоді, коли вони містили важливі деталі, відсутні в остаточній публікації.

У результаті аналізу літератури було виділено кілька тематичних напрямків, дотичних до проблематики нашої роботи, як-от: ШІ-орієнтоване планування; предиктивний автоскейлінг; самовідновлення та управління доступністю; мережево-орієнтована оптимізація; LLM-орієнтоване планування та оркестрація; LLM підтримані операції та безпека. Ці теми не є ізольованими і частково перекриваються. На практиці, ефективне управління робочим навантаженням досягається лише тоді, коли планування, масштабування, спостережуваність та операційне керування розглядаються як частини єдиного замкненого циклу. Тож метою літературного огляду було не лише описати існуючі підходи, а й окреслити технічний ландшафт галузі. Також на меті було виявити сильні та слабкі сторони кожного напрямку та виявити прогалини, які можна усунути за допомогою гібридних рішень, адаптованих до умов гетерогенних динамічних обчислювальних середовищ.

1.3 Порівняльний аналіз легковагових Kubernetes-дистрибутивів

У роботах, присвячених крайовому оркеструванню, зазначається, що повноцінний Kubernetes є ресурсомістким для крайового розгортання та малих фізичних кластерів, особливо в умовах ресурсообмеженої крайової інфраструктури [62]. Тому варто провести порівняльний аналіз сучасних Kubernetes дистрибутивів, орієнтованих на роботу в умовах обмежених обчислювальних ресурсів. Основною метою аналізу є визначення найбільш придатної платформи для побудови гетерогенних Kubernetes середовищ.

Класичний Kubernetes забезпечує найбільшу функціональність та масштабованість, однак має складну архітектуру та високі вимоги до ресурсів, що ускладнює його використання у ресурсообмежених крайових середовищах [53]. Minikube орієнтований переважно на локальну розробку та тестування Kubernetes середовищ і не є оптимальним для виробничого edge розгортання через обмежену масштабованість та відсутність орієнтації на багатнодові фізичні кластери [32]. MicroK8s забезпечує спрощене

розгортання Kubernetes та підтримує крайове розгортання, однак у більшості досліджень демонструє вище споживання ресурсів у порівнянні з K3s [53]. K0s демонструє високу продуктивність control plane та хорошу масштабованість, однак K3s має простішу архітектуру та ширше використовується у крайових обчислювальних середовищах [32]. KubeEdge та OpenYurt орієнтовані безпосередньо на крайовому оркеструванні та підтримують розгортання в edge середовищах, однак мають вищу складність конфігурації та більше споживання ресурсів у порівнянні з легковаговими дистрибутивами Kubernetes [62].

Таблиця 1.1 - Порівняння легковагових дистрибутивів Kubernetes

Параметр	K3s	MicroK8s	K0s	Minikube	KubeEdge	OpenYurt
Рекомендований RAM	1 ГБ	4 ГБ	1 ГБ	2 ГБ	70-256 МБ	1-2 ГБ
Рекомендований CPU	1 vCPU	2 vCPU	1 vCPU	2 vCPU	0.5–1 vCPU	2 vCPU
Формат розгортання	Один виконуваний файл	Snap-пакет	Один виконуваний файл	ВМ/контейнер	CloudCore + EdgeCore	Kubernetes + YurtHub
Тип рішення	Самостійний Kubernetes-дистрибутив	Самостійний Kubernetes-дистрибутив	Самостійний Kubernetes-дистрибутив	Локальне Kubernetes-середовище	Розширення Kubernetes для edge-вузлів	Розширення Kubernetes для cloud-edge середовищ
Основне призначення	Edge, IoT	Edge, локальні кластери	Production-кластери	Локальна розробка	Edge Computing	Cloud-Edge Computing

Як видно з таблиці 1.1, найменші вимоги до ресурсів мають K3s, K0s та KubeEdge. При цьому K3s і K0s є повноцінними Kubernetes-дистрибутивами, тоді як KubeEdge та OpenYurt виступають розширеннями Kubernetes для крайових середовищ.

Тому за результатами проведеного аналізу найбільш придатним рішенням для малих фізичних гетерогенних систем з динамічною архітектурою є K3s. Це обумовлено поєднанням низьких вимог до обчислювальних ресурсів, простоти розгортання, та орієнтації на edge та IoT середовища. На відміну від KubeEdge та OpenYurt, K3s не потребує додаткових компонентів для базового розгортання, а порівняно з MicroK8s характеризується нижчим споживанням ресурсів. Таким чином, K3s забезпечує найбільш збалансоване поєднання функціональності, простоти адміністрування та ефективності використання ресурсів для побудови малих фізичних Kubernetes-кластерів.

1.4 Машинне та глибоке навчання в управлінні ресурсами Kubernetes

Одним з найпоширеніших і найбільш досліджених напрямків застосування штучного інтелекту в управлінні Kubernetes є група методів машинного та глибокого навчання, до якої традиційно відносять як класичні моделі машинного навчання, так і сучасні архітектури глибоких нейронних мереж, зокрема алгоритми навчання з підкріпленням (Reinforcement Learning, – RL) та глибокого навчання з підкріпленням (Deep Reinforcement Learning – DRL). Ці підходи об'єднують спрямованість на аналіз часових рядів метрик споживання ресурсів, виявлення закономірностей у поведінці системи та прийняття адаптивних рішень щодо планування, розміщення, балансування навантаження й масштабування подів.

Роботи незалежного дослідника США Раві Чандри Тоти [56], дослідників Соні Матранги [40], Чжень Сюя та колег [61], Жілі Сяо та колег [60], дослідника Віреш Нунават [43], дослідників Жонгуй Цао та колег [13] та Цзян Цзянь та колег [30] демонструють, що поєднання хмарно-нативної (cloud-native) автоматизації з техніками ML/DL/RL дозволяє суттєво покращити ключові показники якості обслуговування. Так, огляд дослідника Раві Чандра Тота [56] узагальнює застосування машинного навчання та

навчання з підкріпленням для інтелектуального налаштування AWS EKS і показує, що предиктивне розподілення ресурсів, адаптивне масштабування та III-планування робочих навантажень дають змогу знизити затримки обробки запитів приблизно на 25%, одночасно підвищити стійкість системи та суттєво скоротити операційні й хмарні витрати. Досягнення забезпечується завдяки інтелектуальному управлінню чергами, проактивному перерозподілу CPU та пам'яті, а також оптимізації шляхів обробки запитів.

Особливо перспективним у цій групі є алгоритм DeepCTS (Deep Reinforcement Learning-based Cooperative Task Scheduling) Deep Learning-based Cloud Telemetry Scheduling - сучасний науковий фреймворк та архітектурний підхід на основі глибокого навчання, створений для оптимізації планування, прогнозування та управління ресурсами у хмарних і периферійних інфраструктурах на основі аналізу великих масивів телеметричних даних [60]. Алгоритм використовує глибоке навчання з підкріпленням для кооперативного планування завдань у Kubernetes-кластерах і дозволяє оптимізувати розміщення подів з урахуванням поточної завантаженості вузлів. Крім того, Жілі Сяо та колеги [60] продемонстрували, що правильна пріоритизація задач у планувальнику дає змогу скоротити час очікування відповіді на 35 %, оскільки критичні навантаження виконуються раніше й не затримуються в загальній черзі. Подібні результати демонструють Сінг Ванг з колегами [59], які запропонували оптимізацію стратегії планування задач у edge Kubernetes-кластерах на основі deep reinforcement learning. Їхній підхід враховує особливості обмежених ресурсів периферійних середовищ і показує значне покращення ефективності розподілу навантаження.

У напрямі класичного RL/DRL значний внесок зробила магістерська робота Соні Матранга [40]. Вона пропонує плагін на базі Deep Q-Network (DQN) – одного з найвідоміших алгоритмів глибокого навчання з підкріпленням, який інтегрується безпосередньо в фазу оцінювання вузлів планувальника Kubernetes. Алгоритм навчається кільком політикам

одночасно: балансуванню навантаження, оптимізації енергоспоживання та мінімізації затримок. Найкращі результати показала політика EC-RL (Energy-Consumption Reinforcement Learning), яка ефективно балансує між споживанням енергії та затримками, значно перевершуючи стандартний планувальник Kubernetes за цими показниками.

Подібний гібридний підхід DL+RL описують дослідник Чжень Сюй та колеги [61]: глибоке навчання використовується для моніторингу та прогнозування параметрів системи в реальному часі, а отримані прогнози та метрики передаються в RL-алгоритм, який динамічно перебудовує стратегію планування завдань з урахуванням поточної завантаженості та характеристик нових завдань. Цей метод забезпечує актуальне уявлення про стан кластера та адаптивне планування. Янг-Іон Юнг з колегами [63] запропонували адаптивне планування в Kubernetes на основі планування експериментів, що дозволяє оптимізувати як продуктивність, так і вартість роботи кластера, особливо в динамічних гетерогенних середовищах. Дослідник Гаутам Таді демонструє, що рекомендації VPA щодо коригування ресурсів часто неможливо застосувати через ліміт переривання подів, що спричиняє конфлікти та знижує загальну ефективність автосмасштабування [55].

Ще одним вагомим внеском у напрямок є алгоритм K-Double Deep Q-Network (KDDQN), запропонований дослідником Жунгуї Цао зі співавторами [13]. Це адаптивний алгоритм планування контейнерів на основі дво-мережевого глибокого навчання з підкріпленням, спеціально орієнтований на подолання обмежень гетерогенності апаратного забезпечення. На відміну від класичних евристичних планувальників, KDDQN одночасно вирішує чотири взаємопов'язані задачі: балансування навантаження між вузлами, підвищення загальної утилізації ресурсів, мінімізацію енергоспоживання обладнання та врахування специфічних вимог різних типів мікросервісів. Алгоритм було інтегровано безпосередньо в планувальник Kubernetes і протестовано експериментально. Результати показали зростання кількості розгорнутих контейнерів, кращу балансування

навантаження, вищу утилізацію ресурсів, зниження сумарного енергоспоживання кластера та підвищення надійності сервісів.

Важливим доповненням до цього напрямку є робота такого дослідника як Віреш Нунават [43], який пропонує повноцінну багато рівневу архітектуру проактивного самовідновлення (self-healing) Kubernetes. У цій архітектурі навчання з підкріпленням виступає ядром адаптивного прийняття рішень: RL-модель динамічно обирає оптимальні дії відновлення (перезапуск, міграція, масштабування, перерозподіл ресурсів) на основі зворотного зв'язку від середовища. Це дозволяє суттєво знизити середній час відновлення (Mean Time To Recovery, MTTR), час простою, енергоспоживання та середній час виявлення відмов (Mean Time To Detect, MTTD) завдяки ранньому виявленню відхилень ще до переходу системи в критичний стан. RL доповнюється NLP-моделлю для аналізу неструктурованих даних (логів, алертів, описів інцидентів), що дає змогу автоматично класифікувати збої, знаходити кореневі причини та семантично пов'язувати їх з телеметрією. LSTM-моделі формують предиктивний шар, який заздалегідь виявляє аномалії та деградацію за часовими рядами метрик. Завершує цикл цифрові двійники, у яких сценарії відновлення та оптимізації спочатку безпечно тестуються в симуляції, і лише після підтвердження мінімального MTTR та дотримання SLO застосовуються у продакшені. Важливим аспектом оптимізації роботи Kubernetes з AI/ML-навантаженнями є також удосконалення мережевого стеку. Аніла Гогінені [23] досліджує розвиток мережевої інфраструктури Kubernetes для високопродуктивних AI/ML робочих навантажень.

Ще більш практичним рішенням для ресурсообмежених гетерогенних середовищ стала робота Наваза Дхандала [18]. Автор пропонує використовувати модель Prophet для предиктивного автоскейлінгу. Порівняно з LSTM, GRU та ANN модель Prophet має низку суттєвих переваг у контексті гетерогенних динамічних середовищ (ГСДА). Це такі переваги: значно нижчі вимоги до обчислювальних ресурсів і обсягу історичних даних;

автоматичне виявлення трендів, сезонності та без складного ручного налаштування; швидке адаптивне перенавчання; висока інтерпретованість прогнозів; стабільна робота навіть при обмеженій кількості метрик і частій зміні складу вузлів кластера. Саме завдяки цим характеристикам модель Prophet від Наваза Дхандала [18] було обрано як основу ML-компонента нашого гібридного інтелектуального агента для проактивного управління навантаженням у гетерогенних середовищах з обмеженими ресурсами. У нашому дослідженні ми свідомо поєднуємо сильні сторони обох парадигм: ресурсоефективне прогнозування за допомогою Prophet (ML) та контекстний аналіз і пояснюваність рішень за допомогою великих мовних моделей (LLM).

Отже, підхід використовує бібліотеку Prophet для прогнозування CPU-використання на основі історичних даних Prometheus, з подальшим розрахунком необхідної кількості реплік подів. Для складніших патернів передбачено можливість використання LSTM-моделей. Реалізація включає регулярне навчання моделі, проактивне масштабування Deployment та інтеграцію з KEDA для реактивного резервного масштабування.

Загальним результатом застосування методів машинного та глибокого навчання в управлінні Kubernetes є значна оптимізація розміщення та балансування навантаження, проактивне самовідновлення, підвищення утилізації ресурсів, зниження енергоспоживання, зменшення затримок і покращення ключових метрик QoS.

Щодо придатності для малих гетерогенних динамічних середовищ (ГСДА) з обмеженими ресурсами, тут ситуація неоднозначна. Класичні ML-моделі та методи прогнозування часових рядів (які ми частково використовуємо в нашій гібридній реалізації) добре працюють на стабільних даних і можуть забезпечувати прийнятну точність прогнозування навантаження на поди. Однак у ГСДА вони мають певні обмеження: потребують значного обсягу історичних даних для навчання, можуть гірше адаптуватися до швидких змін інтенсивності навантаження на окремі поди та

потребують додаткового перенавчання при зміні патернів, що частково знижує оперативність.

Алгоритми RL/DRL (DeepCTS, DQN/EC-RL, KDDQN) демонструють високу адаптивність і здатність до оптимального планування подів у складних сценаріях, але в малих фізичних ГСДА проявляють помітніші обмеження: вони вимагають порівняно великих обчислювальних ресурсів для навчання, чутливі до якості та обсягу даних, потребують тривалого налаштування політики і можуть показувати знижену стабільність у середовищах з мінливим навантаженням на поди. У таких умовах повноцінне застосування RL/DRL може бути ресурсоємним і потребує обережного підходу.

1.5 Прогнозування навантаження за допомогою методів часових рядів і нейронних мереж

Важливим і практично орієнтованим напрямком застосування штучного інтелекту в управлінні Kubernetes є прогнозування навантаження за допомогою методів часових рядів і нейронних мереж. Цей підхід спрямований на передбачення майбутнього споживання ресурсів (CPU, пам'ять, кількість запитів) для проактивного масштабування подів ще до виникнення перевантаження.

Класичний метод Хольта–Вінтерса ефективно моделює рівень, тренд і сезонність, добре працює з періодичними піками навантаження та є обчислювально легким. У Kubernetes його застосовують для прогнозування CPU, пам'яті та кількості запитів, що дозволяє виконувати масштабування подів заздалегідь. Для складніших сценаріїв використовують GRU (Gated Recurrent Unit) – тип рекурентних нейронних мереж, який добре захоплює довгострокові залежності та забезпечує вищу точність прогнозу. Також застосовуються штучні нейронні мережі (ANN), зокрема багат шаровий перцептрон (MLP) та гібриди з LSTM (Long Short-Term Memory), які

дозволяють моделювати складні нелінійні залежності між різними метриками.

Практичні результати демонструють дослідники Хайбін Юань та Шеньчєнь Ляо [64], які поєднали метод Хольта–Вінтерса з GRU. Їхній предиктивний автоскейлер показав високу точність ($MSE = 0,00166$), скоротив час холодного старту на 1 годину 41 хвилину та зменшив коливання якості обслуговування на 83,3 %. Суттєвим доповненням стала робота професора Люсіляйде да Сілва та співавторів [16], які запропонували підхід ANN-HS (Horizontal Scaling based on Artificial Neural Networks). На відміну від реактивного HPA, ANN-HS використовує штучні нейронні мережі для прогнозування майбутньої навантаженості і динамічного коригування кількості реплік подів заздалегідь, забезпечуючи більш гнучке управління мікросервісами.

Загальним результатом цих методів є більш точне прогнозування навантаження, суттєве зменшення затримок масштабування подів, зниження часу холодного старту та підвищення загальної ефективності використання ресурсів кластера.

Щодо придатності для малих гетерогенних динамічних середовищ (ГСДА) з обмеженими ресурсами, то цей напрямок має помітніші обмеження порівняно з хмарними сценаріями. Методи часових рядів і нейронні мережі добре працюють на стабільних, періодичних даних, але в ГСДА їх ефективність суттєво знижується. По-перше, вони сильно залежать від великого обсягу якісних історичних даних – у динамічному середовищі з мінливим навантаженням на поди таких даних часто недостатньо або вони швидко стають неактуальними. По-друге, моделі погано адаптуються до швидких і непередбачуваних змін інтенсивності навантаження на окремі поди. По-третє, процес навчання та перенавчання моделей потребує додаткових обчислювальних ресурсів і часу, що в малих фізичних кластерах з обмеженою потужністю стає помітним навантаженням. По-четверте, при появі нових, неперіодичних патернів навантаження (які типові для ГСДА)

моделі демонструють обмежену гнучкість і потребують ручного втручання або частого перенавчання, що знижує ступінь проактивності та автономності системи.

1.6 Використання великих мовних моделей (LLM) в управлінні Kubernetes

Останнім часом одним з найбільш динамічних і швидко розвиваючих напрямків застосування штучного інтелекту в управлінні Kubernetes стало використання великих мовних моделей (LLM). На відміну від класичних ML-моделей і алгоритмів RL/DRL, LLM забезпечують не лише аналіз числових метрик, а й глибокий контекстний розбір неструктурованих даних (логів, алертів, описів інцидентів, маніфестів), пояснюваність рішень, роботу з природною мовою та можливість проактивного прийняття рішень на основі семантичного розуміння стану системи.

Ключові роботи в цьому напрямі включають дослідження дослідниці Кав'ї Говіндараджан з колегами [24], які представили GenAI-aware менеджер навантажень КАМ, спеціально оптимізований для робочих навантажень генеративного ШІ та тонкого налаштування великих мовних моделей. Менеджер працює паралельно з існуючими планувальниками Kubernetes і надає гнучкі можливості масштабування кількості GPU, виділених для подів, що дозволяє значно скоротити час очікування задач (до 3,5 раз) та покращити загальний час виконання (makespan) до 1,6 разів. Мосен Сейедказемі Ардебілі та Андреа Бартоліні [51] розробили KubeIntellect – модульну систему LLM-агентів для повного (end-to-end) управління Kubernetes. Система охоплює весь спектр операцій з подами: читання, створення, масштабування, управління життєвим циклом, RBAC та виконання команд. Центральний LLM-супервізор координує спеціалізовані агенти (логи, метрики, RBAC) підтримує та дозволяє взаємодію з кластером через природну мову. Оцінка на 200 реальних запитах показала 93 % успішності синтезу інструментів і 100 % надійність при використанні готових агентів.

Дослідник Сінью Бай з колегами [12] запропонували два підходи – LLMScheduler (онлайн-фреймворк, де LLM безпосередньо бере участь у виборі оптимального розміщення подів у реальному часі) та HybridScheduler (офлайн-ін'єкція інтелекту, коли LLM заздалегідь генерує стратегічні правила для швидкого традиційного планування). Експерименти показали підвищення успішності планування на 7% у складних умовах. Учений Хаою Бай з колегами [11] представили ORACL – фреймворк на основі LLM з ланцюжковим (chain-of-thought) міркуванням для уніфікованого визначення першопричин збоїв (RCA – root-cause analysis) та предиктивного автоскейлінгу подів. Система перетворює необроблені телеметричні дані в семантично багаті описи, діагностує причини деградації подів і видає інтерпретовані рекомендації з урахуванням механізмів гарантування безпеки (safety guardrails). Після донавчання на 80 ГБ даних модель підвищила точність RCA на 15 % і покращила QoS на 6 %.

Значний внесок у цьому напрямку зробили дослідники Артур Вітюї та Тсе Сун Чен [57], які розглядають LLM як допоміжний шар AIOps для проактивного управління IT-операціями. Моделі аналізують великі обсяги логів у реальному часі, виявляють кореляції, генерують рекомендації щодо масштабування подів і усунення проблем. Автори порівняли різні моделі (GPT-4o, Claude 3.5 Sonnet, Mistral Small 7B) і показали, що найкращий баланс точності, швидкості та економії токенів досягається при комбінації LLM з RAG і предиктивними ML-моделями.

Дослідники Лешек Слівко та Йоланта Мізеря-П'єтрашко [54] запропонували семантичний семантичний підхід на основі керування намірами (intent-driven) до планування навантаження подів через розширювач планувальника Kubernetes (Kubernetes scheduler extender), який інтерпретує природномовні анотації для м'якої афінності. Учений Стан Коен та співавтори [52] представили KubeGuard – фреймворк для автоматичного створення та уточнення найменш привілейованих маніфестів

(Roles, NetworkPolicies) для подів на основі аналізу логів виконання (runtime-логів) та ланцюжків промπτів (prompt-chaining).

Одним із найбільш перспективних напрямків застосування великих мовних моделей у Kubernetes є їх використання як інтелектуального шару для аналізу стану системи, інтерпретації метрик та генерації пояснюваних рекомендацій щодо управління ресурсами. Яскравим прикладом такого підходу є робота Шафіна Хасната [26] “IPA: Building AI driven Kubernetes Autoscaler”. Автор пропонує інтелектуальний автомасштабувальник подів Intelligent Pod Autoscaler (IPA) – LLM-орієнтований оператор, який кардинально відрізняється від традиційних засобів автомасштабування на основі жорстких правил (rule-based autoscalers). IPA працює за таким принципом:

1. Збирає метрики з Prometheus.
2. Аналізує логи та події кластера.
3. Передає зібраний контекст у велику мовну модель (у прикладі використовується Gemini).
4. Отримує від LLM обґрунтовані рекомендації щодо масштабування.
5. Застосовує їх через Kubernetes API.

Ключовою перевагою зазначеного вище підходу є пояснюваність рішень. На відміну від «чорної скриньки» у традиційних ML-моделях, LLM здатна не лише запропонувати кількість реплік, але й пояснити, чому саме таке рішення було прийнято, враховуючи контекст: поточний тренд навантаження, поведінку конкретних мікросервісів, можливі ризики тощо. Це робить систему більш прозорою та довіреною для операторів. І хоча робота опублікована на платформі Medium (де багато науковців публікуються саме для того, щоб зробити дослідження відкритим і зрозумілим для широкого кола фахівців без зволікань) і не є рецензованою науковою статтею, вона має надзвичайно високу практичну цінність. Автор наводить конкретну архітектуру, приклади промπτів, механізми обробки відповідей LLM. Саме ці практичні аспекти зробили підхід Шафіна Хасната [26] особливо корисним

при розробці нашого гібридного рішення. У межах нашого дослідження IPA розглядається як прототип LLM-компонента, який може враховувати більший контекст системи, відповідає за контекстний аналіз і семантичну інтерпретацію стану системи з можливістю корекції рішень, доповнюючи детермінований прогнозувальний ML-модуль.

Отже, Kubernetes оператор на базі LLM, який поєднує збір метрик з Prometheus з аналізом логів і подій. IPA працює як Custom Resource Definition (CRD), використовує Gemini (або іншу LLM) через агент для генерації рекомендацій щодо горизонтального та вертикального масштабування подів. Контролер кожен хвилину збирає дані, надсилає їх до LLM-агента і застосовує рекомендації через контур зворотного зв'язку (feedback loop). Підхід усуває потребу в жорстких порогах HPA/VPA і забезпечує контекстно-орієнтоване, динамічне масштабування.

Загальним результатом застосування великих мовних моделей є значне підвищення адаптивності систем керування Kubernetes, покращення якості прийняття рішень у реальному часі, ефективніша діагностика стану подів, проактивне масштабування, автоматична класифікація інцидентів, генерація пояснюваних рекомендацій та спрощення оркестрації навантажень.

Щодо придатності для малих гетерогенних динамічних середовищ (ГСДА) з обмеженими ресурсами цей напрямок демонструє як значний потенціал, так і суттєві виклики. LLM відмінно справляються з контекстним аналізом логів і подій, пов'язаних з подами, можуть працювати з меншими обсягами даних, ніж класичні моделі, і забезпечують високу пояснюваність рішень щодо масштабування. Однак у ГСДА їх пряме застосування обмежене. По-перше, навіть невеликі LLM вимагають помітних обчислювальних ресурсів для інференсу, що критично в середовищах з обмеженою потужністю. По-друге, присутні затримки обробки, які можуть впливати на оперативність масштабування подів. По-третє, зберігається ризик варіативності або некоректних рішень (галюцинацій), що знижує детермінованість – важливу властивість для стабільної роботи подів в

автономних системах. По-четверте, багато рішень все ще орієнтовані на потужні хмарні моделі або потребують значних обсягів токенів, що ускладнює їх використання в повністю автономних малих кластерах.

Таким чином, великі мовні моделі відкривають нові можливості для розумного та пояснюваного управління подами, але їх самостійне застосування в ГСДА обмежене ресурсами, затримками та ризиками.

1.7 Порівняльний аналіз існуючих підходів та їх обмеження в гетерогенних середовищах, постановка задачі дослідження

Проведений тематичний синтез дозволяє систематизувати сучасний стан досліджень управління робочим навантаженням у гетерогенних Kubernetes середовищах. Підкріплювальне навчання (RL/DRL) на сьогодні є найбільш зрілою парадигмою ШІ для адаптивного планування. Роботи таких дослідників, як Жунгуй Цао та колег [13], Жілі Сяо [60] та Цзян Цзяня та колег [30] демонструють, що RL ефективно справляється з багатокритеріальною оптимізацією в динамічних умовах. Однак його практичне застосування обмежене високою обчислювальною складністю навчання, складністю проектування функції винагороди, проблемами перенесення політики (policy transfer) та недостатньою пояснюваністю рішень.

Предиктивне масштабування має чіткіший і швидший шлях до практичного впровадження. Підходи на основі Prophet Наваза Дхандала [18], GRU від дослідників Хайбін Юань та Шеньчєня Ляо [64], ANN від професора Люсіляйде да Сілва [16] та LSTM від дослідника Віреш Нуавата [43] дозволяють суттєво зменшити затримки реакції порівняно з реактивними механізмами HPA/VPA. Водночас більшість таких робіт недостатньо враховують невизначеність прогнозу, міжресурсну інтерференцію та динамічну природу гетерогенних вузлів.

Великі мовні моделі (LLM) відкривають новий рівень семантичного розуміння системи. Аналіз літератури показує чітку закономірність: чим

критичнішою є дія з точки зору безпеки або чим вищою є чутливість до затримок, тим імовірніше LLM використовується в дорадчій або наглядовій ролі, а не як безпосередній виконавчий механізм (актуатор). Цей шаблон чітко простежується в ряді праць [12], дослідника Хаою Байя та колег [11], Шафіна Хасната [26], [52], [54], Артура Віті та Тсе Суна Чена [57]. Гібридні архітектури (LLM планувальник та гібридний планувальник – LLMScheduler та HybridScheduler, ORACL, IPA, KubeGuard) виглядають найбільш перспективними для промислового застосування.

Зазначимо, що перехід до проактивного управління на основі штучного інтелекту необхідний з таких міркувань. У хмарних середовищах нині вже є ефективні комерційні рішення, які успішно застосовують ШІ для прогнозування навантаження та автономного масштабування. Зокрема, Avesha Smart Scaler забезпечує предиктивне автомасштабування на базі генеративного ШІ. Це дозволяє значно знизити хмарні витрати (до 70 %) та час відповіді та оптимізувати поведінку мікросервісів [10]. Amazon Karpenter автоматично розгортає та налаштовує вузли в реальному часі, що зменшує вартість інфраструктури та операційне навантаження [8]. PredictKube від Dynix використовує ШІ для прогнозування навантаження на основі історичних даних на декілька годин уперед, тобто уможливорює проактивне масштабування [19]. Платформа на основі навчання з підкріпленням Sedai (автономна хмарна інфраструктура – self-driving cloud) автоматично оптимізує ресурси, підвищує продуктивність і запобігає простоям [50].

Однак усі перелічені рішення орієнтовані переважно на потужні хмарні інфраструктури, мають високу вартість, вимагають постійного підключення до інтернету та не адаптовані до умов обмежених ресурсів і динамічної гетерогенності (edge, BYON, дослідницькі кластери). Тобто такі рішення погано підходять для ГСДА через високу вартість, вимогу до постійного інтернет-доступу, значні обчислювальні ресурси та відсутність автономності.

Волонтерські обчислення (Volunteer Computing) демонструють вагомий потенціал для створення потужних розподілених систем, зокрема,

використання простоїв персональних пристроїв дозволяє отримувати обчислювальну потужність рівня суперкомп'ютерів за мінімальних витрат [9]. Так, проєкт Folding@Home [58] перевершив найпотужніші суперкомп'ютери, що існували на той момент часу. Однак, як підкреслюють сучасні дослідження [39], основні виклики таких систем – це гетерогенність вузлів та непередбачуваність доступності, що особливо актуально для середовищ типу BYON (Bring Your Own Node “використовуй свій власний вузол” / “підключи свій вузол”) [47]. Саме тому традиційні оркестратори (включаючи Kubernetes) потребують інтелектуальних проактивних механізмів управління ресурсами. Тож на часі є розробка легких, автономних інтелектуальних систем, адаптованих до обмежених ресурсів персональних комп'ютерів, робочих станцій та локальних серверів. Тобто систем, здатних працювати в умовах ГСДА.

Тема цієї роботи продовжує дослідження, що проводяться на кафедрі інформаційних технологій в телекомунікаціях НН ІТС КПІ ім. Ігоря Сікорського. Зокрема, у попередніх роботах Алексєєва М. О. та співавторів розглянуто питання управління життєвим циклом обчислювальних агентів з недетермінованим часом життя в Kubernetes-кластерах [1], а також запропоновано підходи до автомасштабування кластерів на основі персональних обчислювальних ресурсів користувачів [4]. Безпосереднім продовженням цих досліджень стали тези автора роботи [29], у яких викладено базові ідеї гібридного ШІ-підходу до управління ресурсами в гетерогенних середовищах.

Загалом, проаналізована література свідчить про певний методологічний дисбаланс. Так, деякі роботи оптимізують лише один шар системи (планування, масштабування або безпеку) та абстрагуються від інших. Також спостерігається брак реалістичних гетерогенних бенчмарків, недостатнє опрацювання невизначеності та обмежена увага до телекомунікаційних і edge-специфічних сценаріїв.

1.8 Висновки до розділу 1

Kubernetes кластери в гетерогенних динамічних середовищах з динамічною архітектурою (ГСДА) характеризуються високою мінливістю навантаження, нестабільністю обчислювальних вузлів та гетерогенністю ресурсів. Традиційні реактивні механізми масштабування (HRA та VPA) демонструють суттєві обмеження: вони активуються лише після виникнення перевантаження або простою, що призводить до затримок у реагуванні, зниження якості обслуговування та неефективного використання ресурсів. Отже, перехід до проактивного управління на основі штучного інтелекту є актуальним напрямом досліджень.

Наявні нині ефективні комерційні рішення орієнтовані переважно на потужні хмарні інфраструктури і тому є дорогавартісними, вимагають постійного підключення до мережі Інтернет і не адаптовані до умов обмежених ресурсів та динамічної гетерогенності (edge-системи, BYON, дослідницькі кластери). Таким чином, найбільш актуальною є розробка саме легких, автономних інтелектуальних систем, придатних для роботи в ГСДА.

Проведений у розділі порівняльний аналіз сучасних легковагових дистрибутивів Kubernetes, орієнтованих на роботу в умовах обмежених обчислювальних ресурсів показав, що найбільш придатним рішенням для малих фізичних гетерогенних середовищ є K3s завдяки низьким вимогам до ресурсів, простоті розгортання та орієнтації на крайові обчислення.

Проведений аналіз понад 30 сучасних наукових джерел дозволив систематизувати чотири основні напрями застосування штучного інтелекту в управлінні Kubernetes-середовищами. Серед проаналізованих рішень найбільший інтерес становлять роботи DeepCTS, KDDQN, ANN-HS, KubeIntellect та ORACL, проте вони недостатньо враховують обмеження малих ресурсообмежених Kubernetes-кластерів. Для задач управління ресурсами в ГСДА найбільш перспективним є поєднання предиктивного прогнозування навантаження та контекстного аналізу рішень. Як основу для

подальшої розробки було обрано підхід на основі моделі Prophet та підхід ІРА, що використовує великі мовні моделі для аналізу метрик і підтримки прийняття рішень щодо масштабування.

Тому поставлену у роботі задачу було конкретизовано як розроблення власного гібридного інтелектуального підходу до управління робочим навантаженням у гетерогенних динамічних Kubernetes середовищах, який поєднує ресурсоефективне, предиктивне прогнозування навантаження за допомогою методів часових рядів та контекстний аналіз і пояснюваність рішень за допомогою великих мовних моделей. Такий підхід має забезпечити проактивне, адаптивне та пояснюване управління ресурсами в умовах обмежених обчислювальних можливостей і динамічної архітектури ГСДА.

РОЗДІЛ 2 РОЗРОБКА ПІДХОДУ ДО УПРАВЛІННЯ ОБЧИСЛЮВАЛЬНИМИ РЕСУРСАМИ У ГЕТЕРОГЕННИХ KUBERNETES СЕРЕДОВИЩАХ НА ОСНОВІ ПРЕДИКТИВНОГО МАСШТАБУВАННЯ

2.1 Вступ до розділу

Другий розділ присвячено розробці гібридного підходу до проактивного управління ресурсами. Розглянуто предиктивний автоскейлінг на базі моделі Prophet, LLM-орієнтований підхід Intelligent Pod Autoscaler та їх порівняння. Також у цьому розділі обґрунтовано вибір компонентів гібридного агента, який поєднує низькоресурсне прогнозування часових рядів з контекстною інтерпретацією локальної великої мовної моделі, з акцентом на автономність, легковаговість та енергоефективність для малих гетерогенних середовищ.

2.2 Теоретичне обґрунтування гібридного підходу до управління ресурсами

Підхід Intelligent Pod Autoscaler (IPA), запропонований Шафіном Хаснатом [26], належить до напрямку великих мовних моделей (LLM). IPA реалізований у вигляді Kubernetes оператора та використовує LLM для надання рекомендацій щодо масштабування. В цьому підході для збору метрик також використовується Prometheus. Далі LLM аналізує контекст системи, визначає необхідну кількість реплік і параметри ресурсів, а IPA Controller автоматично застосовує рішення стосовно масштабування через Kubernetes API.

Основною перевагою IPA є прийняття рішення на основі контекстно-орієнтованому аналізу навантаження, який дозволяє враховувати складні взаємозв'язки між метриками, на відміну від класичних порогових підходів,

таких як НРА або VPA. Водночас використання LLM як основного механізму масштабування може призводити до недостатньої детермінованості рішень.

Найперспективнішим рішенням видається комплексна гібридна інтеграція цих підходів. Жоден з напрямків сам по собі не забезпечує достатнього балансу між точністю, адаптивністю, швидкістю та надійністю в умовах ГСДА. Саме тому в нашій роботі є доцільною розробка гібридного підходу, що поєднує прогнозування навантаження за допомогою моделі машинного навчання Prophet з контекстним аналізом за допомогою великих мовних моделей та можливістю корекції рішень. Така архітектура дозволить компенсувати недоліки та обмеження окремих підходів і поліпшити загальну стійкість системи в умовах фізичного гетерогенного середовища типу BYON, обмежених ресурсів та нестабільності інфраструктури.

2.3 Аналіз підходу на основі предиктивного автоскейлінгу

Задля вирішення проблеми часових затримок у стандартних механізмах автоскейлінгу у роботі Наваза Дхандала [18] пропонується використання підходу, заснованого на прогнозуванні навантаження. Його сутність полягає у побудові моделі, яка дозволяє оцінити майбутній стан системи на основі історичних даних, що уможливлює проактивне масштабування. Одним з ключових елементів зазначеного підходу є використання системи моніторингу Prometheus [49]. Ця платформа забезпечує збір метрик для формування часових рядів. Основною метрикою тут є використання подами CPU. Для її визначення використовується метрика `container_cpu_usage_seconds_total`. Оскільки ця метрика є накопичувальною, тобто її значення постійно зростає, для визначення реального навантаження використовується функція `rate()`, яка обчислює швидкість зміни метрики за заданий часовий інтервал. Інтервал `[5m]` означає, що обчислення виконується за останні 5 хвилин, що дозволяє отримати більш стабільне значення навантаження та зменшити вплив випадкових коливань. В програмній реалізації, це має наступний вигляд:

```

query = f'''
avg(rate(container_cpu_usage_seconds_total{{
namespace="{self.namespace}",
pod=~"{self.deployment_name}-.*"
}}[5m]))
'''

```

Отримані дані далі обробляються у програмному коді. Зокрема, значення розділяються на два списки: список часових міток та список відповідних значень CPU навантаження. А саме:

```

timestamps = []
values = []

for item in result[0]['values']:
    timestamps.append(datetime.datetime.fromtimestamp(item[0]))
    values.append(float(item[1]))

df = pd.DataFrame({
    'ds': timestamps,
    'y': values
})

```

У результаті виконання цього коду формується набір значень, у якому кожному моменту часу відповідає певне значення використання CPU. Таким чином, створюється впорядкована послідовність даних, яка відображає зміну навантаження у часі. На основі цих даних формується таблиця `df`, яка є стандартним форматом для моделі прогнозування Prophet [27]. Стовпець `ds` містить часові мітки, а стовпець `y` – значення метрики. У такому вигляді дані передаються через виклик функції `model.fit(df)` в модель для подальшого аналізу та побудови прогнозу навантаження.

Модель Prophet представляє часовий ряд у вигляді адитивної моделі. У програмній реалізації налаштування параметрів прогнозування відбувається у такий спосіб:

```

model = Prophet(

```

```

daily_seasonality=True,
weekly_seasonality=True,
yearly_seasonality=False,
changepoint_prior_scale=0.05
)
model.fit(df)

```

Під час навчання модель аналізує історичні дані та виявляє закономірності у зміні навантаження, зокрема, загальний тренд і періодичні коливання. На основі цього вона будує математичну модель, яка дозволяє прогнозувати майбутні значення.

Модель враховує як загальну тенденцію зміни навантаження з часом, так і регулярні повторювані патерни. Це дозволяє враховувати повторювані зміни навантаження. Параметр `daily_seasonality` дозволяє моделі враховувати добові циклічні зміни навантаження, відповідно `weekly_seasonality` забезпечує врахування тижневих патернів зміни навантаження, а `yearly_seasonality`, врахування річної сезонності. Параметр `changepoint_prior_scale` визначає гнучкість моделі при адаптації до змін тренду.

Після формування таблиці `df` модель Prophet виконує аналіз часового ряду, який містить значення навантаження у різні моменти часу. Математично модель Prophet описує часовий ряд як суму компонентів:

$$y(t)=g(t)+s(t)+\varepsilon(t), \text{ де}$$

$g(t)$ - тренд,

$s(t)$ - сезонний складник,

$\varepsilon(t)$ - випадкова похибка.

На основі отриманого прогнозу визначається пікове значення навантаження: $\text{peak_cpu}=\max(y(t))$

Це використовується як ключовий параметр для прийняття рішення щодо масштабування. Використання максимального значення пояснюється

необхідністю підготовки системи до найвищого можливого навантаження в прогнозованому періоді.

Для переходу від прогнозу до розрахунку рішення, у роботі застосовується евристична формула, яка базується на співвідношенні між прогнозованим навантаженням і допустимим навантаженням на одну репліку. У загальному вигляді, ця модель описується формулою:

$\text{required_replicas} = ((\text{predicted_cpu} / \text{target_cpu_per_pod}) * \alpha)$, де

predicted_cpu - прогнозоване значення навантаження,

$\text{target_cpu_per_pod}$ - цільове навантаження на один под,

α - коефіцієнт запасу, дозволяє врахувати можливі похибки прогнозу та забезпечити запас ресурсів у разі раптового зростання навантаження.

Коефіцієнт запасу ($\alpha > 1$) використовується для врахування невизначеності прогнозу та можливих короточасних піків навантаження. Отже, евристична модель виконує роль сполучної ланки між результатами машинного навчання та практичним механізмом масштабування.

Додатково вводяться обмеження на мінімальну та максимальну кількість реплік для експериментального середовища:

$$\text{min_replicas} \leq \text{required_replicas} \leq \text{max_replicas}$$

Окрему увагу в роботі приділено інтеграції предиктивного масштабування з механізмом KEDA (Kubernetes Event-Driven Autoscaling подієво-орієнтоване автомасштабування Kubernetes) [31][33]. У запропонованій архітектурі предиктивний автоскейлер використовується для визначення базового рівня масштабування, тоді як KEDA виконує роль реактивного шару, для застосування параметрів масштабування.

Проведений аналіз показує, що підхід Наваза Дхандала [18] є комплексним рішенням, яке залучає використання часових рядів і машинного навчання для прогнозування, а також евристичний механізм розрахунку кількості реплік та інтеграції з реактивними механізмами масштабування.

2.4 Аналіз LLM-орієнтованого підходу до автоскейлінгу на основі підходу інтелектуального автомасштабувальника подів

Інтелектуальний автомасштабувальник подів (IPA – Intelligent Pod Autoscaler), розроблений Шафіном Хаснатом [26], є прикладом інтеграції LLM у механізм автоскейлінгу. IPA – це такий підхід до масштабування Kubernetes деплойментів, у якому велика мовна модель використовується для аналізування стану системи та надає рекомендації щодо масштабування. Якщо класичні алгоритми базуються на порогових значеннях або формальних моделях, то IPA використовує можливості LLM для обробки різномірних даних і надання рекомендацій щодо масштабування у складних умовах.

Архітектура IPA складається з двох основних компонентів – контролер IPA та LLM агент. Контролер відповідає за збір метрик і взаємодію з кластером, а LLM агент (IPA агент), виконує аналіз даних та формує рекомендації щодо масштабування.

Контролер періодично отримує метрики з Prometheus. Ці метрики включають завантаження процесора, використання оперативної пам'яті, інтенсивність мережевого трафіку, швидкість надходження HTTP-запитів, події деплойменту, а також загальний стан ресурсів кластера. Потім вони агрегуються та передаються до LLM агента у вигляді текстового опису стану системи через HTTP POST-запит.

У вихідній реалізації IPA, використовується Gemini, велику мовну модель від Google. Вона працює через зовнішній API. Отримавши агрегований опис поточного стану Kubernetes-середовища, вона виконує контекстний аналіз навантаження та формує рекомендації для масштабування ресурсів.

В результаті аналізу IPA агент формує структуровану відповідь у форматі JSON, яка містить цільову кількість реплік, параметри обмежень

ресурсів (`cpu_limit`, `memory_limit`), а також параметри запитів ресурсів (`cpu_request`, `memory_request`) для тестового деплою.

Слід наголосити на тому, що важливою особливістю цього підходу є його гнучкість. Коли класичні системи працюють за строго визначеними правилами, LLM здатна адаптуватися до різних сценаріїв і враховувати низку складних залежностей між метриками відповідно до запиту. Це може бути особливо корисним для нашого сценарію з ГСДА, бо надає можливість враховувати затримки ініціалізації вузлів кластера, на які будуть розміщуватись поди.

Але, такий підхід також має свої обмеження. Основним із них є недостатня детермінованість при прийнятті рішень. Крім того, використання зовнішніх LLM API залежить від мережевого з'єднання, а отже може створювати додаткові затримки та підвищує вимоги до безпеки переданих даних.

2.5 Порівняльний аналіз розглянутих підходів

В попередніх розділах було детально розглянуто підходи предиктивного масштабування, запропонованого Навазом Дхандала [18], та LLM-орієнтованого підходу ІРА Шафіна Хасната [26]. Це дозволило визначити їхні ключові відмінності, переваги та обмеження у контексті управління обчислювальними ресурсами фізичних Kubernetes кластерів.

Підхід Наваза Дхандала [18] ґрунтується на обробці часових рядів метрик, отриманих за допомогою Prometheus, та застосування моделі машинного навчання Prophet для прогнозування майбутнього навантаження. Для розрахунку кількості реплік на основі прогнозу використовується евристична формула.

Натомість підхід ІРА Шафіна Хасната [26] використовує велику мовну модель як центральний елемент системи, який аналізує метрики та пропонує рекомендації щодо масштабування. На відміну від предиктивного масштабувальника Наваза Дхандала [18], підхід ІРА покладається на

здатність LLM узагальнювати інформацію та враховувати контекст. Саме це є особливо цінним для цієї праці, де розглядаються ГСДА.

Головна відмінність цих систем у характері прийняття рішень. У підході Наваза Дхандала [18] розрахунок рішення є детермінованим. Натомість у підході ІРА рішення залежить від внутрішніх механізмів мовної моделі, через що можливі галюцинації.

З точки зору точності прогнозування, перевагу має підхід Наваза Дхандала [18], бо він використовує моделі часових рядів, що дозволяє враховувати тренди та сезонність навантаження. У свою чергу ІРА аналізує поточний стан системи та його контекст, що дозволяє перевіряти та, за необхідності, коригувати розраховане значення, при непередбачуваних випадках зміни навантаження. Велика мовна модель здатна враховувати різномірні дані, аналізувати складні залежності між метриками в ГСДА та генерувати пояснення прийнятих рішень. Предиктивний автомасштабувальник Наваза Дхандала [18] має обмежену здатність до інтерпретації та підлаштування під нові непередбачувані зміни, оскільки його рішення визначаються формальними моделями.

Також підхід Наваза Дхандала [18] може бути реалізований у відносно простих умовах, бо використовує класичні алгоритми машинного навчання. Підхід ІРА, передбачає використання великої мовної моделі, а це потребує дещо більших обчислювальних ресурсів або ж доступу до зовнішніх сервісів. У таблиці 2.1 узагальнено основні характеристики розглянутих підходів.

Наведені характеристики показали, що кожен із підходів має як переваги, так і недоліки в рамках використання в ГСДА. Підхід, заснований на машинному навчанні забезпечує стабільність, передбачуваність та математичний розрахунок рішень, але має обмежену здатність до інтерпретації та адаптації в ГСДА. Підхід ІРА, у свою чергу, забезпечує гнучкість і можливість аналізу складних сценаріїв, але характеризується меншою детермінованістю та вищими вимогами до ресурсів.

Таблиця 2.1 - Порівняння підходів предиктивного автомасштабування та інтелектуального автомасштабувальника подів

Підхід	Предиктивний автомасштабувальник Наваза Дханда	ІРА Шафіна Хасната
Основа підходу	Машинне навчання, часові ряди	Велика мовна модель
Тип прийняття рішення	Детермінований	Недетермінований
Наявність прогнозування	Наявна	Відсутнє (неявне)
Інтерпретація рішень	Відсутня	Розширена
Стійкість результатів	Висока	Залежить від моделі
Вимоги до ресурсів	Помірні	Залежить від моделі
Залежність від зовнішніх сервісів	Відсутня	Наявна

Отже, результати порівняльного аналізу свідчать про доцільність поєднання розглянутих підходів з метою використання їхніх сильних сторін для подолання обмежень кожного з них.

2.6 Теоретичне обґрунтування вибору компонентів гібридного підходу для малих гетерогенних дослідницьких кластерів

Запропонований в цій роботі підхід орієнтований на застосування в малих дослідницьких кластерах де безпека та незалежність є головними параметрами, а також у яких обмежені ресурси, відсутня повноцінна хмарна інфраструктура та наявна гетерогенність вузлів. З урахуванням окреслених вимог, із підходу предиктивного автомасштабування Наваза Дханда [18] було обрано компоненти, які забезпечують формалізованість і передбачуваність процесу масштабування, а саме: використання системи Prometheus для моніторингу та збору метрик, застосування моделей машинного навчання для прогнозування навантаження і використання евристичної моделі для визначення кількості реплік.

Ці компоненти дозволять сформувати детермінований механізм прийняття рішень, що є важливим для дослідницького середовища. Водночас у процесі адаптації евристичної моделі нами було уточнено спосіб, яким буде

відбуватись округлення результату розрахунку кількості реплік. У оригінальному підході використовується приведення до цілого значення (функція `int`), що фактично відкидає дробову частину. Такий підхід може призвести до недооцінки необхідної кількості реплік і, як наслідок, до перевантаження окремих подів. З огляду на це, у запропонованому в цій роботі підході доцільним є використання округлення в більшу сторону, тобто функції `ceil`, як рекомендовано офіційною документацією [38].

Це дозволяє гарантувати, що кількість реплік буде достатньою при обробці прогнозованого навантаження. І навіть у випадку отримання дробового значення подів, система виділить цілу кількість, яка буде не меншою за необхідну. Це зменшить ризик перевантаження окремих подів в умовах гетерогенних BYON кластерів.

Водночас використання інструменту KEDA як основного механізму масштабування було визначено недоцільним. Це пов'язано з тим, що KEDA функціонує на основі тригерів і порогових значень, і не враховує контекст змін навантаження, що не відповідає вимогам малих гетерогенних кластерів, побудованих за принципом BYON.

У зв'язку з цим, замість тригерно-орієнтованого підходу ми обрали використання аналітичного компонента на основі великої мовної моделі. З підходу IPA було запозичено концепцію LLM агента, який взаємодіє з системою через API та аналізує метрики. Особливо важливим для цієї роботи є можливість LLM враховувати затримки при управлінні динамічною інфраструктурою.

Окрему увагу приділено способу розміщення великої мовної моделі. З огляду на наявні обмеження деяких віртуалізованих середовищ, зокрема відсутність повноцінної підтримки GPU у гіпервізорах типу VirtualBox [48], LLM було винесено за межі Kubernetes кластера та розгорнуто на хості. Це дозволить зменшити навантаження на вузли кластера та забезпечити універсальність підходу незалежно від типу тестового середовища.

Також використано підхід інженерії запитів без додаткового навчання моделі, що дозволило інтегрувати локальну LLM без додаткового тонкого налаштування (fine-tuning) та значних обчислювальних витрат. Для забезпечення стабільності роботи системи використано жорсткі обмеження на зміну вхідних параметрів. При побудові запиту застосовано техніки рольової моделі та покрокового міркування [28], при якому модель послідовно аналізує співвідношення між поточним і прогнозованим навантаженням, характер тренду та можливі короточасні пікові навантаження. Обґрунтування використання обраних технік інженерії запитів було апробовано на Дев'ятнадцятій міжнародній конференції "Перспективи телекомунікацій" (ПТ-2025) [3].

Вибір моделі Llama2:7b зумовлено необхідністю забезпечення стабільної роботи в умовах обмежених ресурсів. Ця модель має помірні ресурсні вимоги до обчислювальної інфраструктури, забезпечує стабільність роботи в досліджуваному середовищі та достатню якість аналізу для класифікації стану системи. Внесок цієї роботи порівняно з існуючими підходами підсумовано у Таблиці 2.2

У запропонованому гібридному підході поєднано прогнозування навантаження на основі машинного навчання, евристичний механізм розрахунку кількості реплік із урахуванням коректного округлення та компонент на основі великої мовної моделі замість реактивного інструменту KEDA, для того, щоб врахувати більше контексту стану системи.

Обрана гібридна комбінація для реалізації гібридного підходу є обґрунтованою для використання в малих гетерогенних Kubernetes кластерах, позаяк вона забезпечує баланс між точністю прогнозування, стабільністю рішень, можливістю їхньої інтерпретації, а також можливістю коригування при непередбачуваних випадках різкої зміни навантаження.

Таблиця 2.2 - Порівняння нашого гібридного підходу порівняно з іншими

Підхід	Predictive Autoscaling [18]	LLM-орієнтований підхід IPA [26]	Наш гібридний підхід
Основний механізм масштабування	ML	LLM	ML та LLM
Розрахунок кількості реплік	ML-прогноз, евристичний розрахунок за формулою та реактивне/тригерне застосування змін за допомогою KEDA	LLM-агент безпосередньо генерує рішення щодо масштабування, яке передається контролеру для перевірки та застосування	Агент, який поєднує ML-прогноз із евристичним розрахунком за формулою та коригуванням за допомогою LLM
Стабільність і детермінованість	Залежить від якості вхідних даних та характеру навантаження	Можливі варіативні результати	ML-прогноз, евристичний розрахунок забезпечують детермінованість, LLM контекстну корекцію, компенсуючи недоліки предиктивного підходу при різких змінах навантаження.
Використання LLM	Не використовується	IPA Agent (Flask-додаток) як shared або dedicated сервіс, розгортається через Docker, взаємодіє з контролером через HTTP API	Локально через Ollama на фізичному хості поза Kubernetes-кластером
Модель LLM	Не використовується	Gemini 1.5 Flash	Llama 2:7B

2.7 Висновки до розділу 2

У ході проведеного в другому розділі аналізу найбільш придатних до інтеграції в досліджуване середовище підходів було встановлено наступне. Використання моделі Prophet у поєднанні з евристичними механізмами прийняття рішень забезпечує можливість предиктивного масштабування та ефективного використання ресурсів Kubernetes-кластера. Водночас

визначено, що підходи, засновані виключно на прогнозуванні навантаження, мають обмежені можливості щодо врахування загального стану гетерогенного середовища та особливостей його функціонування.

Через це обґрунтовано доцільність відмови від використання тригерно-орієнтованих механізмів, таких як KEDA, як основи системи масштабування та замінено на більш гнучкий варіант, а саме LLM.

Встановлено, що великі мовні моделі можуть використовуватись для контекстного аналізу стану кластера, інтерпретації метрик і формування рекомендацій щодо масштабування. Разом із тим виявлено, що повністю LLM-орієнтовані підходи характеризуються підвищеною недетермінованістю прийняття рішень та значними обчислювальними витратами.

У результаті порівняльного аналізу встановлено, що розглянуті підходи мають взаємодоповнювальні властивості: предиктивне автомасштабування забезпечує стабільність і формалізованість рішень, тоді як LLM підхід – їхню адаптивність, обґрунтованість та коригування розрахованого рішення. Водночас доведено, що жоден із проаналізованих підходів у оригінальному вигляді не є повністю придатним для застосування в умовах малих гетерогенних дослідницьких кластерів. Отримані результати стали основою для подальшого створення комбінованого, гібридного підходу, який здатен забезпечити баланс між детермінованістю та точністю ML і гнучкістю та пояснюваністю LLM, що особливо важливо для невеликих автономних гетерогенних кластерів.

РОЗДІЛ 3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ГІБРИДНОГО ПІДХОДУ ДО ПРЕДИКТИВНОГО МАСШТАБУВАННЯ

3.1 Вступ до розділу

У третьому розділі проведено експериментальну перевірку запропонованого у роботі гібридного підходу. У розділі описано створення тестового гетерогенного стенду, інтеграцію ML та LLM-компонентів у єдиний автономний агент, а також результати тестування в трьох ключових сценаріях динамічного навантаження. Підтверджено працездатність, проактивність та ефективність розробленого в роботі рішення в умовах обмежених ресурсів і динамічної архітектури BYON.

3.2 Створення тестового стенду та розгортання Kubernetes кластера

Обчислювальна система, на якій виконувалось дослідження, представлена персональним комп'ютером із архітектурою x64 під керуванням операційної системи Microsoft Windows 10 Pro. Система оснащена центральним процесором Intel із тактовою частотою 3.4 ГГц та 16 ГБ оперативної пам'яті, графічним процесором NVIDIA GeForce GTX 950 із обсягом відеопам'яті 2 ГБ.

Для проведення експериментального дослідження розгорнуто тестове середовище на базі операційної системи Linux Ubuntu 22.04 із використанням платформи віртуалізації VirtualBox. Обчислювальна інфраструктура представлена у вигляді двох віртуальних машин (ВМ), які виконують ролі вузлів Kubernetes кластера та дозволяють змодельовати гетерогенне середовище.

Першим етапом стало створення віртуальних машин із різними апаратними характеристиками, що дозволило імітувати неоднорідність ресурсів. Вузол керування (control plane) було створено з такими параметрами: 2 віртуальних ЦП та 7 ГБ оперативної пам'яті, тоді як робочий вузол (worker node) мав 2 віртуальних ЦП та 10 ГБ оперативної пам'яті, що дозволило відтворити умови гетерогенних Kubernetes середовищ.

Наступним кроком було налаштування мережевої взаємодії між віртуальними машинами. Для цього для кожної ВМ було налаштовано 2 віртуальних адаптера. Перший, Bridge для підключення до зовнішньої мережі. Це дозволяє отримувати IP-адреси в одній підмережі з хостом, аби забезпечити доступ до сервісів кластера ззовні та для зв'язку з LLM-сервісом. Другий адаптер обрано як `intnet` для внутрішньої взаємодії між вузлами Kubernetes кластера.

Після підготовки інфраструктури було розгорнуто Kubernetes кластер. По-перше, здійснення ініціалізації контрольної площини виконано на вузлі master за допомогою `kubeadm` [34]. `Kubeadm` – це утиліта для автоматизованого розгортання та конфігурації кластера Kubernetes. Під час цього процесу було розгорнуто базові компоненти кластера, а саме: API-сервер, контролери та планувальник. По-друге для забезпечення мережевої взаємодії між подами використано CNI-плагін `Flannel` [15], який реалізує накладену (overlay) мережу.

Після ініціалізації керуючого вузла, робочий вузол було приєднано до кластера за допомогою команди `kubeadm join`. В результаті було сформовано повноцінний Kubernetes кластер, який складається з одного керуючого та одного робочого вузла.

Для забезпечення доступу до сервісів ззовні було розгорнуто компонент `MetalLB` [42], який реалізує балансування навантаження у середовищах без хмарного провайдера. У даному дослідженні `MetalLB` було налаштовано у режимі Layer 2 (L2), що дозволяє призначати сервісам типу `LoadBalancer` IP-адреси з заданого пулу. Цей пул IP-адрес було задано у

ресурсі IPAddressPool у просторі імен metallb-system. Додатково було встановлено Ingress-контролер на базі NGINX, який забезпечує маршрутизацію HTTP-запитів до відповідних сервісів.

Наступним етапом стало встановлення системи моніторингу Prometheus для збору та аналізу метрик. Prometheus було розгорнуто як окремий сервіс із типом LoadBalancer, що дозволило отримувати доступ до метрик ззовні. Основною метрикою, що використовується у дослідженні, є CPU-навантаження подів, яке обчислюється на основі метрики `container_cpu_usage_seconds_total`.

Для проведення експериментів було створено тестовий додаток WireMock, який використовувався як цільовий контейнеризований застосунок на який надходило навантаження для дослідження процесів масштабування.

3.3 Підготовка та налаштування LLM компонента до інтеграції в гібридний підхід

Проміжним етапом між обчисленням ML рекомендаційного значення та його застосуванням є LLM-компонент. Його реалізовано за допомогою середовища Ollama [44] [45]. Окрім рекомендаційного значення розрахованого ML, йому будуть передаватися параметри поточного стану системи, зокрема, кількість активних подів, прогнозоване пікове та середнє навантаження, останнє зафіксоване значення навантаження та тренд зміни навантаження.

Також завданням LLM є класифікація поточної ситуації та формування пояснення. Зокрема, модель визначає один із чотирьох типів ситуацій:

- стабільне зростання (`stable_growth`);
- ризик короткочасного піку (`short_spike_risk`);
- стабільне низьке навантаження (`low_steady_load`);
- невизначеність прогнозу (`forecast_uncertainty`).

Додатково LLM формує текстове пояснення, яке обґрунтовує прийняте рішення на основі вхідних параметрів. Це значно підвищує інтерпретованість системи, що є важливим для практичного застосування та аналізу результатів.

Відповідь LLM повертається у форматі JSON, який містить поля `decision`, `target_replicas`, `relation`, `situation_type` та `reason`. Після отримання відповіді виконується її валідація та парсинг.

Взаємодія Kubernetes кластера та LLM здійснюється через HTTP API. Сервіс предиктивного масштабування формуватиме запити до LLM, передаючи поточні метрики, результати прогнозування та рекомендовану кількість реплік. Для цього використовується кінцева точка (endpoint) `/api/generate`, до якої передається структурований запит у форматі JSON. Запит містить текстовий промпт, який формує контекст для аналізу. Як відповідь, LLM повертатиме структуроване рішення також у форматі JSON, яке буде використано для подальшого аналізу та застосування.

У результаті інтеграції LLM-компонента можна отримати систему, яка поєднує детермінованість розрахунків та гнучкість інтелектуального аналізу. Це дозволяє підвищити якість прийняття рішень, зменшити ризик помилок та забезпечити кращу адаптацію до змін у навантаженні.

Таким чином, LLM-компонент виступає ключовим елементом гібридного підходу, забезпечуючи перехід від простого прогнозування до контекстно-орієнтованого управління навантаженням в Kubernetes кластерах.

3.4 Інтеграція ML в поєднанні з LLM в єдиний гібридний агент

Після розгортання тестового середовища наступним етапом є створення та реалізація алгоритму гібридного масштабування. Реалізація алгоритму здійснювалася у вигляді окремого сервісу `predictive LLM scaler`, який розгортається у Kubernetes кластері. Сервіс функціонує як автономний агент, який збирає метрики, виконує прогнозування, розраховує необхідну

кількість реплік та взаємодіє з LLM для формування остаточного рішення. (Рис. 3.1)

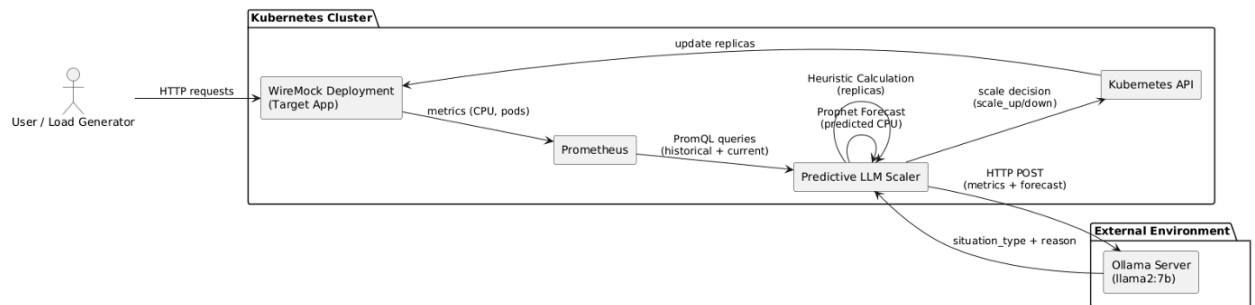


Рисунок 3.1 - Повний цикл роботи розробленої системи

Першим кроком роботи алгоритму є отримання поточних метрик із системи моніторингу Prometheus. Для цього використовується PromQL-запит, який обчислює агреговане (середнє) CPU навантаження для всіх подів заданого деплойменту. Далі визначається кількість активних подів, що дозволяє оцінити поточну конфігурацію системи. Ці дані формують базовий набір параметрів, який використовується для подальшого аналізу.

Наступним етапом є формування історичного часового ряду CPU-навантаження. Для цього виконується запит до Prometheus за часовий інтервал, у 7 днів. Це дає можливість навчити модель закономірностям, а саме добове навантаження, різниця між буднями і вихідними, загальний тренд тощо. Далі модель формує прогнозовані значення CPU для кожного 5 хвилинного інтервалу в межах заданого горизонту прогнозування в 1 годину (12 періодів по 5 хвилин), як було обгрунтовано в запропонованому підході Наваза Дхандала [18].

У результаті прогнозування отримуються значення `yhat`, `yhat_lower` та `yhat_upper`. Для прийняття рішень використовується значення `yhat_upper`, яке відображає верхню межу прогнозу.

Далі виконується розрахунок необхідної кількості реплік. Варто зазначити, що у процесі реалізації було внесено ряд змін у порівнянні з базовим підходом Наваза Дхандала [18]. Зокрема, було змінено значення

допустимого навантаження на один под (TARGET_CPU_PER_POD). У базовому підході використовується значення 0.7 CPU, що відповідає середовищам із потужними вузлами. У нашій роботі це значення було зменшено до 0.05 CPU. Така зміна зумовлена обмеженими ресурсами тестового середовища (VirtualBox, 2 vCPU на вузол) та необхідністю забезпечити чутливість алгоритму до змін навантаження. Також було змінено межі масштабування (MIN_REPLICAS та MAX_REPLICAS). У базовому підході [18] ці значення становлять 2 та 50 відповідно. У нашому дослідженні встановлено межі від 1 до 8 реплік.

Після розрахунку рекомендованої кількості реплік визначається співвідношення між поточним та рекомендованим значенням. Якщо рекомендована кількість перевищує поточну, формується рішення про масштабування вгору. Якщо ж вона є меншою, тоді формується рішення про масштабування вниз. У випадку рівності значень система не виконує жодних дій.

Наступним етапом є передача всіх отриманих даних до LLM-компонента. На основі цих параметрів LLM виконує контекстний аналіз та класифікує ситуацію за одним із типів: стабільне зростання, ризик короткочасного піку, стабільне низьке навантаження або невизначеність прогнозу. Додатково LLM формує текстове пояснення, яке обґрунтовує прийняте рішення.

Після застосування змін, система продовжує моніторинг та переходить до наступного циклу виконання. Так реалізується безперервний цикл управління ресурсами, який включає збір даних, прогнозування, прийняття рішення та його застосування.

Отже, у нашому випадку поєднуються три рівні:

- машинне навчання (для прогнозування навантаження);
- евристична модель (для розрахунку кількості реплік);
- LLM (для інтерпретації стану та надання пояснення з можливістю корегування рішення).

Такий підхід дозволяє досягти балансу між точністю, стабільністю та інтерпретованістю системи.

У результаті реалізації алгоритму було створено повністю автономний механізм управління масштабуванням, який здатний адаптуватися до змін навантаження та працювати у гетерогенному BYON середовищі з обмеженими ресурсами. Це створює основу для подальшого експериментального дослідження ефективності запропонованого підходу.

3.5 Експериментальне дослідження ефективності підходу

Для оцінки ефективності запропонованого гібридного підходу до проактивного масштабування було проведено серію експериментів у тестовому середовищі, основною метою яких є аналіз поведінки системи в умовах різних типів навантаження та перевірка здатності розробленого агента адаптуватися до змін робочого навантаження.

У ході дослідження було розглянуто три основні сценарії, які відображають типові режими роботи реальних систем:

- поступове зростання навантаження;
- різке зростання навантаження;
- зниження навантаження.

Для кожного сценарію здійснювався аналіз взаємозв'язку між фактичним CPU-навантаженням, прогнозованими значеннями та кількістю реплік. Основними метриками оцінювання були такі:

- поточне CPU навантаження;
- прогнозоване пікове значення CPU;
- рекомендована кількість реплік;
- фактична кількість реплік.

Верифікація моделі була спрямована на перевірку коректності роботи системи та включає такі етапи:

1. порівняння з теоретичною оцінкою -
перевірка відповідності розрахованої

кількості реплік значенню, отриманому за евристичною формулою;

аналіз граничних випадків - дослідження поведінки системи при: різкому зростанні навантаження, поступовому зростанні навантаження та спаді навантаження;

2. перевірка логічної узгодженості - оцінка відповідності рішень LLM прогнозу, метрикам та заданим правилам;
3. перевірка реалізації масштабування - підтвердження коректної зміни кількості реплік через Kubernetes API.

Очікується, що реалізований підхід забезпечить коректну та узгоджену роботу системи проактивного масштабування в гетерогенному Kubernetes середовищі. Зокрема, передбачається, що система зможе реагувати на зміну навантаження у 5-хвилинних циклах, враховуючи як поточні метрики, так і прогнозовані значення, використання прогнозування дозволить виявляти тенденції зміни навантаження (зростання, спад, стабільність) та враховувати їх під час прийняття рішень, евристична модель забезпечить базову оцінку необхідної кількості реплік, що відповідає прогнозованому навантаженню, LLM-агент забезпечить узгоджене прийняття рішень, враховуючи метрики, прогноз та евристичну оцінку, а також надаватиме пояснення сформованих рішень, система буде демонструвати стабільну поведінку без надмірних змін масштабу між сусідніми циклами, архітектура з винесеним LLM-агентом забезпечить можливість інтеграції з зовнішніми обчислювальними ресурсами, що відповідає умовам гетерогенних (cloud-edge, BYON) середовищ. Практичне значення: підхід може слугувати основою для систем інтелектуального управління ресурсами Kubernetes у сценаріях з вимогами до проактивності та автономності прийняття рішень.

3.5.1. Сценарій поступового зростання навантаження

У першому експерименті було змодельовано сценарій поступового зростання навантаження, який відповідає ситуаціям із передбачуваним збільшенням кількості запитів.

На початковому етапі система перебуває у стабільному стані з низьким рівнем навантаження: фактичне CPU навантаження є мінімальним, а прогнозоване значення знаходиться на помірному рівні. Це відповідає базовому режиму роботи системи.

У момент початку зростання навантаження, зображеному на рисунку 3.2, спостерігається підвищення як фактичного, так і прогнозованого значення CPU. Прогнозоване навантаження зростає відповідно до фактичного, відображаючи загальний тренд зміни системи. При цьому абсолютне значення прогнозу є вищим, що пояснюється використанням верхньої межі прогнозу (*yhat_upper*) та наявністю коефіцієнта запасу 1.2, тобто 20% як і в підході Наваза Дхандала [18].

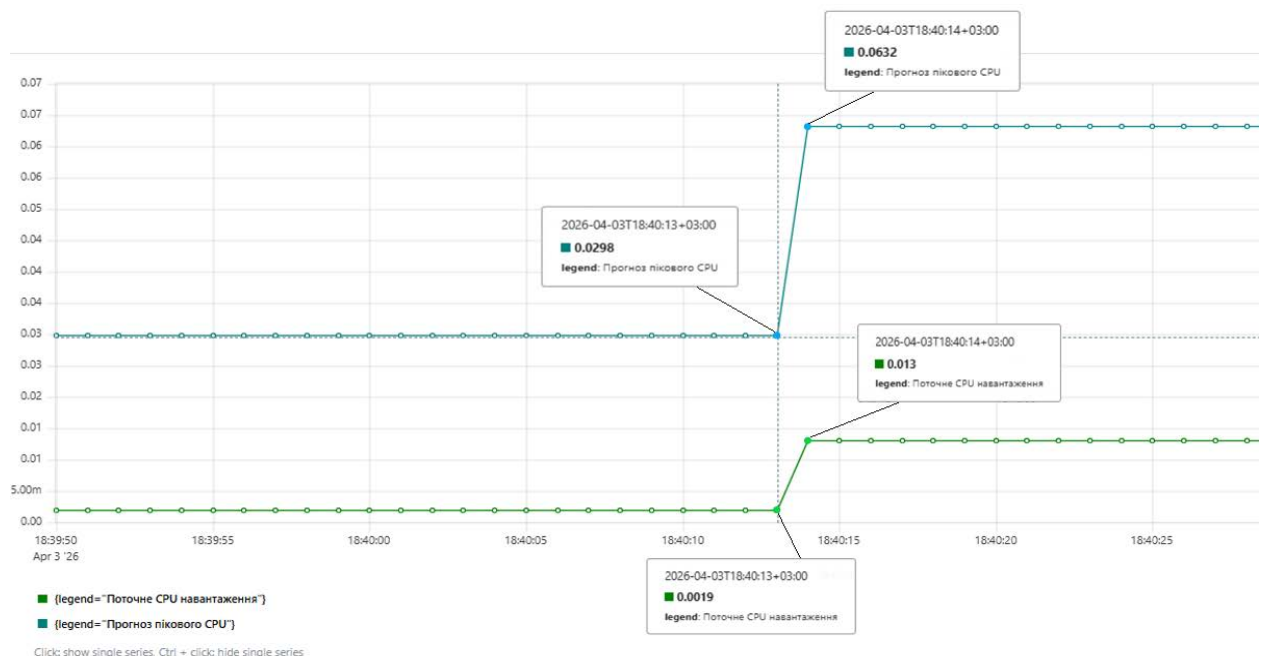
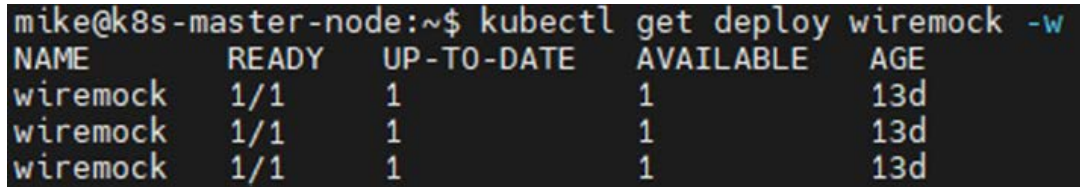


Рисунок 3.2 - Залежність фактичного та прогнозованого CPU при поступовому зростанні навантаження

Після первинної реакції система переходить у фазу стабілізації: фактичне навантаження фіксується на новому рівні, а прогноз також стабілізується, відображаючи оновлений тренд. Це свідчить про те, що модель поступово адаптується до нових умов та починає коректніше відображати поведінку системи.



```
mike@k8s-master-node:~$ kubectl get deploy wiremock -w
NAME          READY    UP-TO-DATE    AVAILABLE    AGE
wiremock      1/1      1             1            13d
wiremock      1/1      1             1            13d
wiremock      1/1      1             1            13d
```

Рисунок 3.3 - Динаміка зміни кількості реплік при поступовому зростанні навантаження

Незважаючи на зростання прогнозованого CPU, кількість реплік залишається на рівні 1 (Рис. 3.3), це свідчить про достатню продуктивність одного поду для обробки даного навантаження. Таким чином, система коректно визначає, що додаткове масштабування не є необхідним, і уникає надлишкового використання ресурсів.

У логах, наведених на рисунку 3.4, відображено повний цикл роботи гібридного агента масштабування. На початку система ініціює новий цикл (“Starting scaling cycle”) та виконує обробку даних, у межах якої збираються поточні метрики з Prometheus. У розглянутому випадку значення CPU-навантаження є низьким (близько 0.013), а кількість подів становить 1. Далі формується прогноз за допомогою моделі Prophet, який також свідчить про відсутність значного зростання навантаження. На основі прогнозу, евристична модель визначає рекомендовану кількість реплік, яка дорівнює поточній. Після цього дані передаються до LLM-компонента, який виконує контекстний аналіз і формує рішення. У даному випадку LLM підтверджує відсутність необхідності масштабування (no_action), оскільки навантаження є стабільним. На завершальному етапі система порівнює поточну та цільову кількість реплік і, за їх рівності, не виконує масштабування (“No scaling action applied”). Таким чином, лог демонструє узгоджену роботу всіх

компонентів системи та прийняття обґрунтованого рішення без надлишкового використання ресурсів.

```
mike@k8s-master-node:~$ kubectl logs -f deployment/predictive-llm-scaler
2026-04-03 18:40:13 - INFO - Starting scaling cycle...
2026-04-03 18:40:13 - INFO - Chain [1] start processing
2026-04-03 18:40:13 - INFO - Chain [1] done processing
2026-04-03 18:40:13 - INFO - Current metrics: {'cpu_cores': 0.013, 'active_pods': 1}
2026-04-03 18:40:13 - INFO - Forecast: {'peak_cpu_next_hour': 0.063, 'avg_cpu_next_hour': 0.013, 'current_cpu_baseline': 0.013, 'trend': 'steady'}
2026-04-03 18:40:13 - INFO - Recommended replicas: 1
2026-04-03 18:40:13 - INFO - Current replicas: 1
2026-04-03 18:40:57 - INFO - Ollama raw response: {
  "relation": "equal",
  "decision": "no action",
  "target_replicas": 1,
  "situation_type": "stable_growth",
  "reason": "recommended=1 is justified by steady baseline and forecasted growth"
}
2026-04-03 18:40:57 - INFO - Parsed decision: {'decision': 'no_action', 'target_replicas': 1, 'reason': 'recommended=1 is justified by steady baseline and forecasted growth', 'relation': 'equal', 'situation_type': 'stable_growth'}
2026-04-03 18:40:57 - INFO - Decision=no_action, target=1, current=1, relation=equal, situation_type=stable_growth, reason=recommended=1 is justified by steady baseline and forecasted growth
2026-04-03 18:40:57 - INFO - No scaling action applied
```

Рисунок 3.4 - Логи роботи агента при поступовому зростанні навантаження

Отже, у даному сценарії система демонструє узгоджену поведінку між фактичним навантаженням, прогнозом та кількістю реплік. Відсутність масштабування підтверджує, що один под ефективно справляється з поточним навантаженням, а алгоритм прийняття рішень коректно запобігає зайвому збільшенню ресурсів. Це підтверджує ефективність нашого підходу в умовах передбачуваного навантаження.

3.5.2. Сценарій різкого зростання навантаження

У другому експерименті було змодельовано сценарій різкого зростання навантаження, який відповідає ситуаціям із раптовими піками трафіку. На початковому етапі система перебувала у стані низького навантаження. Після запуску сценарію відбулося різке зростання поточного навантаження, що проявилось у вигляді стрімкого підйому відповідної кривої (Рис. 3.5).

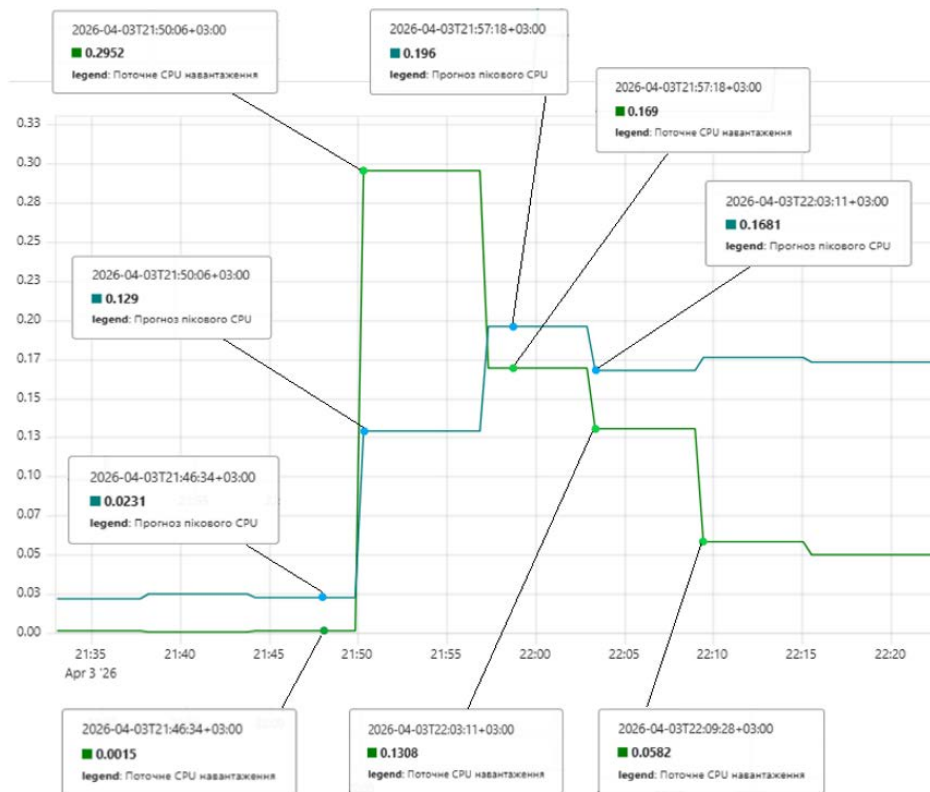


Рисунок 3.5 - Залежність фактичного та прогнозованого CPU при різкому зростанні навантаження

Прогнозована величина CPU навантаження також починає зростати, однак її поведінка є більш згладженою та відстає від фактичного значення. Це пояснюється тим, що модель Prophet через відсутність подібних патернів не здатна миттєво врахувати такі зміни. Додатково на точність прогнозу вплинуло використання агрегованого середнього значення за попередній інтервал моніторингу навантаження, що також призвело до згладжування різкого піку.

Незважаючи на це, система поступово адаптувалась та виконала масштабування. Система компенсує зазначені обмеження за рахунок використання верхньої межі прогнозу (`yhat_upper`) та коефіцієнта запасу. Це дозволяє системі зберігати стабільність роботи та виконувати поетапне масштабування відповідно до зростання навантаження. Динаміку масштабування наведено на Рисунку 3.6.

```
mike@k8s-master-node:~$ kubectl get deploy wiremock -w
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
wiremock	1/1	1	1	15d
wiremock	1/4	1	1	15d
wiremock	1/4	1	1	15d
wiremock	1/4	1	1	15d
wiremock	1/4	4	1	15d
wiremock	2/4	4	2	15d
wiremock	3/4	4	3	15d
wiremock	4/4	4	4	15d
wiremock	4/5	4	4	15d
wiremock	4/5	4	4	15d
wiremock	4/5	4	4	15d
wiremock	4/5	5	4	15d
wiremock	5/5	5	5	15d

Рисунок 3.6 - Динаміка зміни кількості реплік при різкому зростанні навантаження

Як видно з рисунка, система виконує поетапне масштабування до 4 та 5 реплік, що дозволяє адаптуватися до зростаючого навантаження, навіть за умови неточного прогнозу. Робота системи відображена у логах (Рис. 3.7).

З логів видно, що на кожному етапі виконувався повний цикл, тобто збір метрик, прогнозування, розрахунок кількості реплік, аналіз за допомогою LLM та прийняття рішення. При цьому рішення про масштабування приймається навіть у випадках, коли прогноз не повністю відображає реальний пік навантаження.

```
2026-04-03 21:42:55 - INFO - Starting scaling cycle...
2026-04-03 21:42:55 - INFO - Chain [1] start processing
2026-04-03 21:42:55 - INFO - Chain [1] done processing
2026-04-03 21:42:55 - INFO - Current metrics: {'cpu_cores': 0.0015, 'active_pods': 1}
2026-04-03 21:42:55 - INFO - Forecast: {'peak_cpu_next_hour': 0.0211, 'avg_cpu_next_hour': 0.0, 'current_cpu_baseline': 0.0015, 'trend': 'steady'}
2026-04-03 21:42:55 - INFO - Recommended replicas: 1
2026-04-03 21:42:55 - INFO - Current replicas: 1
2026-04-03 21:43:58 - INFO - Ollama raw response: {
  "relation": "equal",
  "decision": "no_action",
  "target_replicas": 1,
  "situation_type": "stable_growth",
  "reason": "Recommended=1 is justified by steady baseline and forecasted growth"
}
2026-04-03 21:43:58 - INFO - Parsed decision: {'decision': 'no_action', 'target_replicas': 1, 'reason': 'recommended=1 is justified by steady baseline and forecasted growth', 'relation': 'equal', 'situation_type': 'stable_growth'}
2026-04-03 21:43:58 - INFO - Decision: no_action, target=1, current=1, relation=equal, situation_type=stable_growth, reason=recommended=1 is justified by steady baseline and forecasted growth
2026-04-03 21:43:58 - INFO - No scaling action applied
2026-04-03 21:48:58 - INFO - Starting scaling cycle...
2026-04-03 21:48:58 - INFO - Chain [1] start processing
2026-04-03 21:48:58 - INFO - Chain [1] done processing
2026-04-03 21:48:59 - INFO - Current metrics: {'cpu_cores': 0.2952, 'active_pods': 1}
2026-04-03 21:48:59 - INFO - Forecast: {'peak_cpu_next_hour': 0.129, 'avg_cpu_next_hour': 0.0581, 'current_cpu_baseline': 0.2953, 'trend': 'down'}
2026-04-03 21:48:59 - INFO - Recommended replicas: 4
2026-04-03 21:48:59 - INFO - Current replicas: 1
2026-04-03 21:49:55 - INFO - Ollama raw response: {
  "relation": "greater",
  "decision": "scale_up",
  "target_replicas": 4,
  "situation_type": "stable_growth",
  "reason": "Recommended=4 is justified by the increasing load and low forecast uncertainty"
}
2026-04-03 21:49:55 - INFO - Parsed decision: {'decision': 'scale_up', 'target_replicas': 4, 'reason': 'recommended=4 is justified by the increasing load and low forecast uncertainty', 'relation': 'greater', 'situation_type': 'stable_growth'}
2026-04-03 21:49:55 - INFO - Decision: scale_up, target=4, current=1, relation=greater, situation_type=stable_growth, reason=recommended=4 is justified by the increasing load and low forecast uncertainty
2026-04-03 21:49:55 - INFO - Scaled deployment default/wiremock to 4 replicas
2026-04-03 21:54:55 - INFO - Starting scaling cycle...
2026-04-03 21:54:55 - INFO - Chain [1] start processing
2026-04-03 21:54:55 - INFO - Chain [1] done processing
2026-04-03 21:54:56 - INFO - Current metrics: {'cpu_cores': 0.168, 'active_pods': 4}
2026-04-03 21:54:56 - INFO - Forecast: {'peak_cpu_next_hour': 0.196, 'avg_cpu_next_hour': 0.105, 'current_cpu_baseline': 0.1691, 'trend': 'down'}
2026-04-03 21:54:56 - INFO - Recommended replicas: 5
2026-04-03 21:54:56 - INFO - Current replicas: 4
2026-04-03 21:56:09 - INFO - Ollama raw response: {
  "relation": "greater",
  "decision": "scale_up",
  "target_replicas": 5,
  "situation_type": "stable_growth",
  "reason": "Recommended=5 is justified by the increasing load and low forecast uncertainty"
}
2026-04-03 21:56:09 - INFO - Parsed decision: {'decision': 'scale_up', 'target_replicas': 5, 'reason': 'recommended=5 is justified by the increasing load and low forecast uncertainty', 'relation': 'greater', 'situation_type': 'stable_growth'}
2026-04-03 21:56:09 - INFO - Decision: scale_up, target=5, current=4, relation=greater, situation_type=stable_growth, reason=recommended=5 is justified by the increasing load and low forecast uncertainty
2026-04-03 22:01:09 - INFO - Starting scaling cycle...
2026-04-03 22:01:09 - INFO - Chain [1] start processing
2026-04-03 22:01:09 - INFO - Chain [1] done processing
2026-04-03 22:01:09 - INFO - Current metrics: {'cpu_cores': 0.1308, 'active_pods': 5}
2026-04-03 22:01:09 - INFO - Forecast: {'peak_cpu_next_hour': 0.1681, 'avg_cpu_next_hour': 0.0956, 'current_cpu_baseline': 0.1308, 'trend': 'steady'}
2026-04-03 22:01:09 - INFO - Recommended replicas: 5
```

Рисунок 3.7 - Логи роботи агента при різкому зростанні навантаження

Таким чином, результати експерименту демонструють як сильні сторони, так і обмеження запропонованого нами підходу. З одного боку, система здатна адаптуватися до різких змін навантаження завдяки використанню коефіцієнта запасу та верхньої межі прогнозу. З іншого боку, реакція на непередбачуване навантаження моделі при використанні згладжених усереднених значень призводить до недооцінки пікових навантажень, що ще раз підкреслює необхідність корекції рішення за допомогою LLM.

3.5.3. Сценарій зниження навантаження

У третьому експерименті було досліджено поведінку системи при поступовому зниженні навантаження після різкого, непередбачуваного підвищення. Спостерігалось поступове зниження фактичного навантаження (Рис. 3.8). Це відображає зменшення кількості запитів та повернення системи до стабільного стану.

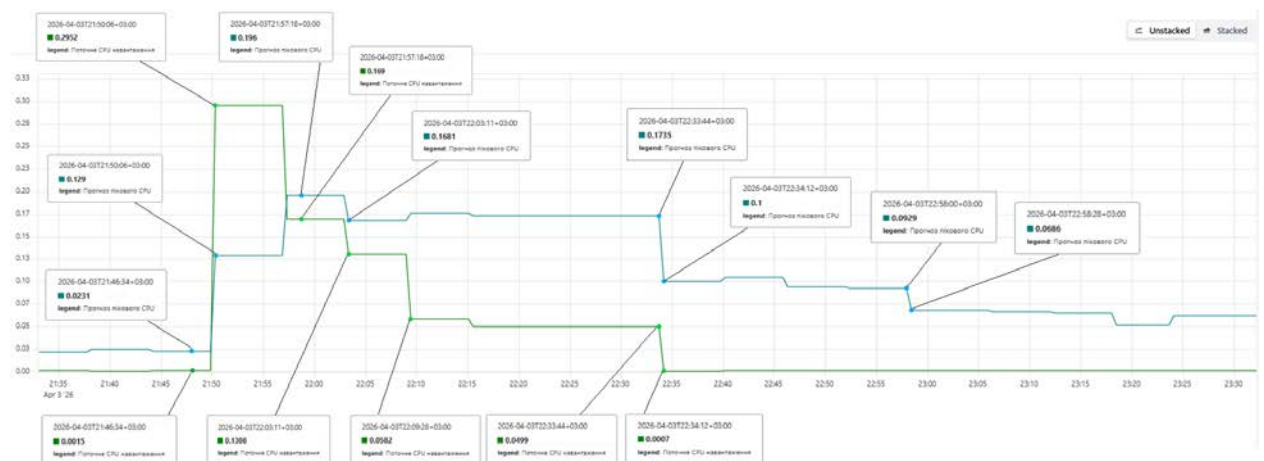


Рисунок 3.8 - Залежність фактичного та прогнозованого CPU при зменшенні навантаження

Водночас прогнозована крива демонструвала більш інерційний характер. Тобто, навіть після зниження фактичного навантаження прогнозоване значення зменшувалось поступово, більш плавно. Це пояснюється особливостями моделі Prophet, яка враховувала попередню

динаміку різкого зростання. У цей період система не виконувала негайного зменшення кількості реплік, що дозволило уникнути передчасного масштабування вниз у випадку можливого повторного зростання навантаження. Лише після стабілізації прогнозу система почала поступове зменшення кількості реплік. Зменшення відбувалось поетапно, що дозволяє уникнути різких змін і забезпечує стабільність системи. Таким чином, у даному сценарії підтверджується, що інерційність прогнозу виконує захисну функцію, а наш підхід забезпечує контрольоване та стабільне масштабування вниз.

3.6 Аналіз результатів експериментального дослідження

У результаті проведених експериментів було отримано набір даних, що дозволяє оцінити працездатність запропонованого нами підходу. Аналіз здійснювався на основі порівняння фактичного навантаження, прогнозованого значень та динаміки зміни кількості реплік у трьох основних сценаріях роботи системи.

Передусім зазначимо, що результати експериментів підтверджують коректність реалізації загальної логіки підходу. У всіх розглянутих сценаріях спостерігається чіткий причинно-наслідковий зв'язок між зміною навантаження, результатами прогнозування та прийняттям рішення про масштабування. Це свідчить про те, що розроблений механізм функціонує як цілісна система, яка поєднує збір метрик, прогнозування, евристичний розрахунок та застосування рішення.

У сценарії з поступовим зростанням навантаження система показала стабільну та передбачувану поведінку. Прогнозоване значення навантаження змінювалося узгоджено з фактичним, що підтверджує коректність роботи моделі прогнозування у випадку плавних змін. Крім цього, відсутність масштабування показала, що система здатна адекватно оцінювати достатність ресурсів і не виконує зайвих дій.

У сценарії різкого зростання навантаження було виявлено обмеження моделі прогнозування, зокрема її інерційність та залежність від характеру історичних даних. Прогнозована величина CPU відставала від фактичного значення на початковому етапі піку, що свідчить про недостатню адаптацію моделі до різких змін. Водночас система зберігала працездатність завдяки використанню верхньої межі прогнозу та коефіцієнта запасу, що дозволяло виконувати масштабування навіть за умов неповної точності прогнозу.

У сценарії зниження навантаження спостерігалася інерційна поведінка прогнозу, коли прогнозоване значення певний час залишалося вищим за фактичне. Це не призводило до різких змін, запобігаючи передчасному масштабуванню вниз і забезпечувало стабільність системи. Масштабування вниз виконувалося поступово після стабілізації прогнозу.

Окремо було підтверджено ефективність використання евристичної моделі та функції округлення в більшу сторону (`ceil`), що дозволяє уникнути недомасштабування.

Важливим результатом також є підтвердження доцільності інтеграції LLM, що дозволило враховувати контекст стану системи та формувати пояснення до прийнятих рішень. Це підвищує інтерпретованість підходу та відкриває шлях для керування динамічною інфраструктурою для зменшення затримок пов'язаних з ініціалізацією її вузлів.

Таким чином, результати експериментального дослідження підтверджують, що запропонований підхід до управління обчислювальними ресурсами є працездатним та може бути застосований у гетерогенних Kubernetes середовищах. Система демонструє здатність адаптуватися до різних типів навантаження, забезпечує узгодженість між прогнозом і рішенням та функціонує як єдиний інтегрований механізм.

Водночас отримані результати вказують на напрямки подальших досліджень, зокрема на можливість підвищення точності прогнозування у випадку різких пікових навантажень та оптимізації параметрів моделі. Таким чином, у нашій роботі основний акцент зроблено саме на розробці та

підтвердженні працездатності підходу, тоді як питання його оптимізації можуть бути предметом подальших покращень.

3.7 Адаптація підходу до хмарного середовища AWS

З метою забезпечення універсальності запропонованого підходу було виконано його адаптацію до хмарного середовища Amazon Web Services [7]. Це дозволило забезпечити роботу розробленого підходу як у локальних Kubernetes-кластерах, так і в хмарній інфраструктурі.

Для автоматизації розгортання інфраструктури використовувалась концепція інфраструктури як коду (Infrastructure as Code (IaC)) на основі Terraform [25]. За допомогою Terraform реалізовано автоматизоване створення інфраструктури, включаючи віртуальну приватну хмару (Virtual Private Cloud (VPC)), підмережі (subnets), групи безпеки (security groups) та два EC2 інстанси. Kubernetes-кластер було розгорнуто в Amazon Elastic Kubernetes Service (Amazon EKS) із групою робочих вузлів (Node Group), що складалася з одного Amazon EC2 інстансу, на якому були розгорнуті агент предиктивного масштабування, система моніторингу Prometheus та тестовий сервіс. Для збереження історичних метрик Prometheus використовувалося постійне сховище Amazon Elastic Block Store (Amazon EBS) через EBS CSI Driver.

Архітектура зберегла основні принципи локальної реалізації підходу. Велика мовна модель була розгорнута окремо від Kubernetes-кластера на виділеному EC2 інстансі через платформу Ollama, тоді як компоненти системи керування ресурсами працювали в середовищі Amazon EKS.

Для автоматизації процесів доставки та оновлення застосунків було реалізовано CI/CD пайплайн [21] на базі GitHub Actions [22] та Amazon ECR, що дозволило автоматизувати збирання контейнерних образів, їх публікацію та розгортання в Kubernetes-кластері. Для підвищення безпеки інтеграцію GitHub Actions із AWS реалізовано через OpenID Connect (OIDC) без використання статичних ключів доступу AWS.

Результати адаптації, включаючи конфігурації інфраструктури, Kubernetes маніфести та засоби автоматизації розгортання, опубліковано у репозиторії GitHub [29].

3.8 Висновки до розділу 3

У межах даного розділу було реалізовано запропонований гібридний підхід до управління обчислювальними ресурсами у Kubernetes кластері, що поєднує методи машинного навчання, а саме прогнозування часових рядів, евристичні алгоритми та аналіз на основі великих мовних моделей.

На першому етапі було створено тестовий стенд, який дозволив змодельовати гетерогенне середовище із використанням двох віртуальних машин та обмежених обчислювальних ресурсів. Було розгорнуто середовище для проведення експериментів з використанням платформи VirtualBox, а також налаштовано мережеву взаємодію та необхідні для роботи компоненти Kubernetes (Flannel, MetalLB, Ingress).

Використання LLM-компонента дозволило доповнити предиктивне масштабування контекстним аналізом, підвищити пояснюваність прийнятих рішень та забезпечити можливість корекції рекомендацій, сформованих на основі прогнозу моделі Prophet, а також заклало основу для подальшого розширення функціональності системи в задачах управління динамічною інфраструктурою ГСДА.

У результаті було створено повністю функціональний гібридний агент, який виконує безперервний цикл управління ресурсами: збір метрик, прогнозування, розрахунок кількості реплік, аналіз за допомогою LLM та застосування рішення через Kubernetes API.

Проведене експериментальне дослідження підтвердило працездатність запропонованого підходу в різних сценаріях навантаження. У випадку поступового зростання навантаження система демонструє стабільну та узгоджену поведінку без надлишкового масштабування. У сценарії різкого зростання навантаження виявлено обмеження моделі прогнозування, зокрема

її інерційність до пікових значень, що обумовлено відсутністю до цього подібних патернів, однак використання коефіцієнта запасу дозволяє частково компенсувати ці недоліки. У сценарії зниження навантаження система забезпечує контрольоване масштабування вниз без різких коливань.

Отримані результати свідчать про те, що розроблений підхід забезпечує баланс між предиктивністю, стабільністю та ефективністю використання ресурсів. Інтеграція LLM-компонента підвищує інтерпретованість прийнятих рішень та створює передумови для подальшого розвитку системи, зокрема у напрямі більш глибокої інтеграції інтелектуальних механізмів прийняття рішень.

Для забезпечення універсальності реалізованого в роботі підходу та можливості його використання як у локальних, так і в хмарних середовищах додатково було виконано адаптацію інфраструктури до хмарного середовища Amazon Web Services за допомогою Terraform. Автоматизація цього процесу дозволила скоротити час розгортання експериментального стенду та зменшити витрати на його підтримку.

Таким чином, у розділі 3 було підтверджено ефективність розробленого гібридного підходу до масштабування в умовах гетерогенного Kubernetes- середовища, обґрунтовано доцільність його використання для задач управління інфраструктурою в ГСДА з обмеженими ресурсами та проведено адаптацію до хмарного середовища.

ЗАГАЛЬНІ ВИСНОВКИ

1. Проведений аналіз сучасних підходів до організації розподілених обчислень показав, що контейнерна оркестрація стала базовим механізмом управління прикладними сервісами, а використання децентралізованих обчислювальних ресурсів відкриває перспективи створення масштабованих обчислювальних платформ без залучення дорогих спеціалізованих інфраструктур. Встановлено, що гетерогенним середовищам з динамічним складом вузлів властиві висока мінливість доступних ресурсів і непередбачуваність робочого навантаження, що істотно ускладнює застосування традиційних механізмів оркестрації. Також визначено, що більшість існуючих рішень проактивного масштабування орієнтовані на централізовані хмарні середовища та недостатньо враховують особливості автономних систем із неоднорідними ресурсами. Тому розроблення нових підходів до інтелектуального управління обчислювальними ресурсами та робочим навантаженням у гетерогенних Kubernetes-середовищах є актуальним.

2. На основі аналізу понад 30 сучасних наукових джерел встановлено, що найбільш перспективними для управління ресурсами в умовах малих фізичних Kubernetes-кластерів та BYON-середовищ є підходи, засновані на прогнозуванні навантаження методами часових рядів та використанні великих мовних моделей. Це дозволило обґрунтувати побудову гібридного агента шляхом поєднання підходу до предиктивного масштабування Kubernetes на основі прогнозування навантаження із використанням моделі Prophet та підходу IPA (Intelligent Pod Autoscaler), який використовує великі мовні моделі для аналізу метрик і формування рекомендацій щодо масштабування.

3. У роботі запропоновано та створено легковаговий гібридний предиктивний підхід, який адаптовано саме до умов малих гетерогенних

динамічних середовищ з обмеженими ресурсами. Запропоноване рішення поєднує детермінованість ML компонента з гнучкістю та пояснюваністю LLM. Це дозволило уникнути дороговартісних та ресурсовитратних вимог повноцінних RL/DRL моделей, а також зовнішніх хмарних сервісів.

4. Експериментальне дослідження запропонованого у роботі гібридного підходу показало його ефективність: запропонований підхід забезпечує кращий баланс між предиктивністю, стабільністю та ефективністю використання обмежених ресурсів порівняно з класичними реактивними механізмами. Розроблений підхід демонструє перспективність інтеграції методів штучного інтелекту в оркестрацію контейнерів і створює практичну основу для подальшого розвитку інтелектуальних систем управління ресурсами в сучасних розподілених обчислювальних середовищах. Перспективи подальших досліджень полягають в удосконаленні точності прогнозування різких пікових навантажень, оптимізації системи та розширенні ролі LLM до активної корекції рішень, врахування гетерогенності та динамічної доступності нод.

5. Результати праці апробовано на Міжнародній науково-технічній конференції «Перспективи телекомунікацій» (2025, 2026 рр.), опубліковано у вигляді тез конференції та наукової статті [2][3], а також робота була рекомендована до публікації у Springer.

СПИСОК ЛІТЕРАТУРИ

1. Алексеев, М. О., & Смаглюк, В. (2023). Management of Life Cycle of Computing Agents with Non-deterministic Lifetime in a Kubernetes Cluster. In 2023 IEEE 6th International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo) (pp. 80–83). IEEE. ISBN: 979-8-3503-4848-4.
2. Ільченко, М. В., & Алексеев, М. О. (2026). Підходи до управління обчислювальними ресурсами і робочим навантаженням у гетерогенних Kubernetes середовищах на основі штучного інтелекту. In *Перспективи телекомунікацій (ПТ-2026): матеріали Двадцятої міжнародної науково-технічної конференції* (13–17 квітня 2026 р., Київ) (pp. 245–249). КПІ ім. Ігоря Сікорського. <https://its.kpi.ua/sites/default/files/NDI%20TK%202021/%D0%9A%D0%BE%D0%BD%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D1%96%D1%8F%20%D0%9F%D0%A2/%D0%97%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA%D0%B8%20%D0%9F%D0%A2/%D0%97%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA%20%D0%BC%D0%B0%D1%82%D0%B5%D1%80%D1%96%D0%B0%D0%BB%D1%96%D0%B2%20%D0%BA%D0%BE%D0%BD%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D1%96%D1%97%20%D0%9F%D0%A2%202026%20ISSN%20%D1%80.pdf>
3. Ільченко, М. В., Іщенко, М. О., & Педан, С. І. (2025). Побудова запитів до великих мовних моделей для вирішення телекомунікаційних задач. *Збірник матеріалів Міжнародної науково-технічної конференції «Перспективи телекомунікацій»*, 255–257. <https://conferenc-journal.its.kpi.ua/article/view/332843>
4. Ковальов, А. В., & Алексеев, М. О. (2025). Підхід до автомасштабування кластерів Kubernetes на основі персональних обчислювальних ресурсів

- користувачів. In *XIX Міжнародна науково-технічна конференція «Перспективи телекомунікацій» (ПТ-2025): Збірник матеріалів конференції* (pp. 282–286). КПІ ім. Ігоря Сікорського. <https://doi.org/10.20535/PT2025.2663-502x>
5. Омельченко, В. В. (2025). Інформаційна технологія управління обчислювальними ресурсами в Kubernetes кластері [Дисертація на здобуття наукового ступеня доктора філософії, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»]. ELA KPI. <https://ela.kpi.ua/items/fbbc65ed-c320-4e20-b965-8c490bf68862>
 6. Тхай, О. О. (2025). Оптимізація автоскейлінгу в Kubernetes за допомогою штучного інтелекту [Дипломна робота бакалавра, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»]. ELA KPI. <https://ela.kpi.ua/items/65303611-1f3d-482c-8fe3-0340a5600ff9>
 7. Amazon Web Services. (2026a). Welcome to AWS documentation. <https://docs.aws.amazon.com/>
 8. Amazon Web Services. (2026b). Karpenter. <https://karpenter.sh/>
 9. Anderson, D. P. (2020). BOINC: A Platform for Volunteer Computing. *Journal of Grid Computing*, 18(1), 99–122. <https://doi.org/10.1007/s10723-019-09497-9>
 10. Avesha. (2026). Smart Scaler: Predictive autoscaling powered by Generative AI. <https://avesha.io/products/smart-scaler>
 11. Bai, H., Islam, M. T., Xu, M., & Buyya, R. (2026). ORACL: Optimized reasoning for autoscaling via chain of thought with LLMs for microservices. arXiv. <https://doi.org/10.48550/arXiv.2602.05292>
 12. Bai, X., Tang, Z., Yao, Z., Chen, Y., Guo, J., Wang, T., & Jia, W. (2025). Intelligent Kubernetes scheduling with large language models. In 2025 IEEE International Conference on Security, Privacy and Anonymity in Computation,

- Communication and Storage (SpaCCS) (pp. 16–23). IEEE. <https://doi.org/10.1109/SpaCCS67922.2025.00012>
- 13.Cao, R., Zhang, P., Wu, Y., Liu, J., & Su, H. (2025). Adaptive container scheduling based on reinforcement learning in Kubernetes. *CCF Transactions on High Performance Computing*. Advance online publication. <https://doi.org/10.1007/s42514-025-00223-4>
 - 14.Cloud Native Computing Foundation. (2026). Kubernetes established as the de facto ‘operating system’ for AI as production use hits 82% in 2025 CNCF Annual Cloud Native Survey. <https://www.cncf.io/announcements/2026/01/20/kubernetes-established-as-the-de-facto-operating-system-for-ai-as-production-use-hits-82-in-2025-cncf-annual-cloud-native-survey/>
 - 15.CNI Authors. (2026). flannel plugin. Container Networking Interface (CNI) Documentation. <https://www.cni.dev/plugins/v0.8/meta/flannel/>
 - 16.da Silva, L. M. D., Alves, P. V. A., Silva, S. N., et al. (2025). Adaptive horizontal scaling in Kubernetes clusters with ANN-based load forecasting. *Cluster Computing*, 28, 176. <https://doi.org/10.1007/s10586-024-04887-5>
 - 17.Dakić, V., Redžepagić, J., Bašić, M., & Žgrablić, L. (2024). Performance and latency efficiency evaluation of Kubernetes container network interfaces for built-in and custom tuned profiles. *Electronics*, 13(19), 3972. <https://doi.org/10.3390/electronics13193972>
 - 18.Dhandala, N. (2026). How to implement predictive autoscaling with Kubernetes and ML models. OneUptime Blog. <https://oneuptime.com/blog/post/2026-02-09-predictive-autoscaling-ml-models/view>
 - 19.Dysnix. (2026). PredictKube: AI-based predictive Kubernetes autoscaling. <https://dysnix.com/predictkube>
 - 20.El Kafhali, S. (2026). A survey of adaptive scheduling techniques, goals, and challenges in Kubernetes. *Archives of Computational Methods in Engineering*, 33, 6047–6070. <https://doi.org/10.1007/s11831-026-10497-8>

- 21.GitHub. (2024, November 7). What is CI/CD? GitHub Resources. <https://github.com/resources/articles/ci-cd>
- 22.GitHub. (2026). GitHub Actions documentation. GitHub Docs. <https://docs.github.com/en/actions>
- 23.Gogineni, A. (2024). Advancing Kubernetes network stack for high-performance AI/ML workloads. *International Journal on Science and Technology*, 15(4), 1–12. <https://www.ijssat.org/research-paper.php?id=2260>
- 24.Govindarajan, K., Naik, P., Goel, S., Prakash, A. P., & Jayachandran, P. (2025). Bringing GenAI awareness to workload management. In *Proceedings of the 26th International Middleware Conference Industry Track*. <https://doi.org/10.1145/3721463.3772058>
- 25.HashiCorp. (2026). Terraform overview. Terraform | HashiCorp Developer. <https://developer.hashicorp.com/terraform/docs>
- 26.Hasnat, S. (2025). IPA: Building AI driven Kubernetes Autoscaler. Medium. <https://shafinhasnat97.medium.com/ipa-building-ai-driven-kubernetes-autoscaler-54bfb17ac61e>
- 27.Hyndman, R. J., & Athanasopoulos, G. (2021). 12.2 Prophet model. In *Forecasting: Principles and Practice* (3rd ed.). OTexts. <https://otexts.com/fpp3/prophet.html>
- 28.Ilchenko, M. (2025). Next-Gen Telecom AI: Mastering Prompt Engineering for Innovation. *Information and Telecommunication Sciences*, 16(1), 22–29. <https://doi.org/10.20535/2411-2976.12025.22-29>
- 29.Ilchenko, M. V. (2026). Hybrid-predictive-AI-K8s-scaler [Computer software]. GitHub. <https://github.com/Michael13i/Hybrid-predictive-AI-K8s-scaler>
- 30.Jiang, J., Li, Q., Wang, P., & Liu, Y. (2025). DRKC: Deep reinforcement learning enhanced microservice scheduling on Kubernetes clusters in cloud-edge environment. *IEEE Transactions on Cloud Computing*. <https://doi.org/10.1109/TCC.2025.3624031>
- 31.KEDA Authors. (2026). KEDA | Kubernetes Event-driven Autoscaling. <https://keda.sh/>

32. Koziolk, H., et al. (2023). Lightweight Kubernetes distributions: A performance comparison. ACM SIGSOFT Software Engineering Notes.
<https://dl.acm.org/doi/10.1145/3578244.3583737>
33. Kubernetes Authors. (2025). Client Libraries: Python client for Kubernetes API. Official Kubernetes documentation.
<https://kubernetes.io/docs/reference/using-api/client-libraries/#python>
34. Kubernetes Authors. (2026). Kubeadm. Kubernetes Documentation. <https://kubernetes.io/docs/reference/setup-tools/kubeadm/>
35. Kubernetes Project. (2026). Cluster Architecture. Kubernetes Documentation. <https://kubernetes.io/docs/concepts/architecture/>
36. Kubernetes Project. (2025). Kubernetes v1.35 release blog. <https://kubernetes.io/blog/2025/12/17/kubernetes-v1-35-release/>
37. Kubernetes Enhancement Proposal. (2025). KEP-3063: Dynamic Resource Allocation. <https://kubernetes.io/docs/concepts/scheduling-eviction/dynamic-resource-allocation/>
38. Kubernetes. (2026, March 15). Horizontal Pod Autoscaling. Kubernetes Documentation. <https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale/>
39. Ma, P., Garg, S., & Barika, M. (2024). Research allocation in mobile volunteer computing system: Taxonomy, challenges and future work. *Future Generation Computer Systems*, 154, 251–265. <https://doi.org/10.1016/j.future.2024.01.015>
40. Matranga, S. (2024). *Scheduling Kubernetes tasks with reinforcement learning* [Master's thesis, Politecnico di Torino]. <https://webthesis.biblio.polito.it/31868/>
41. Meta Platforms, Inc. (2026). Quick start. Prophet. https://facebook.github.io/prophet/docs/quick_start.html
42. MetalLB Authors. (2026). MetalLB. <https://metallb.io/>
43. Nunavath, V. (2025). AI-enhanced self-healing Kubernetes for scalable cloud operations. *World Journal of Advanced Engineering Technology and Sciences*, 16(2), 21–29. <https://doi.org/10.30574/wjaets.2025.16.2.1255>

- 44.Ollama. (2025). Ollama: Get up and running with large language models. Official documentation. <https://docs.ollama.com/>
- 45.Ollama. (2026). Introduction. Ollama Documentation. <https://docs.ollama.com/api/introduction>
- 46.Omprakash, S. (2025). AI-driven adaptive pod co-location in Kubernetes using real-time multi-objective optimization. *Internet Technology Letters*. <https://doi.org/10.1002/itl2.70203>
- 47.Oracle. (2026a). Working with self-managed nodes. Oracle Cloud Infrastructure Documentation. <https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengworkingwithselfmanagednodes.htm>
- 48.Oracle. (2026b). Chapter 8. VBoxManage. Oracle VM VirtualBox Documentation. <https://www.virtualbox.org/manual/ch08.html>
- 49.Prometheus Authors. (2026). Overview. Prometheus. <https://prometheus.io/docs/introduction/overview/>
- 50.Sedai. (2026). Sedai: Self-driving cloud optimization platform. <https://sedai.io/>
- 51.Seyedkazemi Ardebili, M., & Bartolini, A. (2025). KubeIntellect: A modular LLM-orchestrated agent framework for end-to-end Kubernetes management. arXiv. <https://arxiv.org/abs/2509.02449>
- 52.Sgan Cohen, O., Malul, E., Meidan, Y., Mimran, D., Elovici, Y., & Shabtai, A. (2025). KubeGuard: LLM-assisted Kubernetes hardening via configuration files and runtime logs analysis. arXiv. <https://doi.org/10.48550/arXiv.2509.04191>
- 53.Skoularikis, M., Liatifis, A., Pliatsios, D., et al. (2025). Kubernetes in Edge and Cloud Computing: A Comparative Study of K3s, K0s, MicroK8s, and K8s. IEEE Conference Proceedings. <https://doi.org/10.1109/EEITE65381.2025.11166028>
<https://www.semanticscholar.org/paper/Kubernetes-in-Edge-and-Cloud-Computing%3A-A-Study-of-Skoularikis-Liatifis/bf5a95463722b3138c3b497a9529a08eabaed3cf>

- 54.Sliwko, L., & Mizeria-Pietraszko, J. (2026). Cluster workload allocation: Semantic soft affinity using natural language processing. *IEEE Access*. Advance online publication. <https://doi.org/10.1109/ACCESS.2026.3665989>
- 55.Tadi, G. (2025). The Vertical Pod Autoscaler and Pod Disruption Budget Chicken-and-Egg Problems: Analysis and Solutions for Kubernetes Workload Management. *International Journal of Science and Research*, 14(8), 974–977. <https://doi.org/10.21275/SR25819102904>
- 56.Thota, R. C. (2023). Optimizing Kubernetes workloads with AI-driven performance tuning in AWS EKS. *International Journal of Science and Research Archive*, 9(2), 1063–1073. <https://doi.org/10.30574/ijsra.2023.9.2.0546>
- 57.Vitui, A., & Chen, T.-H. (2025). Empowering AIOps: Leveraging large language models for IT operations management. arXiv. <https://doi.org/10.48550/arXiv.2501.12461>
- 58.Voelz, V. A., Bowman, G. R., & the Folding@home Consortium. (2023). Folding@home: Achievements from over twenty years of citizen science herald the exascale era. *Biophysical Journal*, 122(14), 2852–2863. <https://doi.org/10.1016/j.bpj.2023.03.025>
- 59.Wang, X., Zhao, K., & Qin, B. (2023). Optimization of task-scheduling strategy in edge Kubernetes clusters based on deep reinforcement learning. *Mathematics*, 11(20), 4269. <https://doi.org/10.3390/math11204269>
- 60.Xiao, Z., Liu, K., Hu, M., & Wu, D. (2024). DeepCTS: A deep reinforcement learning approach for AI container task scheduling. In *Proceedings of the 3rd Asia Conference on Algorithms, Computing and Machine Learning (CACML '24)*. <https://doi.org/10.1145/3654823.3654885>
- 61.Xu, Z., Gong, Y., Zhou, Y., Bao, Q., & Qian, W. (2024). Enhancing Kubernetes automated scheduling with deep learning and reinforcement techniques for large-scale cloud computing optimization. arXiv. <https://doi.org/10.48550/arXiv.2403.07905>

62. Yakubov, D., & Hästbacka, D. (2025). Comparative analysis of lightweight Kubernetes distributions for edge computing: Performance and resource efficiency. In Service-Oriented and Cloud Computing. ESOC 2025 (Lecture Notes in Computer Science, vol. 15547). Springer. https://doi.org/10.1007/978-3-031-84617-5_7 <https://arxiv.org/abs/2504.03656>
63. Yoon, Y., Choi, B., & Lee, J. (2025). Design of experiments-based adaptive scheduling in Kubernetes for performance and cost optimization. *Applied Sciences*, 15(18), 10098. <https://doi.org/10.3390/app151810098>
64. Yuan, H., & Liao, S. (2024). A time series-based approach to elastic Kubernetes scaling. *Electronics*, 13(2), Article 285. <https://doi.org/10.3390/electronics13020285>