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

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК ____004.8_____

До захисту допущено:
В.О. Завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем» зі спеціальності 121 “Інженерія програмного забезпечення” на тему: «Система масштабування контейнерів у Kubernetes кластері з використанням штучного інтелекту»

Виконав:
студент ІІ курсу, групи ІМ-31мп
Пєнской Володимир Вікторович _____

Керівник:
проф., д.т.н., с.н.с.
Сергієнко Анатолій Михайлович

_____

Консультант з нормоконтролю:
проф. каф. ОТ, д.т.н., професор
Жабін Валерій Іванович

_____

Рецензент:

_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 року

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

Факультет/ННІ Факультет інформатики та обчислювальної техніки
(повна назва)

Кафедра Кафедра обчислювальної техніки
(повна назва)

Рівень вищої освіти – другий (магістерський)

Спеціальність, спеціалізація 121 «Інженерія програмного забезпечення»
(код і назва)

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Михайло НОВОТАРСЬКИЙ
«__»_____20__р.

ЗАВДАННЯ
на магістерську дисертацію студенту

_____Пєнской Володимир Вікторович_____
(прізвище, ім’я, по батькові)

1. Тема дисертації «Система масштабування контейнерів у Kubernetes кластері з використанням штучного інтелекту»

науковий керівник дисертації Сергієнко Анатолій Михайлович, проф., д.т.н., с.н.с.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «08» листопада 2024р. №5016-с

2. Строк подання студентом дисертації 30.11.2024р.
3. Об’єкт дослідження Процеси автоматичного масштабування контейнеризованих застосунків у хмарних середовищах на базі Kubernetes.
4. Вихідні дані Методи та алгоритми прогнозування навантаження на контейнеризовані застосунки з використанням штучного інтелекту для оптимізації процесу горизонтального масштабування.
5. Перелік завдань, які потрібно розробити аналіз існуючих рішень масштабування контейнерів, розробка системи автоматичного масштабування з використанням штучного інтелекту, створення та навчання моделі прогнозування навантаження, проведення експериментальних досліджень системи, розробка стартап-проекту.

6. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1.			
2.			
3.			
4.			

7. Дата видачі завдання 01.09.2024

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1.	Аналіз предметної області та формування вимог до системи	01.09.2024 - 15.09.2024	
2.	Розробка архітектури системи та проектування підсистеми прогнозування	16.09.2024 - 30.09.2024	
3.	Формування навчального датасету та розробка моделі машинного навчання	01.10.2024 - 15.10.2024	
4.	Імплементація Kubernetes оператора та REST API	16.10.2024 - 31.10.2024	
5.	Проведення експериментальних досліджень та порівняльного аналізу	01.11.2024 - 15.11.2024	
6.	Розробка стартап-проекту та оформлення документації	16.11.2024 - 30.11.2024	

Студент _____ Володимир ПЄНСКОЙ
(підпис)

Науковий керівник дисертації _____ Анатолій СЕРГІЄНКО
(підпис)

Анотація

Магістерська дисертація присвячена розробці системи автоматичного масштабування контейнерів у Kubernetes кластері з використанням штучного інтелекту. Розроблена система використовує LSTM нейронну мережу для прогнозування навантаження та включає механізми попереднього та покрокового навчання моделі. Реалізовано Kubernetes оператор та REST API для взаємодії з системою. Експериментальні дослідження підтвердили ефективність розробленого рішення в різних сценаріях навантаження. Розроблено бізнес-модель для комерціалізації системи у форматі стартап-проекту.

Ключові слова: Kubernetes, контейнеризація, горизонтальне масштабування, машинне навчання, LSTM, прогнозування навантаження.

Annotation

The master's thesis is devoted to developing a container autoscaling system in Kubernetes clusters using artificial intelligence. The developed system uses LSTM neural networks for load prediction and includes mechanisms for pre-training and incremental learning of the model. A Kubernetes operator and REST API have been implemented for system interaction. Experimental studies have confirmed the effectiveness of the solution in various load scenarios. A business model has been developed for system commercialization as a startup project.

Keywords: Kubernetes, containerization, autoscaling, machine learning, LSTM, load prediction.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

ШІ – Штучний Інтелект

KEDA – Kubernetes Event-driven Autoscaling (Автомасштабування на основі подій у Kubernetes)

API – Application Programming Interface (Прикладний програмний інтерфейс)

CPU – Central Processing Unit (Центральний процесор)

RPS – Requests per Second (Кількість запитів за секунду)

HPA – Horizontal Pod Autoscaler (Горизонтальне автомасштабування подів)

SDK – Software Development Kit (Набір засобів розробки програмного забезпечення)

CRD – Custom Resource Definition (Визначення користувацького ресурсу в Kubernetes)

Kopf – Kubernetes Operator Pythonic Framework (Python-фреймворк для створення операторів Kubernetes)

LSTM – Long Short-Term Memory (Довга короткочасна пам'ять)

GRU – Gated Recurrent Unit (Вентильний рекурентний блок)

ARIMA – Autoregressive Integrated Moving Average (Авторегресійне інтегроване ковзне середнє)

RNN – Recurrent Neural Network (Рекурентна нейронна мережа)

MAE – Mean Absolute Error (Середня абсолютна похибка)

MSE – Mean Squared Error (Середньоквадратична похибка)

RMSE – Root Mean Square Error (Середньоквадратичне відхилення)

MAPE – Mean Absolute Percentage Error (Середня абсолютна відсоткова похибка)

HTTP – Hypertext Transfer Protocol (Протокол передачі гіпертексту)

PromQL – Prometheus Query Language (Мова запитів Prometheus)

RBAC – Role-Based Access Control (Контроль доступу на основі ролей)

MLP – Multi-Layer Perceptron (Багатошаровий перцептрон)

ARF – Adaptive Random Forest (Адаптивний Випадковий Ліс)

ADWIN – Adaptive Windowing (Адаптивне віконне керування)

ВСТУП

В сучасному світі, де розподілені системи та хмарні інфраструктури стають стандартом для розгортання та управління додатками, контейнеризація та оркестрація контейнерів виявляються ключовими компонентами. Kubernetes, як відома та широко використовувана платформа для оркестрації контейнерів, надає відмінні можливості для автоматизації та керування розгортанням додатків.

Однак із зростанням розміру та складності додатків, постає проблема ефективного масштабування контейнерів у Kubernetes кластері. У цьому контексті, штучний інтелект може виступати як потужний інструмент для автоматизації процесів масштабування та оптимізації ресурсів.

Мета цієї магістерської дисертації полягає в розробці та вдосконаленні системи масштабування контейнерів у Kubernetes з використанням штучного інтелекту. У цьому контексті, досліджуються історія розвитку контейнеризації та Kubernetes, теоретичні аспекти масштабування, роль штучного інтелекту у цьому процесі, а також проводиться огляд і порівняння існуючих рішень.

Дана дисертація важлива з погляду вирішення актуальних проблем ефективного управління ресурсами у розподілених системах. Її результати можуть стати основою для подальших розробок та вдосконалень в області оркестрації контейнерів та використання штучного інтелекту для оптимізації хмарних середовищ.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Огляд існуючих рішень систем масштабування контейнерів з використанням штучного інтелекту

Єдиним комерційним існуючим рішенням, що повністю відповідає темі магістерської дисертації, є PredictKube – інструмент, створений компанією Dysnix, для автоматичного масштабування та оптимізації інфраструктури Kubernetes на основі прогнозів штучного інтелекту, створених на основі як історичних даних навантаження проекту, так і будь-яких власних метрик [1].

PredictKube використовує штучний інтелект для прогнозування майбутнього навантаження на Kubernetes-систему. Це дозволяє їй автоматично масштабувати систему відповідно до прогнозованого навантаження, щоб забезпечити оптимальну продуктивність і витрати.

Процес прогнозованого автомасштабування можливий завдяки моделі штучного інтелекту, яка спостерігає за значеннями запитів в секунду (RPS) або CPU протягом певного періоду часу під час проекту, а потім показує тенденцію на період до 6 годин. PredictKube використовує клієнтські та відкриті джерела даних для навчання моделі.

PredictKube складається з двох частин:

1. PredictKube використовує KEDA, проект з відкритим кодом, який включає в себе автоматичне масштабування на основі подій для робочих навантажень у Kubernetes кластері. На стороні KEDA інтерфейс підключається через API до джерел даних про ваш трафік. PredictKube далі використовує Prometheus – галузевий стандарт для зберігання метрик. Там він анонімізує дані про трафік на стороні

клієнта, перш ніж надсилати їх до API, де модель працює з повністю знеособленою інформацією.

2. Далі, модель ШІ починає отримувати дані про те, що відбувається у Kubernetes кластері. На відміну від стандартних алгоритмів, заснованих на правилах, таких як горизонтальне автомасштабування (HPA), PredictKube використовує моделі машинного навчання для прогнозування даних часових рядів, наприклад, метрик CPU або RPS (кількість запитів у секунду) [1].

Чим більше даних надається моделі з самого початку, тим точнішим буде прогнозування. PredictKube рекомендує для початку використовувати дані за 2+ тижні.

Серед недоліків PredictKube можна виділити наступні:

1. Закрита модель – PredictKube надає можливість працювати з моделлю лише через зовнішнє API. Сама модель є закритою від коритсувачів. Це означає, що не можна налаштувати модель під свої потреби або зрозуміти, як вона працює. Також це може бути проблемою для системи, що не має доступу до мережі Інтернет, оскільки PredictKube не надає можливості розгорнути модель у власній системі.
2. Складність налаштування – PredictKube може бути складним у налаштуванні. При налаштуванні користувач має повністю заповнити усі необхідні параметри для навчання моделі, такі як горизонт прогнозування, часове вікно історичних даних, крок та граничний поріг вказаної метрики.
3. Залежність від сторонніх інструментів – PredictKube залежить від сторонніх інструментів, таких як KEDA та Prometheus. Це може ускладнити використання системи, якщо ці інструменти не використовуються або не можуть бути встановлені у системі.
4. Відсутність адаптивності – Модель, що надає PredictKube є натренованою на певному датасеті і не адаптується під унікальні

шаблони поведінки користувацького застосунку. PredictKube пропонує лише повне перенавчання моделі на користувацькому датасеті.

В додаток до PredictKube, існують інші системи масштабування контейнерів в середовищі Kubernetes. Наприклад, в самому Kubernetes існує вбудований механізм HorizontalPodAutoscaler (HPA), який дозволяє автоматично масштабувати кількість контейнерів залежно від навантаження. Крім того, раніше згаданий KEDA (Kubernetes Event-driven Autoscaler) розширює можливості автоматичного масштабування, дозволяючи реагувати на різні події і подавати їх як вхідні сигнали для масштабування.

Хоча ці системи надають ефективні методи автоматичного масштабування, важливо відзначити, що вони не мають вбудованого штучного інтелекту. Вони працюють на основі параметрів, таких як завантаження CPU або кількість запитів, і виконують масштабування відповідно до чітко заданих правил.

Також існують системи масштабування, що використовують штучний інтелект, однак вони заточені на інші завдання. Наприклад, CAST AI – платформа автоматизації та управління витратами у хмарному середовищі для кластерів Kubernetes. Вона використовує штучний інтелект для прогнозування майбутнього навантаження на кластери Kubernetes, а потім автоматично масштабує кластери відповідно до прогнозованого навантаження, щоб забезпечити оптимальну продуктивність і витрати. Тобто ця система масштабує саме віртуальні машини, на яких встановлено кластер, а не контейнери, що розгорнуті у кластері.

1.2. Огляд існуючих досліджень масштабування контейнерів за допомогою нейронних мереж

László Toka та співавтори у дослідженні "Machine Learning-Based Scaling Management for Kubernetes Edge Clusters" використовують LSTM та HTM для

прогнозування навантаження в edge-кластерах. Їхнє дослідження демонструє можливості цих моделей у передбаченні патернів навантаження [2].

У роботі "Machine Learning Based Auto-Scaling for Containerized Applications" Mahmoud Imdoukh та співавтори порівнюють ефективність LSTM та ARIMA моделей для прогнозування навантаження. Дослідження показує здатність обох моделей ефективно працювати з часовими рядами за умови наявності якісних історичних даних [3]. Однак для ефективного навчання LSTM потребує значного обсягу даних, що охоплюють різні сценарії навантаження, а також LSTM не є адаптивною, тобто при появі нових патернів навантаження, які не були представлені в тренувальному наборі, потрібно проводити повне перенавчання моделі.

Дослідження Nisarg S Joshi та співавторів "ARIMA-PID: container auto scaling" демонструє використання ARIMA моделі для прогнозування навантаження. Їхня робота підтверджує ефективність статистичних моделей у роботі з часовими рядами [4]. Однак ARIMA базується на припущенні, що дані є стаціонарними або можуть стати стаціонарними після диференціювання, а навантаження в контейнеризованих застосунках часто має нестаціонарний характер з раптовими сплесками та складними патернами. Також ARIMA є лінійною моделлю, що означає її обмежену здатність захоплювати нелінійні залежності в даних.

Guangba Yu та співавтори у дослідженні "Microscaler: Automatic Scaling for Microservices" представили систему на основі онлайн-навчання. Головною перевагою їхнього підходу є створення абстрактного критерію service power, на основі якого приймається рішення, щодо масштабування контейнеру. Проте автори відзначають, що Microscaler діє реактивно, масштабуючи додаток лише після виявлення порушення SLA, що може призводити до затримок порівняно з проактивними підходами [5].

У контексті масштабування контейнерів використовуються різні підходи машинного навчання. Традиційні моделі, такі як LSTM та ARIMA, мають здатність ефективно виявляти та прогнозувати патерни у часових рядах. Однак ці моделі не є адаптивними – вони потребують попереднього навчання на готовому наборі даних, який має містити сталі патерни навантаження. Такі моделі не можуть адаптуватися до нових патернів без повного перенавчання. Підходи на основі онлайн-навчання, на відміну від традиційних моделей, здатні адаптуватися до змін у реальному часі.

1.3. Постановка задачі дослідження

В результаті аналізу існуючих рішень та їхніх обмежень було виявлено потребу в розробці системи масштабування контейнерів, яка поєднує в собі адаптивність до змін з можливістю проактивного реагування на майбутні навантаження.

Система повинна мати мінімальний набір обов'язкових параметрів для початкової конфігурації моделей машинного навчання. Це дозволить спростити процес налаштування та розгортання системи в різних середовищах без необхідності глибокої експертизи в області машинного навчання.

Система має володіти здатністю до навчання на основі користувацьких патернів навантаження та динамічної адаптації до нових патернів у реальному часі. Це означає, що система повинна:

- Початково навчатися на поширених існуючих патернах навантаження
- Виявляти та адаптуватися до нових патернів без необхідності повного перенавчання
- Зберігати ефективність при зміні характеру навантаження

Система повинна забезпечувати проактивне масштабування на основі прогнозування майбутнього навантаження. Це включає:

- Передбачення зростання або падіння навантаження
- Врахування часу, необхідного для розгортання нових контейнерів
- Забезпечення готовності системи до прийому користувацьких запитів до моменту фактичного зростання навантаження

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі проведено аналіз існуючих рішень та наукових досліджень у сфері масштабування контейнерів з використанням штучного інтелекту. Розглянуто комерційне рішення PredictKube, яке використовує штучний інтелект для прогнозування навантаження, але має обмеження у вигляді закритої моделі та складності налаштування. Досліджено стандартні інструменти Kubernetes для масштабування – HorizontalPodAutoscaler та KEDA, які працюють на основі простих правил без використання прогнозування.

Аналіз наукових робіт показав, що існуючі підходи з використанням LSTM та ARIMA моделей потребують значного обсягу даних для навчання та не забезпечують адаптивності до нових патернів навантаження. Підходи на основі онлайн-навчання, хоча і є адаптивними, працюють реактивно, що може призводити до затримок у масштабуванні системи.

На основі проведеного аналізу сформульовано задачу дослідження – розробка системи масштабування контейнерів, яка забезпечить простоту налаштування, адаптивність до змін користувацьких патернів та проактивне масштабування на основі прогнозування навантаження. Система має початково навчатися на поширених патернах, адаптуватися до нових без повного перенавчання та враховувати час, необхідний для розгортання нових контейнерів.

У висновку можна зазначити, що дослідження в рамках першої частини магістерської дисертації виконане відповідно до поставлених завдань. Виявлені та проаналізовані ключові аспекти проблеми, що дозволяє перейти до наступного етапу дослідження – експериментальної частини, де буде розроблено та випробувано власну систему масштабування контейнерів у Kubernetes кластері за допомогою штучного інтелекту. Проведений аналіз створив теоретичну базу для подальшої розробки системи інтелектуального

масштабування, яка буде поєднувати переваги сучасних технологій машинного навчання та контейнерної оркестрації.

РОЗДІЛ 2
РОЗРОБКА СИСТЕМИ АВТОМАТИЧНОГО МАСШТАБУВАННЯ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

2.1. Архітектура системи масштабування контейнерів

Архітектура системи автоматичного масштабування, що базується на застосуванні нейронної мережі для прогнозування навантаження на контейнери в кластері Kubernetes, складається з взаємопов'язаних компонентів, кожен з яких виконує специфічні функції у процесі автоматичного масштабування додатків.

Основні компоненти системи включають:

- Нейромережевий предиктор (прогнозування навантаження)
- Prometheus адаптер (збір та обробка метрик)
- Kubernetes оператор (керування масштабуванням)
- Адаптивний конфігуратор (налаштування параметрів)
- Система моніторингу та логування

Нейронна мережа є ключовим елементом системи та використовується для прогнозування майбутнього навантаження на основі історичних даних. Математично процес прогнозування можна описати наступним чином:

$$y_{t+1} = f(x_t, x_{t-1}, ..., x_{t-n})$$

де y_{t+1} – прогнозоване значення навантаження, x_t – вектор метрик у момент часу t, а n – розмір часового вікна для аналізу історичних даних.

Система моніторингу Prometheus забезпечує збір метрик з кластера Kubernetes. Процес збору даних відбувається через визначені часові інтервали та формує часові ряди метрик. Кожна метрика m представляється у вигляді пари значень:

$$m = (v, t)$$

де v – значення метрики, а t – часова мітка.

Kubernetes оператор функціонує як асинхронний контролер, який обробляє події та метрики кластера. Оператор реалізує шаблон узгодження стану системи, порівнюючи поточний стан з бажаним та виконуючи необхідні дії для досягнення цільового стану. Процес прийняття рішень про масштабування базується на аналізі прогнозів нейронної мережі та поточного стану системи. Асинхронна природа оператора реалізується через використання черг подій та патерну публікації-підписки, що дозволяє системі ефективно обробляти велику кількість метрик та подій без створення надмірного навантаження на кластер. При цьому оператор підтримує власний кеш стану системи для оптимізації взаємодії з Kubernetes API та зменшення латентності операцій масштабування.

Система автоматичного масштабування реалізує комплексний підхід до забезпечення надійності та відмовостійкості. Архітектура системи передбачає механізми відновлення після збоїв та обробки помилок на всіх рівнях функціонування. Система логування забезпечує детальну фіксацію всіх операцій, що надає можливість проводити глибокий аналіз роботи системи та оперативно реагувати на виникаючі проблеми. Система реалізує автоматичне налаштування ключових параметрів функціонування, включаючи частоту збору метрик та горизонт прогнозування, що забезпечує її адаптивність до різноманітних умов експлуатації та профілів навантаження.

Модульна архітектура системи є одним з її ключових технічних рішень. Такий підхід до проектування забезпечує високий рівень гнучкості та розширюваності системи. Модульність дозволяє здійснювати модифікацію окремих компонентів без впливу на функціонування інших частин системи. Зокрема, архітектура передбачає можливість зміни структури нейронної мережі або інтеграції додаткових джерел метрик при збереженні загальної стабільності системи.

Нейромережевий предиктор системи базується на рекурентній архітектурі з довгою короткочасною пам'яттю, що забезпечує ефективний аналіз часових рядів та прогнозування майбутніх значень навантаження. Предиктор включає компоненти для обробки вхідних даних, навчання моделі та генерації прогнозів. Система реалізує механізм періодичного перенавчання моделі на основі нових даних, що забезпечує її адаптацію до змін характеристик навантаження та підтримує високу якість прогнозування протягом усього життєвого циклу системи.

Prometheus адаптер є важливим компонентом системи, розробленим з урахуванням специфіки роботи з часовими рядами метрик. Він реалізує асинхронний механізм отримання даних та забезпечує їх попередню обробку. Процес обробки включає фільтрацію аномальних значень та агрегацію даних за різними часовими інтервалами. Адаптер підтримує гнучке налаштування інтервалів опитування та може працювати з широким спектром метрик, що використовуються для прийняття рішень про масштабування.

Kubernetes оператор виконує роль центрального координатора системи. Він забезпечує інтеграцію з Kubernetes API та керує процесами масштабування. Оператор здійснює моніторинг стану розгорнутих додатків та забезпечує коректне виконання операцій масштабування з урахуванням встановлених обмежень та політик безпеки кластера.

Адаптивний конфігуратор системи виконує безперервний аналіз ефективності роботи та здійснює автоматичне коригування параметрів. Використовуючи методи статистичного аналізу, він виявляє закономірності у навантаженні та оптимізує налаштування системи. При виявленні значних змін у характері навантаження конфігуратор може адаптувати такі параметри як частота опитування метрик або розмір вікна аналізу даних.

Система моніторингу та логування забезпечує комплексний збір інформації про роботу всіх компонентів. Архітектура системи базується на

асинхронній взаємодії між компонентами, що забезпечує високу продуктивність та масштабованість. Комунікація відбувається через визначені інтерфейси, що спрощує процеси модифікації та розширення функціональності. Кожен компонент системи реалізує механізми автономної роботи та відновлення після збоїв, що забезпечує загальну відмовостійкість системи.

Інтеграція з Kubernetes реалізована через Custom Resource Definition (CRD), що розширює стандартний API Kubernetes та надає можливості керування налаштуваннями автоматичного масштабування. Взаємодія з Prometheus здійснюється через спеціалізований адаптер, який використовує PromQL для формування запитів до метрик. Система використовує механізм RBAC для забезпечення безпеки при взаємодії з API Kubernetes. Для цього системі надаються окремий ServiceAccount та відповідні ролі, які обмежують права доступу оператора необхідними операціями. Відмовостійкість системи забезпечується механізмами повторних спроб при тимчасових проблемах доступу до API Kubernetes або Prometheus. Система коректно обробляє ситуації недоступності окремих контейнерів або метрик, продовжуючи функціонування з доступними даними та відновлюючи повну функціональність при нормалізації ситуації.

2.2. Формування та аналіз навчального датасету веб-запитів

У даному дослідженні використовується модифікована версія датасету NASA-HTTP, який містить інформацію про HTTP-запити до веб-серверу Космічного центру Кеннеді NASA у Флориді. Оригінальний датасет охоплює період з 1 липня по 31 серпня 1995 року та містить детальну інформацію про кожен запит, включаючи хост, часову мітку, тип запиту, код відповіді HTTP та розмір відповіді в байтах [6]. Даний датасет містить важливі патерни веб-трафіку, які залишаються актуальними і сьогодні [7].

Оригінальні дані були трансформовані у часовий ряд з хвилинними інтервалами, де кожен запис містить дві колонки: часову мітку (minute) та кількість запитів за відповідну хвилину (count). Така трансформація дозволяє зосередитись на аналізі частоти звернень до серверу та виявленні часових патернів у навантаженні системи. Формат даних був оптимізований для проведення статистичного аналізу та побудови прогнозних моделей часових рядів.

У даному наборі даних наявна перерва у записах з 14:52:01 1 серпня до 04:36:13 3 серпня 1995 року, спричинена вимкненням веб-серверу через ураган Ерін. Ця перерва створює природний розрив у часовому ряді, який необхідно враховувати при аналізі та моделюванні даних. Загальна кількість зареєстрованих запитів за весь період становить 3 461 612, що свідчить про високу інтенсивність використання серверу.

Оригінальний датасет поданий у форматі ASCII, де кожен рядок відповідає одному HTTP-запиту. Після агрегації даних до хвилинних інтервалів було отримано часовий ряд, який дозволяє ефективно досліджувати патерни навантаження на сервер у різних часових масштабах: від хвилинних коливань до добової та тижневої циклічності. Часова роздільна здатність в одну хвилину є оптимальною для виявлення значущих патернів у навантаженні без надмірної деталізації, яка могла б ускладнити аналіз. У модифікованому наборі даних кожен запис представляє агреговану кількість запитів за хвилину, що дозволяє легко обчислювати статистичні показники та будувати візуалізації для різних часових періодів.

У процесі аналізу датасету HTTP-запитів до серверів NASA були отримані важливі статистичні показники, які дозволяють глибше зрозуміти характер навантаження на систему. Середнє значення запитів становить 42,53 запити за хвилину, що вказує на достатньо інтенсивне використання ресурсів серверів. Медіанне значення у 36 запитів за хвилину дещо нижче за середнє, що

свідчить про наявність певної асиметрії у розподілі даних та присутність викидів у бік більших значень.

Стандартне відхилення у 29,01 запитів демонструє значну варіативність навантаження на сервери протягом досліджуваного періоду. Це вказує на нерівномірність розподілу запитів у часі та потребу в детальному дослідженні причин таких коливань. Мінімальна кількість запитів становить 1 запит за хвилину, що може відповідати періодам низької активності користувачів або технічного обслуговування систем.

Максимальне значення у 405 запитів за хвилину є особливо важливим показником, оскільки воно визначає пікове навантаження на сервери. Цей показник майже в 10 разів перевищує середнє значення, що вказує на необхідність забезпечення значного запасу потужності серверної інфраструктури для обробки пікових навантажень. Загальна кількість запитів за весь період спостереження склала 3 461 612, що свідчить про високу загальну активність користувачів та інтенсивне використання ресурсів NASA.

Аналіз статистичних показників дозволяє зробити припущення про наявність певних часових патернів у навантаженні на сервери. Значна різниця між середнім та медіанним значеннями, а також велике стандартне відхилення можуть вказувати на циклічність навантаження, пов'язану з добовими, тижневими або сезонними змінами активності користувачів. Присутність екстремальних значень також може бути пов'язана з особливими подіями або періодами підвищеного інтересу до ресурсів NASA.

На основі візуалізації даних можна провести детальний аналіз часових патернів HTTP-запитів до серверів NASA. Діаграма часового ряду датасету зображена на рисунку 2.1.

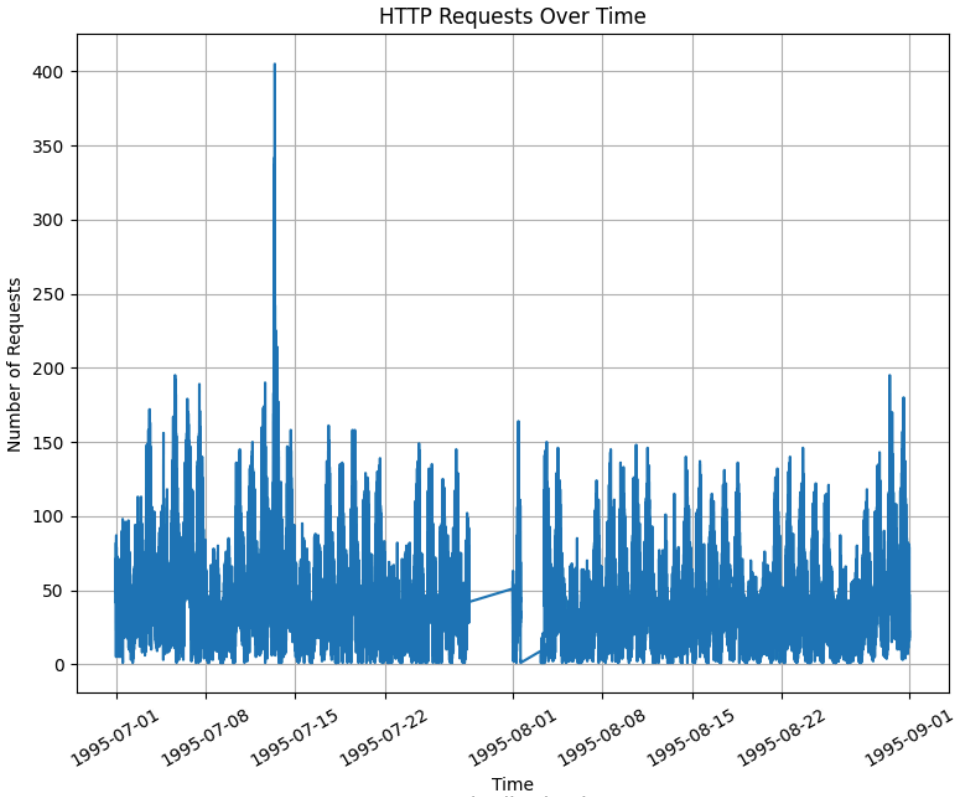

Рис. 2.1. Діаграма часового ряду датасету NASA

На графіку часового ряду спостерігається значна варіативність кількості запитів протягом всього періоду спостереження з липня по вересень 1995 року. Помітний один екстремальний пік, що сягає приблизно 400 запитів, який суттєво відрізняється від загального розподілу даних.

Гістограма розподілу кількості запитів, зображена на рисунку 2.2, демонструє асиметричний розподіл з довгим правим хвостом.

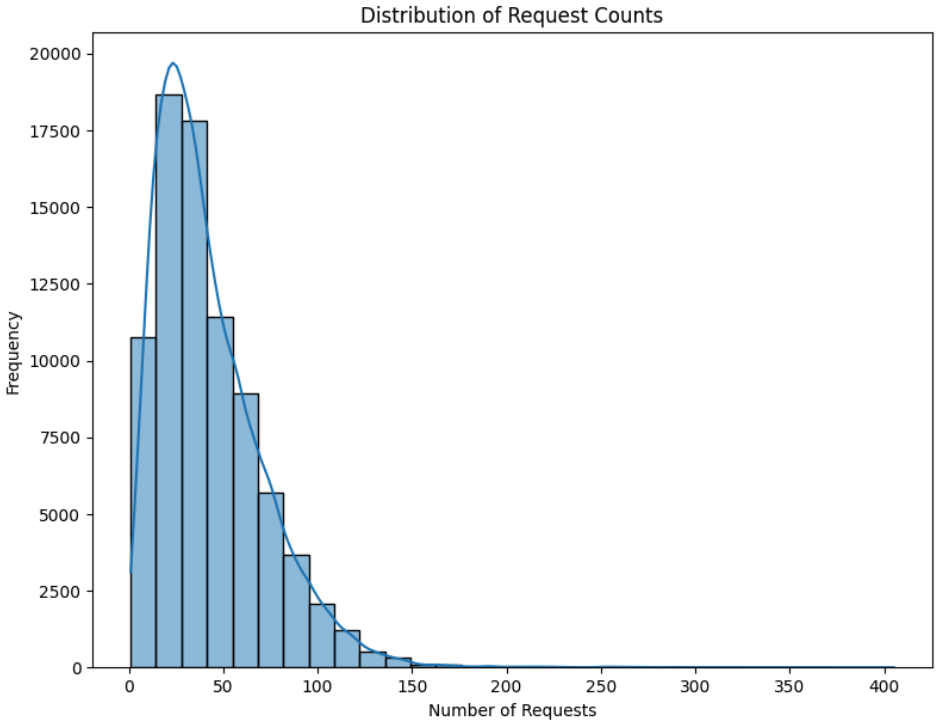

Рис. 2.2. Гістограма розподілу кількості запитів

Найбільша частота спостерігається в діапазоні 20-40 запитів за хвилину, що узгоджується з раніше визначеним медіанним значенням. Форма розподілу вказує на те, що екстремально високі значення кількості запитів є відносно рідкісними подіями.

Аналіз розподілу запитів за годинами доби, зображений на рисунку 2.3, показує чітку добову циклічність.

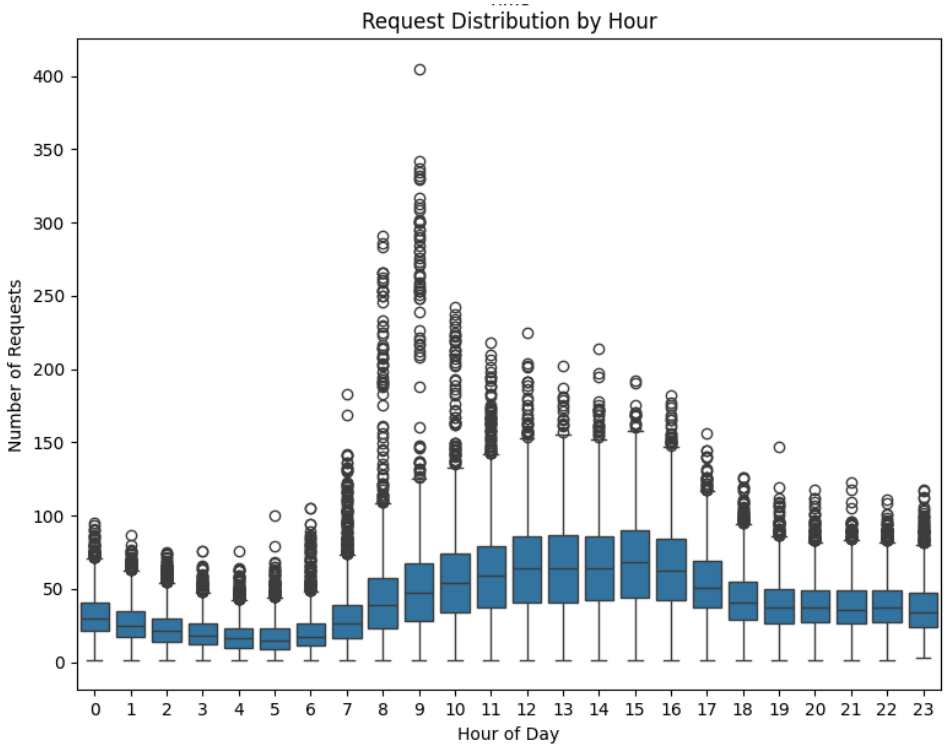

Рис. 2.3. Діаграма розподілу запитів за годинами доби

Спостерігається поступове зростання активності починаючи з 8 години ранку, з досягненням максимуму в період з 12 до 16 години. Після цього відбувається поступовий спад активності. Точки коробкового графіку також демонструють наявність значної кількості викидів, особливо у години пікового навантаження.

Розподіл запитів за днями тижня, зображений на рисунку 2.4, вказує на зниження активності у вихідні дні (позиції 5 та 6 на графіку, що відповідають суботі та неділі).

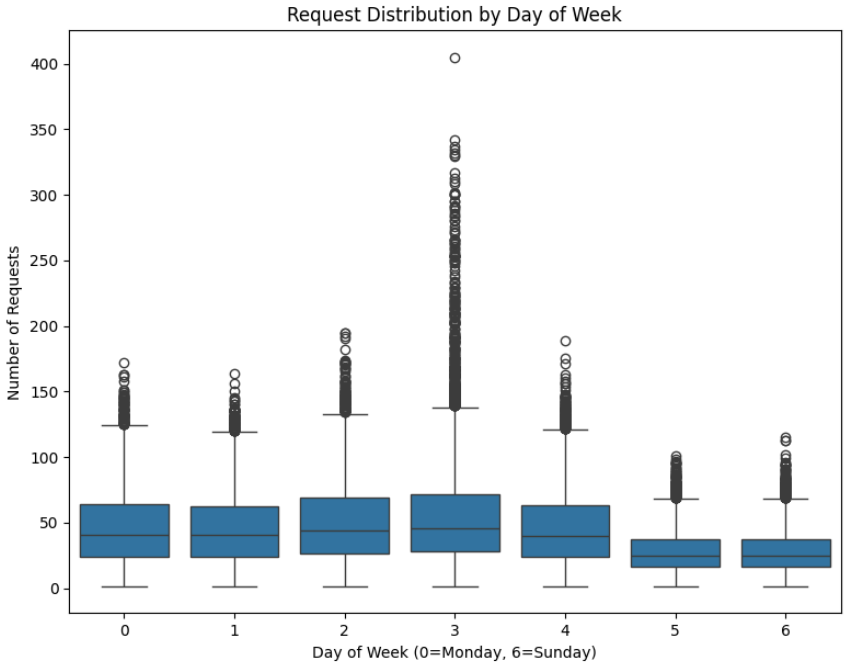

Рис. 2.4. Діаграма розподілу запитів за днями тижня

Робочі дні характеризуються вищою медіаною кількості запитів та більшою варіативністю даних, що відображається у розмірі точок коробкового графіку.

Теплова карта середньої кількості запитів за годинами та днями тижня, дозволяє візуалізувати комплексну картину активності користувачів. Найінтенсивніше використання ресурсів спостерігається в робочі дні з 11 до 15 години, що позначено темно-червоними кольорами на карті. У вихідні дні та в нічний час активність значно знижується, що відображено світлішими відтінками.

Теплова карта середньої кількості запитів за годинами та днями тижня, зображена на рисунку 2.5

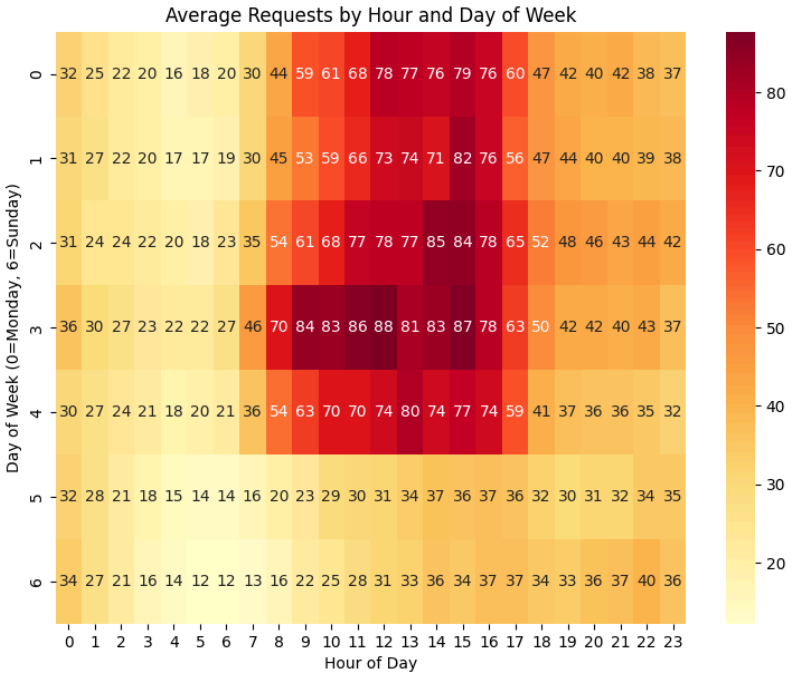

Рис. 2.5. Теплова карта кількості запитів за годинами та днями тижня

Числові значення на тепловій карті додатково підтверджують ці спостереження, демонструючи зміну середньої кількості запитів від приблизно 20-30 у періоди низької активності до 70-80 у пікові години.

2.3. Проектування підсистеми прогнозування навантаження

2.3.1. Архітектура моделі машинного навчання

У даній магістерській дисертації досліджується гібридна архітектура машинного навчання, яка поєднує довгу короткочасну пам'ять (LSTM) та адаптивний випадковий ліс (ARF) для прогнозування часових рядів. LSTM є

різновидом рекурентних нейронних мереж, що має здатність запам'ятовувати довгострокові залежності у послідовних даних. Архітектура LSTM містить спеціальні вентильні механізми, які контролюють потік інформації через мережу. Ці механізми дозволяють мережі вибірково зберігати або забувати інформацію протягом тривалого часу.

Адаптивний випадковий ліс є модифікацією класичного алгоритму випадкового лісу, розробленого для роботи з потоковими даними. ARF автоматично адаптується до змін у розподілі даних шляхом оновлення та заміни дерев рішень у ансамблі. Кожне дерево в ARF навчається на різних підмножинах ознак, що забезпечує різноманітність прогнозів та підвищує стійкість моделі до шуму в даних.

Поєднання цих двох підходів у гібридній архітектурі дозволяє використовувати переваги обох моделей. LSTM ефективно обробляє часові залежності та нелінійні патерни у даних, тоді як ARF забезпечує адаптивність до змін та робастність прогнозування. Взаємодія між компонентами реалізується через механізм зваженого усереднення прогнозів, де ваги динамічно коригуються на основі історичної точності кожної моделі.

Архітектура системи передбачає паралельну обробку вхідних даних обома моделями. LSTM отримує на вхід послідовність значень часового ряду фіксованої довжини та генерує прогноз наступного значення. Одночасно з цим, ARF обробляє поточний стан системи через набір дерев рішень, кожне з яких враховує різні аспекти вхідних даних. Результати обох моделей комбінуються через механізм зваженого усереднення.

Запропонована архітектура демонструє потенціал для підвищення точності прогнозування часових рядів завдяки синергії між глибоким навчанням та ансамблевими методами. LSTM забезпечує ефективне моделювання складних темпоральних залежностей, в той час як ARF додає адаптивність та стійкість до змін у характері даних. Динамічне зважування прогнозів дозволяє

системі автоматично визначати оптимальний баланс між цими підходами в залежності від поточних характеристик часового ряду

У контексті гібридної моделі прогнозування мережа LSTM виконує функцію обробки довгострокових залежностей у часових рядах. Ключовим елементом архітектури LSTM є комірка пам'яті, яка зберігає інформацію про минулі стани системи. Ця комірка містить три основні вентильні механізми: вхідний вентиль, вентиль забування та вихідний вентиль.

При роботі з фіксованою довжиною послідовностей LSTM демонструє високу ефективність завдяки здатності створювати внутрішнє представлення темпоральних патернів. Мережа автоматично визначає, які аспекти минулих станів є релевантними для поточного прогнозу. Це досягається через механізм селективного оновлення стану комірки, де вентиль забування видаляє неактуальну інформацію, а вхідний вентиль додає нові значущі дані.

Адаптивний випадковий ліс є сучасною модифікацією класичного алгоритму випадкового лісу, розробленою спеціально для роботи з потоковими даними. У контексті онлайн-навчання ARF використовує механізм адаптивного оновлення ансамблю дерев рішень для обробки концептуального дрейфу – явища зміни статистичних властивостей цільової змінної з часом. Кожне дерево в ансамблі оновлюється на основі нових спостережень, при цьому використовується алгоритм ADWIN (Adaptive Windowing) для виявлення значущих змін у розподілі даних.

Механізм адаптації ARF базується на підтримці множини дерев різного віку та точності. При надходженні нового спостереження обчислюється зважена помилка прогнозування для кожного дерева:

$$E_i = \sum_{j=1}^{n} w_j \cdot L(y_j, \hat{y_{ij}})$$

,

де E_i – помилка і-го дерева, w_j – вага j-го спостереження, L – функція втрат, y_j – фактичне значення, $\hat{y}_{ij}$ – прогноз і-го дерева для j-го спостереження.

В контексті ансамблевої моделі прогнозування поєднання LSTM та ARF реалізується через механізм динамічного зважування. Основна ідея полягає у використанні адаптивної схеми комбінування прогнозів, яка враховує історичну точність кожної моделі та поточні характеристики часового ряду. Процес інтеграції прогнозів двох моделей можна представити через зважену суму їх виходів:

$$P_t = \alpha_t \cdot P_{LSTM} + (1 - \alpha_t) \cdot P_{ARF},$$

де P_t – фінальний прогноз на момент часу t, P_{LSTM} та P_{ARF} – прогнози відповідних моделей, а α_t – динамічний ваговий коефіцієнт.

2.3.2. Механізм попереднього навчання на історичних даних

Попереднє навчання моделі LSTM на історичних часових рядах є важливим етапом розробки системи прогнозування. Процес починається з підготовки даних, де історичні часові ряди проходять нормалізацію для приведення значень до єдиного масштабу. Наступним кроком є створення послідовностей для навчання моделі. Часовий ряд розбивається на фрагменти фіксованої довжини з перекриттям, де кожен фрагмент містить вхідну послідовність та цільове значення для прогнозування. Довжина вхідної послідовності визначається експериментальним шляхом та залежить від характеру даних та задачі прогнозування. При цьому важливо зберігати часову структуру даних та не допускати змішування послідовностей з різних часових періодів.

Для валідації моделі під час навчання застосовується метод ковзного вікна, коли модель навчається на певному часовому проміжку, а валідація відбувається на наступному проміжку. Це дозволяє оцінити здатність моделі до узагальнення та прогнозування на нових даних. Розмір валідаційної вибірки зазвичай становить 20-30% від загального обсягу даних. Важливо забезпечити,

щоб валідаційна вибірка містила дані з різних часових періодів для перевірки стійкості моделі до сезонних та інших часових патернів.

Архітектура LSTM моделі складається з вхідного шару, одного або декількох шарів LSTM-комірок та вихідного щільного шару. Кількість LSTM-комірок та їх параметри підбираються експериментально в залежності від складності задачі та обсягу навчальних даних [8]. Під час навчання використовується функція втрат середньоквадратичної помилки та оптимізатор Adam з динамічним коефіцієнтом навчання. Для запобігання перенавчання застосовується рання зупинка навчання при відсутності покращення метрик на валідаційній вибірці протягом заданої кількості епох. Процес навчання моделі включає також регуляризацію через застосування dropout між шарами LSTM та L2-регуляризацію вагів моделі. Це допомагає зменшити ефект перенавчання та покращити узагальнюючу здатність моделі. Навчання відбувається міні-батчами, розмір яких підбирається з урахуванням обсягу пам'яті та обчислювальних ресурсів. Для оцінки якості моделі під час навчання використовуються метрики MAE та RMSE на валідаційній вибірці.

Підбір оптимальних параметрів моделі виконується з використанням методу пошуку по сітці або випадкового пошуку. Основними параметрами для налаштування є кількість прихованих шарів LSTM, кількість нейронів у кожному шарі, коефіцієнт відсіву (dropout rate) та параметри оптимізатора. Для кожної конфігурації параметрів проводиться навчання моделі та оцінка її якості на валідаційній вибірці. При цьому враховується не тільки точність прогнозування, але й обчислювальна складність та час навчання моделі [9].

Для оцінки якості прогнозування на історичних даних застосовується техніка ковзного прогнозування, коли модель послідовно робить прогнози на фіксований горизонт часу, зсуваючись вперед на один крок. Це дозволяє оцінити стабільність прогнозів та виявити періоди, де модель працює найкраще або найгірше. Аналіз таких періодів допомагає зрозуміти обмеження моделі та можливі шляхи її покращення.

Додатково проводиться аналіз чутливості моделі до різних характеристик вхідних даних, таких як наявність шуму, пропущені значення та викиди. Це допомагає оцінити надійність моделі в реальних умовах застосування та визначити необхідні попередні етапи обробки даних. При цьому важливо забезпечити баланс між точністю прогнозування та стійкістю моделі до різних типів вхідних даних.

Підготовка історичних даних для попереднього навчання LSTM моделі в контексті прогнозування навантаження має ряд специфічних викликів та особливостей. Першочерговим завданням є обробка пропущених значень у часових рядах, які можуть виникати через збої в системі збору даних, технічні несправності або людський фактор. Для заповнення пропусків використовується метод інтерполяції з урахуванням часової структури даних, де значення розраховуються на основі локального тренду та сезонності [10].

Для виявлення аномалій використовується комбінація статистичних методів та експертних правил, що враховують фізичні обмеження системи навантаження. Виявлені аномалії не видаляються повністю, а позначаються спеціальними маркерами, які модель може використовувати під час навчання для розрізнення нормальних та аномальних патернів поведінки. Дані групуються за часовими вікнами з урахуванням природних циклів навантаження, таких як добові, тижневі та сезонні патерни. Для кожного вікна зберігається контекстна інформація, що включає метадані про погодні умови, календарні події та інші зовнішні фактори, які можуть впливати на характер навантаження.

2.3.3. Механізм покрокового навчання та адаптації моделі

В якості механізму покрокового навчання використовується адаптивний випадковий ліс, що являє собою ефективний алгоритм для обробки потокових даних, який розширює класичний метод випадкового лісу для роботи з динамічними даними у реальному часі. В основі його роботи лежить принцип

онлайн-навчання, коли модель постійно оновлюється з надходженням кожного нового спостереження, що дозволяє адаптуватися до змін у розподілі даних.

Процес оновлення прогнозів у Adaptive Random Forest відбувається наступним чином. При надходженні нового спостереження, воно паралельно обробляється всіма деревами у лісі. Кожне дерево використовує метод онлайн баггінгу (Online Bagging), який визначає вагу спостереження відповідно до розподілу Пуассона. Алгоритм підтримує множину дерев рішень, кожне з яких навчається на різних підмножинах атрибутів, що обираються випадковим чином. При проходженні нового спостереження через дерево, воно оновлює статистику в вузлах та може ініціювати розщеплення листового вузла, якщо виконуються певні умови, наприклад, досягнення мінімальної кількості спостережень у вузлі. Важливою особливістю є те, що алгоритм використовує ковзне вікно для забування старих спостережень, що дозволяє адаптуватися до змін у даних [11].

Механізм виявлення концептуального дрейфу (concept drift) реалізований за допомогою ADWIN. Цей метод відстежує продуктивність кожного дерева та автоматично визначає момент, коли необхідно замінити старе дерево на нове через зміну в розподілі даних. При виявленні дрейфу створюється нове дерево, яке починає навчатися на нових даних, в той час як старе дерево поступово видаляється з ансамблю.

Фінальний прогноз формується шляхом зваженого голосування всіх дерев в ансамблі. Ваги дерев динамічно оновлюються в залежності від їх ефективності на останніх спостереженнях, що дозволяє надавати більшого значення тим деревам, які краще працюють на поточному розподілі даних. Такий підхід забезпечує робастність прогнозування та швидку адаптацію до змін у потокових даних.

Алгоритм демонструє високу обчислювальну ефективність завдяки можливості паралельної обробки дерев та інкрементального оновлення моделі.

Це дозволяє використовувати його в системах, де потрібна обробка великих обсягів даних у реальному часі з обмеженими обчислювальними ресурсами. При цьому зберігається висока точність прогнозування, характерна для ансамблевих методів.

Концептуальний дрейф у прогнозуванні часових рядів представляє собою явище, при якому статистичні властивості цільової змінної, яку намагається передбачити модель, змінюються з часом непередбачуваним чином. У контексті реальних застосувань це може проявлятися як зміна сезонності, тренду, або фундаментальних взаємозв'язків між змінними. Наприклад, у фінансових часових рядах такі зміни можуть бути викликані економічними кризами, змінами ринкових умов або регуляторними змінами.

Для виявлення концептуального дрейфу в Adaptive Random Forest використовується алгоритм ADWIN (Adaptive Windowing), який базується на статистичному аналізі двох "вікон" спостережень. Механізм адаптації ARF до концептуального дрейфу реалізується через систему попереджень та виявлення. Коли алгоритм ADWIN виявляє потенційний дрейф, він спочатку генерує попередження, і модель починає створювати альтернативні дерева рішень. Ці нові дерева навчаються паралельно з існуючими на нових даних, але не беруть участі у формуванні прогнозів. Якщо протягом певного періоду часу продуктивність нових дерев перевищує продуктивність старих, відбувається заміна старих дерев на нові.

Процес адаптації також включає механізм регулювання розміру ансамблю. При виявленні дрейфу може відбуватися як збільшення кількості дерев для кращого охоплення нового розподілу даних, так і видалення застарілих дерев, які більше не відповідають поточним закономірностям у даних. Цей динамічний підхід до управління структурою ансамблю забезпечує баланс між обчислювальною ефективністю та якістю прогнозування. Додатковим аспектом адаптації є використання механізму ковзного вікна змінного розміру. Розмір вікна автоматично регулюється залежно від

стабільності даних – при виявленні дрейфу вікно зменшується для швидшої адаптації до змін, а в періоди стабільності може збільшуватися для накопичення більшої кількості інформації про поточний розподіл даних.

ARF також використовує механізм зважування прогнозів окремих дерев, де ваги динамічно оновлюються на основі їх останньої продуктивності. Дерева, які краще адаптуються до поточного розподілу даних, отримують більшу вагу у формуванні фінального прогнозу. Такий підхід забезпечує плавний перехід між різними режимами роботи моделі при виникненні концептуального дрейфу. У реалізації механізму виявлення дрейфу також враховується можливість виникнення помилкових спрацьовувань. Для цього використовується система порогових значень та періодів підтвердження, що допомагає відрізнити справжній концептуальний дрейф від випадкових флуктуацій у даних.

2.3.4. Конструювання ознак для прогнозування навантаження

Серед темпоральних ознак, що витягуються з часових міток, найбільш інформативними є година доби, яка відображає добові патерни активності користувачів, день тижня для виявлення тижневих циклів, та місяць року для врахування сезонних коливань. Додатково враховуються індикатори вихідних днів та святкових періодів, а також маркери робочих годин. Для врахування циклічної природи часових ознак використовується їх циклічне кодування:

$$x_{sin} = sin(2\pi\frac{x}{max\_value}), x_{cos} = cos(2\pi\frac{x}{max\_value}),$$

де x – значення часової характеристики, max_value – максимальне значення для даної характеристики [12].

Темпоральні ознаки включають:

- Година доби (0-23) – для виявлення добових патернів
- День тижня (1-7) – для виявлення тижневих патернів
- Місяць (1-12) – для виявлення сезонних патернів
- Індикатор вихідного дня (0/1)

- Індикатор святкового дня (0/1)
- Індикатор робочих годин (0/1)
- Номер тижня в році (1-52)
- Квартал (1-4)

Історичні агрегації включають обчислення ковзних середніх з різними вікнами для виявлення трендів різної тривалості. Для короткострокових трендів використовуються вікна від 1 до 24 годин, для середньострокових – від 1 до 7 днів, для довгострокових – від 1 до 12 місяців. Важливими є також статистичні показники останнього періоду: мінімальні та максимальні значення навантаження, стандартне відхилення, медіана. Для врахування динаміки змін обчислюються показники швидкості зміни навантаження як відношення поточних значень до попередніх на різних часових масштабах.

Для покращення якості прогнозування використовуються історичні значення навантаження з фіксованим зсувом у часі. Наприклад, для добових патернів використовуються значення з лагом 24 години, для тижневих – 168 годин. Такі ознаки дозволяють моделі враховувати циклічні патерни, які повторюються з певною періодичністю.

Також до ознак є сенс включити характеристики навантаження у пікові та непікові години. Для пікових годин додатково обчислюються показники тривалості піків, інтенсивності навантаження та швидкості наростання навантаження. У контексті навантаження контейнерів доречно врахувати також системні індикатори, такі як обсяг доступної пам'яті та завантаженість процесора.

Для забезпечення стабільності та інформативності ознак застосовується їх нормалізація та масштабування. Всі числові ознаки приводяться до єдиного масштабу, що запобігає домінуванню окремих ознак у моделі через різницю в порядках величин. Категоріальні ознаки кодуються з використанням методів

one-hot encoding або target encoding, залежно від їх кардинальності та значущості для прогнозування.

2.4. Алгоритм прийняття рішень щодо масштабування

Процес прийняття рішень розпочинається зі збору метрик з системи моніторингу Prometheus. Система отримує часовий ряд значень метрик за останній певний період часу. Далі ці дані передаються до моделі машинного навчання, яка генерує прогноз майбутнього навантаження. На основі отриманого прогнозу система обчислює необхідну кількість реплік для забезпечення оптимальної продуктивності системи.

Розрахунок необхідної кількості реплік базується на формулі, яка враховує прогнозоване навантаження та встановлений поріг навантаження на одну репліку:

$$R = \left| \frac{P}{T} \right|,$$

де R – необхідна кількість реплік, P – прогнозоване значення метрики навантаження, T – пороговий рівень навантаження на одну репліку, при цьому відношення P до T округлюється до найближчого більшого цілого числа.

Для запобігання частим коливанням кількості реплік система використовує гістерезис. Масштабування у бік зменшення кількості реплік відбувається лише коли навантаження на одну репліку падає нижче 60% від порогового значення. Це створює буферну зону, яка дозволяє системі залишатися стабільною при незначних коливаннях навантаження. Система також враховує обмеження на мінімальну та максимальну кількість реплік, які встановлюються адміністратором системи відповідно до наявних ресурсів та вимог до відмовостійкості. Мінімальна кількість реплік забезпечує базовий рівень доступності сервісу, тоді як максимальна кількість реплік запобігає надмірному споживанню ресурсів кластера.

Після прийняття рішення про необхідність масштабування система взаємодіє з Kubernetes API для зміни кількості реплік цільового розгортання. Процес масштабування відбувається поступово, що дозволяє системі адаптуватися до змін навантаження без різких перепадів у продуктивності. Система також зберігає інформацію про кожну операцію масштабування, включаючи час, кількість реплік та значення метрик, що використовувались для прийняття рішення.

У випадку виникнення помилок при отриманні метрик або при взаємодії з Kubernetes API система використовує механізм повторних спроб з експоненціальною затримкою. Це забезпечує стійкість системи до тимчасових збоїв у роботі компонентів інфраструктури та запобігає втраті контролю над масштабуванням.

2.5. Метод автоматичного визначення параметрів прогнозування

2.5.1. Алгоритм обчислення оптимального розміру вікна

У контексті гібридної моделі, що поєднує попередньо навчену LSTM мережу та онлайн-навчання за допомогою Adaptive Random Forest регресії, постає питання визначення оптимального розміру вікна для обробки вхідних даних. Алгоритм обчислення оптимального розміру вікна базується на динамічній адаптації розміру вікна з урахуванням концептуального дрейфу даних та якості прогнозування моделі.

Основним принципом роботи алгоритму є безперервний моніторинг похибки прогнозування та автоматичне коригування розміру вікна на основі детектування змін у структурі вхідних даних. Алгоритм використовує детектор концептуального дрейфу ADWIN, який аналізує послідовність похибок прогнозування та сигналізує про наявність значущих змін у даних. При виявленні концептуального дрейфу розмір вікна зменшується для швидшої

адаптації до нових патернів у даних. У періоди стабільної роботи моделі розмір вікна поступово збільшується для врахування довгострокових залежностей.

Математично процес адаптації розміру вікна можна описати наступними формулами. У випадку виявлення дрейфу:

$$w_{t+1} = max(w_{min}, 0.5 \cdot w_t)$$

При стабільній роботі, якщо середня похибка зменшується:

$$w_{t+1} = min(w_{max}, 1.1 \cdot w_t)$$

В інших випадках:

$$w_{t+1} = w_t,$$

де w_t – поточний розмір вікна, w_{min} та w_{max} – мінімальний та максимальний допустимі розміри вікна.

Алгоритм передбачає наявність буферу останніх похибок прогнозування фіксованого розміру для обчислення статистик та прийняття рішень щодо зміни розміру вікна. При ініціалізації встановлюється мінімальний розмір вікна, який поступово збільшується в процесі роботи системи за умови стабільної якості прогнозування. У випадку виявлення концептуального дрейфу відбувається швидке зменшення розміру вікна для забезпечення швидкої адаптації до змін.

Запропонований алгоритм не потребує додаткового навчання чи налаштування гіперпараметрів, окрім встановлення початкових значень мінімального та максимального розміру вікна. Ці значення можуть бути визначені на основі специфіки предметної області та вимог до швидкості адаптації системи. Алгоритм органічно інтегрується з архітектурою гібридної моделі, забезпечуючи автоматичне налаштування розміру вікна в процесі онлайн-навчання та прогнозування. У процесі роботи алгоритму відбувається природна регуляція між швидкістю адаптації моделі та її здатністю враховувати довгострокові залежності у даних. При появі нових патернів у даних швидке

зменшення розміру вікна дозволяє моделі швидко адаптуватися до змін, тоді як в періоди стабільності поступове збільшення вікна дає можливість враховувати більш довгострокові залежності для покращення якості прогнозування.

2.5.2. Алгоритм визначення інтервалу прогнозування

Алгоритм визначення інтервалу прогнозування базується на динамічному аналізі часу готовності контейнерів до обробки запитів, враховуючи специфіку розгортання в кластері Kubernetes. Інтервал прогнозування визначається на основі метрики часу готовності контейнера, яка обчислюється як період від початку створення поду до моменту успішного проходження всіх перевірок готовності (readiness probe). Цей час включає в себе завантаження образу контейнера, його ініціалізацію та перехід у стан готовності до обробки запитів.

$$T_{forecast} = max(T_{ready} \cdot 1.2, T_{min}),$$

де $T_{forecast}$ – інтервал прогнозування, T_{ready} – усереднений час готовності контейнера, T_{min} – мінімально допустимий інтервал прогнозування. Коефіцієнт 1.2 додає 20% запасу часу для врахування можливих затримок при розгортанні.

Kubernetes оператор здійснює постійний моніторинг метрик часу готовності контейнерів через Kubernetes API. Для кожного нового поду фіксується час від створення до переходу в стан Ready. Ці дані накопичуються у внутрішньому буфері оператора та використовуються для обчислення усередненого значення часу готовності. При цьому враховуються останні N розгортань, де N є конфігурованим параметром, що дозволяє адаптувати чутливість алгоритму до змін у системі.

Процес визначення інтервалу прогнозування включає декілька етапів валідації та обробки даних. При отриманні нових метрик про готовність контейнера оператор виконує фільтрацію аномальних значень, які можуть виникати через мережеві затримки або проблеми з інфраструктурою. Відфільтровані дані використовуються для оновлення ковзного середнього часу готовності, на основі якого розраховується новий інтервал прогнозування.

Алгоритм враховує специфіку різних типів контейнерів та їх вимоги до ініціалізації. Для контейнерів з важкою ініціалізацією, наприклад тих, що потребують завантаження великих моделей машинного навчання або встановлення складних залежностей, час готовності може бути значно більшим. У таких випадках алгоритм автоматично коригує інтервал прогнозування, забезпечуючи достатній запас часу для повного розгортання та ініціалізації.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі магістерської дисертації було детально розглянуто та описано проектування системи автоматичного масштабування контейнерів на основі машинного навчання. Архітектура системи побудована з урахуванням сучасних вимог до масштабованості та надійності контейнеризованих застосунків. Вона включає компоненти для збору метрик, їх аналізу, прогнозування навантаження та прийняття рішень щодо масштабування.

Було сформовано навчальний датасет веб-запитів, який містить історичні дані про навантаження на систему. Датасет включає часові ряди з метриками використання ресурсів контейнерів та параметрами вхідного трафіку. Проведений аналіз даних дозволив виявити основні патерни навантаження та їх періодичність, що є критичним для точності подальшого прогнозування.

Підсистема прогнозування навантаження базується на гібридній моделі машинного навчання, яка поєднує архітектуру LSTM та ARF. Така комбінація дозволяє ефективно обробляти як короткострокові, так і довгострокові залежності в даних. LSTM відповідає за виявлення складних часових патернів, тоді як ARF забезпечує адаптивність до раптових змін у навантаженні. Механізм попереднього навчання використовує накопичені історичні дані для формування базової моделі, тоді як механізм покрокового навчання забезпечує її адаптацію до змін у характері навантаження через оновлення параметрів обох компонентів моделі. Розроблений підхід до конструювання ознак дозволяє ефективно представити часові залежності та сезонні коливання у даних.

Алгоритм прийняття рішень щодо масштабування враховує як поточний стан системи, так і прогнозовані значення навантаження. Він використовує набір правил та порогових значень для визначення необхідності зміни кількості контейнерів. Метод автоматичного визначення параметрів прогнозування включає алгоритми для обчислення оптимального розміру вікна спостереження та інтервалу прогнозування.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

3.1. Розробка Kubernetes оператора автоматичного масштабування

Розробка Kubernetes оператора автоматичного масштабування базується на використанні фреймворку Kopf (Kubernetes Operator Pythonic Framework), який надає зручний інтерфейс для створення операторів на мові Python. Даний оператор призначений для автоматичного масштабування деплойментів (Deployment) на основі прогнозованих значень метрик, отриманих за допомогою машинного навчання.

Архітектура оператора складається з двох основних функцій-обробників подій. Перший обробник активується при створенні нового ресурсу типу MLScaler і встановлює початкові параметри масштабування, включаючи мінімальну кількість реплік та час останнього масштабування. Фрагмент коду першого обробнику зображено на рисунку 3.1. Другий обробник працює як періодичний таймер, який запускається кожні 60 секунд та виконує основну логіку масштабування.

```
23  @kopf.timer('example.com', 'v1', 'mlscalers', interval=60.0)
24  async def scale_handler(spec, status, name, namespace, logger, **kwargs):
25      try:
26          model_service_url = spec.get('modelServiceUrl')
27          prometheus_url = spec.get('prometheusUrl')
28          metric_query = spec.get('metricQuery')
29          target_deployment = spec.get('targetDeployment')
30          min_replicas = spec.get('minReplicas', 1)
31          max_replicas = spec.get('maxReplicas', 10)
32          metric_threshold = spec.get('metricThreshold', 100)
```

Рис. 3.1. Лістинг коду, що відповідає за обробку параметрів

У процесі роботи оператор взаємодіє з Prometheus для отримання історичних даних метрик за останні 30 хвилин. Ці дані передаються до окремого сервісу машинного навчання, який виконує тренування моделі та надає прогноз майбутнього навантаження. На основі отриманого прогнозу оператор обчислює необхідну кількість реплік, враховуючи задані обмеження мінімальної та максимальної кількості реплік.

Алгоритм розрахунку кількості реплік базується на порівнянні прогнозованого навантаження на одну репліку з заданим пороговим значенням. Якщо навантаження перевищує поріг, кількість реплік збільшується пропорційно до перевищення. При зниженні навантаження нижче 60% від порогового значення відбувається зменшення кількості реплік. У випадку, коли навантаження знаходиться в межах допустимого діапазону, кількість реплік залишається незмінною.

Для взаємодії з Kubernetes API оператор використовує офіційний клієнт kubernetes-client, який дозволяє отримувати інформацію про поточний стан розгортань та виконувати їх масштабування. Оператор працює в межах кластера Kubernetes, використовуючи автоматично налаштовані облікові дані сервісного акаунта. При виникненні помилок під час масштабування оператор використовує механізм тимчасових помилок Kopf для повторних спроб виконання операції через визначений інтервал часу [13].

Фрагмент коду, що підвищує кількість реплік у ресурсі Deployment зображений на рисунку 3.2.

```python
def scale_deployment(deployment, namespace, replicas):
    try:
        patch = {
            "spec": {
                "replicas": replicas
            }
        }

        v1.patch_namespaced_deployment(
            name=deployment,
            namespace=namespace,
            body=patch
        )
        logging.info(f"Scaled deployment {deployment} to {replicas} replicas")
    except kubernetes.client.rest.ApiException as e:
        logging.error(f"Error scaling deployment: {e}")
```

Рис. 3.2. Лістинг коду, що відповідає за масштабування контейнерів

Реалізація включає механізми логування всіх важливих подій та операцій, що дозволяє відстежувати роботу оператора та діагностувати можливі проблеми. Стан масштабування зберігається у полі status користувацького ресурсу, що містить інформацію про останнє масштабування, поточну кількість реплік, останній прогноз та використане порогове значення.

В рамках розробки Kubernetes оператора автоматичного масштабування було створено визначення користувацького ресурсу (CRD) для забезпечення можливості декларативного опису параметрів масштабування. Даний CRD визначає новий тип ресурсу MLScaler, який працює в межах окремих просторів імен кластера.

Структура користувацького ресурсу складається з двох основних секцій: специфікації (spec) та статусу (status). Специфікація містить обов'язкові поля для налаштування роботи оператора, такі як адреса сервісу машинного навчання (modelServiceUrl), адреса Prometheus (prometheusUrl), запит для отримання метрик (metricQuery) та назва цільового ресурсу для масштабування (targetDeployment).

Додатково специфікація включає опціональні параметри для тонкого налаштування процесу масштабування. Серед них – мінімальна (minReplicas) та максимальна (maxReplicas) кількість реплік, які обмежують діапазон можливого масштабування. Порогове значення метрики (metricThreshold) встановлює максимально допустиме навантаження на одну репліку.

Секція статусу використовується оператором для збереження поточного стану масштабування. Вона містить час останнього масштабування (lastScaling), поточну кількість реплік (currentReplicas), останнє прогнозоване значення метрики (lastPrediction) та використане порогове значення (thresholdUsed). Ця інформація дозволяє відстежувати історію масштабування та поточний стан системи. Лістинг коду користувацького ресурсу зображено на рисунку 3.3.

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: mlscalers.example.com
spec:
  group: example.com
  names:
    kind: MLScaler
    plural: mlscalers
    singular: mlscaler
  scope: Namespaced
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              required:
                - modelServiceUrl
                - prometheusUrl
                - metricQuery
                - targetDeployment
              properties:
                modelServiceUrl:
                  type: string
                prometheusUrl:
                  type: string
                metricQuery:
                  type: string
                targetDeployment:
                  type: string
                minReplicas:
                  type: integer
                  minimum: 1
                maxReplicas:
                  type: integer
                  minimum: 1
                metricThreshold:
                  type: integer
                  minimum: 1
                  description: "Maximum load per replica before scaling up"
```

Рис. 3.3. Лістинг коду користувацького ресурсу MLScaler

Схема валідації, визначена за допомогою OpenAPI v3, забезпечує контроль коректності значень полів при створенні та модифікації ресурсів MLScaler. Зокрема, встановлено обмеження на мінімальні значення для числових параметрів, що запобігає встановленню некоректних значень, які могли б призвести до неправильної роботи оператора.

3.2. Реалізація REST API для взаємодії з моделлю прогнозування

Для забезпечення взаємодії з моделлю нейронної мережі було розроблено REST API, яке надає зручний інтерфейс для керування процесами прогнозування та навчання. API реалізовано з використанням FastAPI, що забезпечує високу продуктивність та автоматичну генерацію документації. Список ендпоінтів API описаний у таблиці 3.2.

Таблиця 3.2 – Ендпоінти API до нейромережевої моделі

Ендпоінт	Метод	Опис
/train	POST	Запуск навчання моделі
/predict	POST	Отримання прогнозу
/health	GET	Перевірка доступності моделі

Виклики API з оператору зображені на рисунку 3.4.

```
57        requests.post(
58            f"{model_service_url}/train",
59            json={"data": data_points},
60            headers={"Content-Type": "application/json"}
61        )
62
63        predict_response = requests.post(
64            f"{model_service_url}/predict",
65            json={"data": data_points},
66            headers={"Content-Type": "application/json"}
67        )
68
```

Рис. 3.4. Лістинг коду, що відповідає за виклики API

Для взаємодії з API розроблено механізми валідації вхідних даних та обробки помилок. Кожен запит перевіряється на коректність формату та наявність усіх необхідних параметрів. У випадку помилок система повертає детальні повідомлення, що допомагають в діагностиці проблем.

3.3. Реалізація та навчання моделі прогнозування

3.3.1. Імплементація LSTM моделі

Основою системи є фреймворк TensorFlow, який забезпечує ефективну роботу з нейронними мережами та надає широкий спектр інструментів для їх навчання. Для обробки та підготовки даних застосовується бібліотека pandas, що дозволяє ефективно працювати з часовими рядами та виконувати необхідні трансформації даних. Бібліотека numpy використовується для оптимізованих обчислень та маніпуляцій з багатовимірними масивами даних. Архітектура моделі реалізована з використанням високорівневого API Keras, що входить до складу TensorFlow. Це забезпечує зручний інтерфейс для конструювання складних нейромережевих архітектур та керування процесом навчання. Для масштабування даних використовується модуль preprocessing з scikit-learn, який надає реалізацію різних методів нормалізації та стандартизації даних.

Основним компонентом є клас LoadPredictor, який використовує методи машинного навчання для передбачення часових рядів різних показників навантаження системи. Архітектура предиктора базується на підготовці вхідних даних з урахуванням темпоральних особливостей навантаження. Модель опрацьовує часові ряди, де кожен запис містить мітку часу та значення цільової метрики. Для покращення якості прогнозування було впроваджено додаткові часові характеристики, такі як година доби, день тижня, позначення робочих годин та пікових періодів активності. Окремо виділяються ранкові години та вихідні дні, що дозволяє моделі краще адаптуватися до типових патернів навантаження різних типів систем.

Фрагмент коду, що відповідає за генерацію додаткових ознак часових рядів, зображено на рисунку 3.5.

```
df['hour'] = df.index.hour
df['day_of_week'] = df.index.dayofweek

df['is_business_hours'] = ((df['hour'] >= 8) & (df['hour'] <= 17)).astype(float)

df['is_peak_hours'] = ((df['hour'] >= 10) & (df['hour'] <= 15)).astype(float)

df['is_weekend'] = (df['day_of_week'] >= 5).astype(float)

df['is_early_morning'] = (df['hour'] <= 6).astype(float)

df['scaled_count'] = self.scaler.fit_transform(df[['count']])

df['hour_norm'] = df['hour'] / 23.0
df['day_norm'] = df['day_of_week'] / 6.0

features = ['scaled_count', 'hour_norm', 'day_norm',
            'is_business_hours', 'is_peak_hours',
            'is_weekend', 'is_early_morning']
```

Рис. 3.5. Лістинг коду, що відповідає за генерацію додаткових ознак часових рядів

Підготовка даних включає нормалізацію числових значень для забезпечення стабільності навчання нейронної мережі. Використовується масштабування MinMax для приведення значень метрик до стандартизованого діапазону, а також нормалізація темпоральних характеристик. Дані організовуються у послідовності фіксованої довжини, що дозволяє моделі враховувати історичний контекст при формуванні прогнозів незалежно від типу метрики, що аналізується.

Нейронна мережа побудована на основі архітектури LSTM, що оптимально підходить для обробки часових рядів будь-якої природи. Модель складається з двох LSTM шарів з активацією ReLU та проміжними шарами виключення для запобігання перенавчання. Фінальна частина мережі включає повнозв'язні шари для обробки витягнутих часових характеристик та формування кінцевого прогнозу.

Метод класу, що відповідає за створення моделі, зображено на рисунку 3.6.

```
def create_model(self):
    """Create model with architecture optimized for HTTP request patterns"""
    model = Sequential([
        LSTM(64, activation='relu', return_sequences=True,
             input_shape=(self.sequence_length, 7)),
        Dropout(0.2),

        LSTM(32, activation='relu'),
        Dropout(0.2),

        Dense(32, activation='relu'),
        Dense(16, activation='relu'),
        Dense(1)
    ])

    optimizer = Adam(learning_rate=0.0005)

    model.compile(optimizer=optimizer,
                  loss='mse',
                  metrics=['mae'])

    self.model = model
    return model
```

Рис. 3.6. Лістинг коду, що відповідає за створення LSTM моделі

Процес навчання моделі оптимізовано для роботи з різноманітними патернами системних метрик. Використовується оптимізатор Adam з низькою швидкістю навчання для забезпечення стабільної збіжності. Впроваджено механізми раннього зупинення та динамічного зменшення швидкості навчання при досягненні плато, що дозволяє знаходити оптимальні параметри моделі та уникати перенавчання. Дані розділяються на навчальну та валідаційну вибірки для контролю якості навчання та забезпечення генералізації моделі на різних типах метрик.

3.3.2. Попереднє навчання моделі на історичних даних

Було проведено навчання моделі на датасеті, описаному у другому розділі дисертації. Процес навчання тривав 20 епох, протягом яких модель обробила

значний обсяг даних, розділених на навчальний та валідаційний набори. Аналіз процесу навчання показує стабільне покращення показників точності моделі. Початкове значення функції втрат становило 0.0024, а середня абсолютна помилка (MAE) – 0.0351. Протягом навчання ці показники послідовно покращувались, досягнувши на останній епосі значень 0.0014 для функції втрат та 0.0278 для MAE. Валідаційні метрики також демонструють позитивну динаміку, що свідчить про відсутність перенавчання моделі. Фінальне тестування моделі на окремому наборі даних, що складався з 509 пакетів, підтвердило стабільність досягнутих результатів. Час обробки тестових даних становив в середньому 7 мілісекунд на батч, що свідчить про можливість використання моделі в режимі реального часу для прогнозування навантаження та прийняття рішень щодо масштабування системи.

Графік, зображений на рисунку 3.7, відображає порівняння реальних та прогнозованих значень RPM (запитів за хвилину) протягом часу.

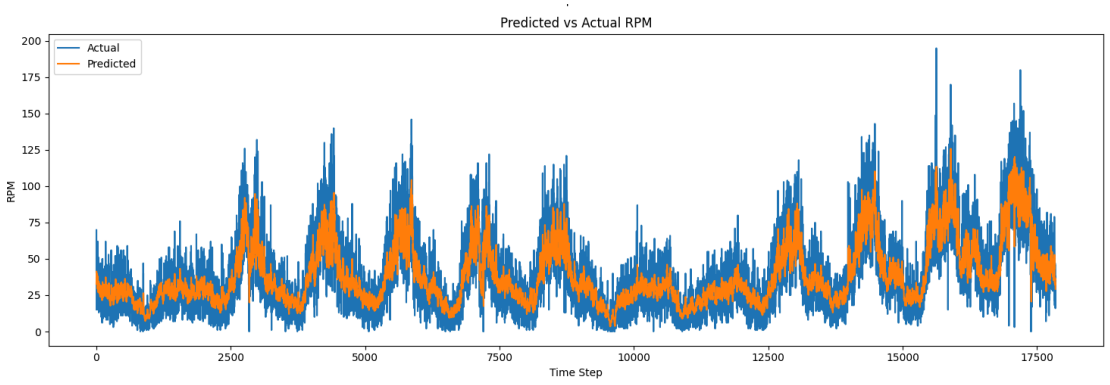

Рис. 3.7. Діаграма порівняння реальних та прогнозованих значень RPM

Синя лінія представляє фактичні значення метрики, а помаранчева – передбачені моделлю. На графіку чітко простежуються добові та тижневі патерни навантаження, які модель успішно відтворює. Модель демонструє здатність прогнозувати як загальні тренди, так і локальні піки навантаження,

хоча спостерігається певне згладжування екстремальних значень. Діапазон значень RPM коливається від близько 10 до 150 запитів за хвилину, причому модель найкраще відтворює середні значення та основні патерни змін навантаження. Часовий ряд охоплює приблизно 17500 кроків, що відповідає тривалому періоду спостережень та забезпечує достатню репрезентативність даних для навчання та валідації моделі.

3.3.2. Імплементація механізму покрокового навчання

У процесі розробки системи автоматичного масштабування для Kubernetes кластера було створено гібридний предиктор навантаження, який поєднує два підходи машинного навчання: глибинне навчання на основі LSTM та покрокове навчання з використанням адаптивного випадкового лісу з бібліотеки River [14].

Основною архітектурною складовою предиктора є клас HybridLoadPredictor, який інкапсулює логіку обох моделей. Адаптивний випадковий ліс налаштовано на використання 50 дерев рішень з періодом оновлення 50 спостережень та коефіцієнтом використання ознак 0.8.

Процес оновлення моделі зосереджено виключно на покроковому навчання, що дозволяє системі швидко адаптуватися до змін у характері навантаження. Для цього використовується метод update_model, який приймає нові дані та оновлює модель River у режимі онлайн-навчання. Кожне нове спостереження перетворюється у набір ознак, що включає попередні значення часового ряду, та використовується для навчання моделі за допомогою методу learn_one.

Прогнозування майбутніх значень навантаження здійснюється обома моделями паралельно. LSTM модель використовує нормалізовані послідовності даних для генерації прогнозів, тоді як адаптивний випадковий ліс працює з необробленими даними, перетвореними у формат часових лагів. Система також

обчислює показник впевненості у прогнозі на основі середньої абсолютної помилки адаптивного випадкового лісу.

Для забезпечення надійності та можливості відновлення роботи після перезапуску реалізовано механізми збереження та завантаження стану обох моделей. LSTM модель зберігається у форматі JSON для архітектури та H5 для ваг нейронної мережі, тоді як модель River серіалізується за допомогою бібліотеки joblib. Додатково зберігаються параметри нормалізації даних та загальні налаштування предиктора.

3.4. Розгортання системи масштабування в тестовому середовищі

У процесі розробки системи автоматичного масштабування навчальних моделей було створено тестове середовище для перевірки функціональності та взаємодії всіх компонентів системи. Тестове середовище розгорнуто на базі Google Cloud, зокрема з використанням сервісу Google Kubernetes Engine (GKE). Даний вибір зумовлений тим, що GKE надає повністю керований Kubernetes кластер з можливістю швидкого розгортання та налаштування всіх необхідних компонентів інфраструктури.

В рамках тестового середовища було розгорнуто Kubernetes кластер, який складається з трьох вузлів. На даних вузлах функціонують основні компоненти системи масштабування: Kubernetes оператор, який відповідає за автоматизацію процесів масштабування, та тестовий застосунок, який симулює робоче навантаження на систему. Всі контейнерні образи компонентів системи завантажено до публічного реєстру Docker Hub, що забезпечує зручний доступ до них під час розгортання. Для збору метрик про стан системи було встановлено Prometheus сервер, який забезпечує моніторинг ресурсів та продуктивності всіх компонентів.

Модель машинного навчання та відповідний API для взаємодії з нею розміщено на окремій віртуальній машині в межах того ж проекту GCP. Таке

розділення компонентів дозволяє ізолювати обчислювальні ресурси для навчання моделі від ресурсів кластера, що забезпечує стабільність роботи системи масштабування. API моделі реалізує покрокове навчання, що дає можливість контролювати процес навчання та збирати метрики про його прогрес.

Розгортання оператора потребує створення декількох ресурсів Kubernetes, які забезпечують його коректну роботу та надають необхідні права доступу до кластера. Ресурс Deployment оператора зображений на рисунку 3.8.

```
39  ---
40  apiVersion: apps/v1
41  kind: Deployment
42  metadata:
43    name: mlscaler-operator
44  spec:
45    replicas: 1
46    selector:
47      matchLabels:
48        app: mlscaler-operator
49    template:
50      metadata:
51        labels:
52          app: mlscaler-operator
53      spec:
54        serviceAccountName: mlscaler-operator
55        containers:
56        - name: operator
57          image: pienskoi/operator:latest
58          imagePullPolicy: Always
59          env:
60          - name: KOPF_NAMESPACE
61            value: default
62
```

Рис. 3.8. Лістинг коду ресурсу Deployment для розгортання оператора

Для забезпечення належного рівня безпеки та розмежування прав доступу було створено окремий ServiceAccount з назвою "mlscaler-operator". ClusterRole "mlscaler-operator" визначає набір дозволів для роботи з різними ресурсами

кластера. Оператору надано права на операції з базовими ресурсами Kubernetes, такими як поди та сервіси, а також з розгортаннями з групи apps. Окремо визначено права для роботи з користувацькими ресурсами, що включають ресурси mlscalers та їх статуси. Оператор також має можливість переглядати визначення користувацьких ресурсів та працювати з ними. ClusterRoleBinding пов'язує створений ClusterRole з ServiceAccount, що надає оператору визначені права доступу в межах всього кластера. Цей зв'язок встановлюється для ServiceAccount "mlscaler-operator" у просторі імен "default". Сам оператор розгортається як Deployment з однією реплікою. У специфікації поду вказано використання створеного ServiceAccount та образу контейнера "pienskoi/operator:latest". Код даних ресурсів зображений на рисунку 3.9.

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mlscaler-operator
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: mlscaler-operator
rules:
- apiGroups: [""]
  resources: ["pods", "services", "events"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["example.com"]
  resources: ["mlscalers", "mlscalers/status", "mlscalers/finalizers"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["apiextensions.k8s.io"]
  resources: ["customresourcedefinitions"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["coordination.k8s.io"]
  resources: ["leases"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: mlscaler-operator
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: mlscaler-operator
subjects:
- kind: ServiceAccount
  name: mlscaler-operator
  namespace: default
```

Рис. 3.9. Лістинг коду RBAC ресурсів для розгортання оператора

Для тестування системи автоматичного масштабування було розроблено тестовий застосунок на основі фреймворку FastAPI. Застосунок реалізує декілька HTTP ендпоінтів з різними характеристиками навантаження та інтегрований з системою моніторингу Prometheus для збору метрик про його роботу.

Тестовий застосунок розгортається в Kubernetes кластері та надає чотири основні ендпоінти для симуляції різних сценаріїв роботи. Кореневий ендпоінт "/" симулює звичайні запити з випадковою затримкою від 0.1 до 0.5 секунд. Ендпоінт "/slow" імітує довготривалі операції з затримкою від 0.5 до 2 секунд, що дозволяє тестувати поведінку системи при обробці ресурсоємних запитів. Ендпоінт "/error" моделює нестабільну роботу застосунку з ймовірністю виникнення помилки 30%, що дає можливість перевірити реакцію системи масштабування на проблемні ситуації.

Для збору метрик про роботу застосунку реалізовано інтеграцію з Prometheus. Визначено два основні типи метрик: лічильник загальної кількості HTTP запитів та гістограма часу відповіді на запити. Лічильник http_requests_total відстежує кількість запитів з розподілом за методом, ендпоінтом та статусом відповіді. Гістограма http_request_duration_seconds вимірює час обробки запитів для кожного ендпоінту. Метрики доступні через спеціальний ендпоінт "/metrics" у форматі, який підтримується Prometheus.

Застосунок розроблено з використанням асинхронного підходу, що забезпечує ефективну обробку паралельних запитів. Для запуску використовується ASGI-сервер Uvicorn, який налаштовано для прослуховування всіх мережевих інтерфейсів на порту 8000. Контейнерний образ застосунку опубліковано у реєстрі Docker Hub для зручного розгортання в Kubernetes кластері.

Даний тестовий застосунок дозволяє моделювати різні патерни навантаження та збирати детальні метрики про його роботу, що необхідно для тестування та налагодження системи автоматичного масштабування. Завдяки різним типам ендпоінтів можна симулювати реалістичні сценарії використання та перевіряти коректність роботи алгоритмів масштабування при різних умовах навантаження.

Список усіх розгорнутих контейнерів у GKE кластері зображено на рисунку 3.10.

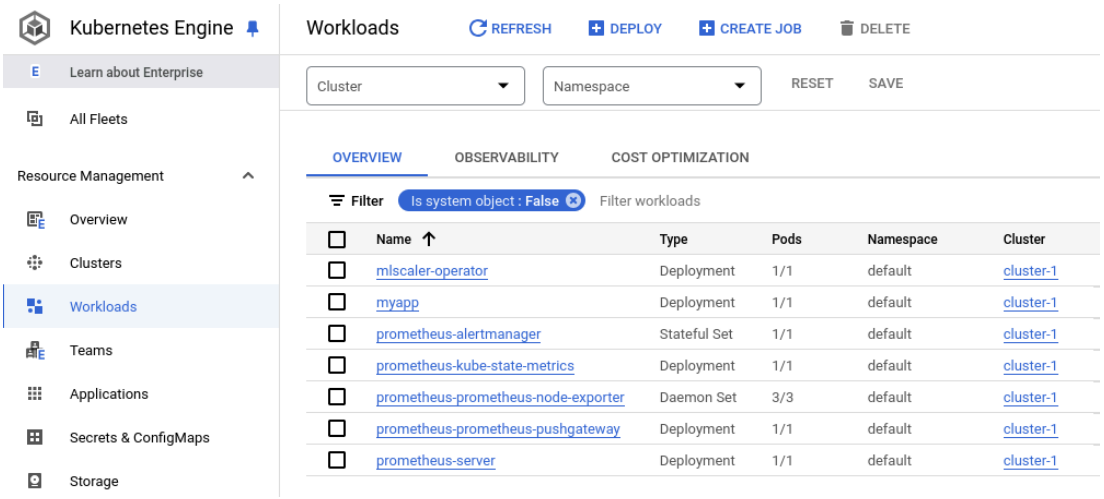

Рис. 3.10. Список розгорнутих контейнерів у тестовому GKE кластері

Для активації системи автоматичного масштабування в тестовому середовищі було розгорнуто користувацький ресурс MLScaler, який описує конфігурацію та параметри масштабування для тестового застосунку. У специфікації ресурсу визначено URL-адресу сервісу з моделлю машинного навчання, який розгорнуто на окремій віртуальній машині в GCP з публічною IP-адресою. Також вказано адресу Prometheus серверу, який доступний в кластері через сервіс prometheus-server. Для отримання метрик використовується PromQL запит, який обчислює швидкість надходження HTTP запитів до застосунку за останні 5 хвилин.

У конфігурації вказано назву цільового розгортання "myapp", яке буде масштабуватися системою. Встановлено обмеження на мінімальну кількість реплік застосунку – 1, та максимальну – 10. Порогове значення метрики встановлено на рівні 100 запитів за одиницю часу, при перевищенні якого система буде приймати рішення про необхідність масштабування. Код розгорнутого користувацького ресурсу зображено на рисунку 3.11.

```yaml
apiVersion: example.com/v1
kind: MLScaler
metadata:
  name: myapp-scaler
spec:
  modelServiceUrl: "http://model-vm:8000"
  prometheusUrl: "http://prometheus-server:80"
  metricQuery: "rate(http_requests_total{app='myapp'}[5m])"
  targetDeployment: "myapp"
  minReplicas: 1
  maxReplicas: 10
  metricThreshold: 100
```

Рис. 3.11. Лістинг коду тестового користувацького ресурсу MLScaler

Після застосування цього користувацького ресурсу в кластері, оператор починає відстежувати стан системи та автоматично керувати кількістю реплік цільового застосунку на основі даних моніторингу та прогнозів моделі машинного навчання.

3.5. Експериментальні дослідження системи масштабування

3.5.1. Експеримент з поступовим зростанням навантаження

В межах експериментальної перевірки роботи системи автоматичного масштабування було проведено тестування з поступовим зростанням

навантаження протягом 50 хвилин. Для створення навантаження використовувався інструмент Apache Benchmark, який надсилав HTTP-запити до тестового застосунку з різною інтенсивністю.

Експеримент складався з трьох основних фаз. Протягом перших 15 хвилин підтримувалось стабільне базове навантаження близько 50 запитів на хвилину з невеликими природними коливаннями. У наступні 15 хвилин спостерігалось помірне лінійне зростання навантаження зі швидкістю приблизно 6 додаткових запитів на хвилину. В останні 20 хвилин експерименту темп зростання навантаження збільшився до 8 додаткових запитів на хвилину.

Система автоматичного масштабування продемонструвала предиктивну поведінку, аналізуючи тренди метрик та виконуючи масштабування до того, як навантаження досягало критичних значень. При наближенні метрики до порогового значення 100 запитів на хвилину для поточної кількості реплік, система збільшувала кількість реплік застосунку.

Результати експерименту були візуалізовані на графіку, зображеному на рисунку 3.12, який відображає зміну кількості запитів у часі та моменти масштабування системи.

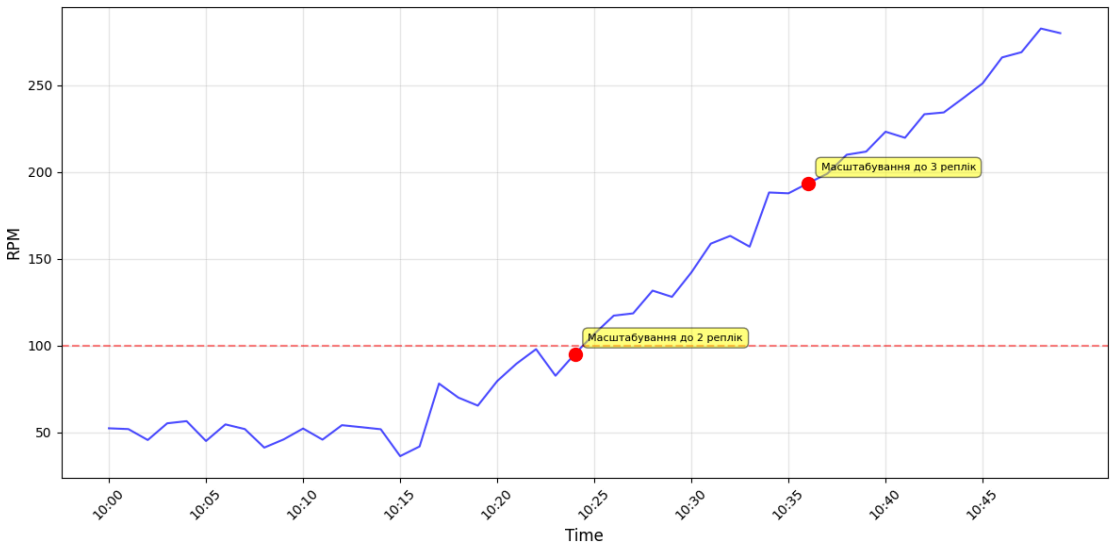

Рис. 3.12. Графік експерименту з поступовим зростанням навантаження

На графіку чітко видно точки, де система приймала рішення про збільшення кількості реплік, що відбувалось до досягнення метрикою критичних значень. Це підтверджує ефективність предиктивного підходу до масштабування, який дозволяє системі завчасно реагувати на зростання навантаження та підтримувати стабільну роботу застосунку.

3.5.2. Експеримент з різкими стрибками навантаження

В межах тестування системи автоматичного масштабування було проведено експеримент із раптовими стрибками навантаження. На відміну від попереднього експерименту з поступовим зростанням, цей тест був спрямований на перевірку реакції системи на різкі зміни у кількості запитів.

Експеримент тривав 50 хвилин та включав три чіткі стрибки навантаження. Перший стрибок відбувся на 10-й хвилині, коли кількість запитів різко зросла до 180 запитів на хвилину. Система масштабування миттєво відреагувала на це зростання, збільшивши кількість реплік до трьох. Після п'яти хвилин зниженого навантаження, на 15-й хвилині, система повернулась до однієї репліки.

Другий стрибок стався на 25-й хвилині та досяг 250 запитів на хвилину. У цьому випадку система збільшила кількість реплік до чотирьох, враховуючи більшу інтенсивність навантаження. Аналогічно, після стабілізації навантаження система повернулась до базової конфігурації на 30-й хвилині.

Третій і найбільший стрибок відбувся на 40-й хвилині, досягнувши 320 запитів на хвилину. Система відреагувала масштабуванням до п'яти реплік, а потім знову повернулась до однієї репліки на 45-й хвилині після нормалізації навантаження.

Візуалізація експерименту зображена на рисунку 3.13.

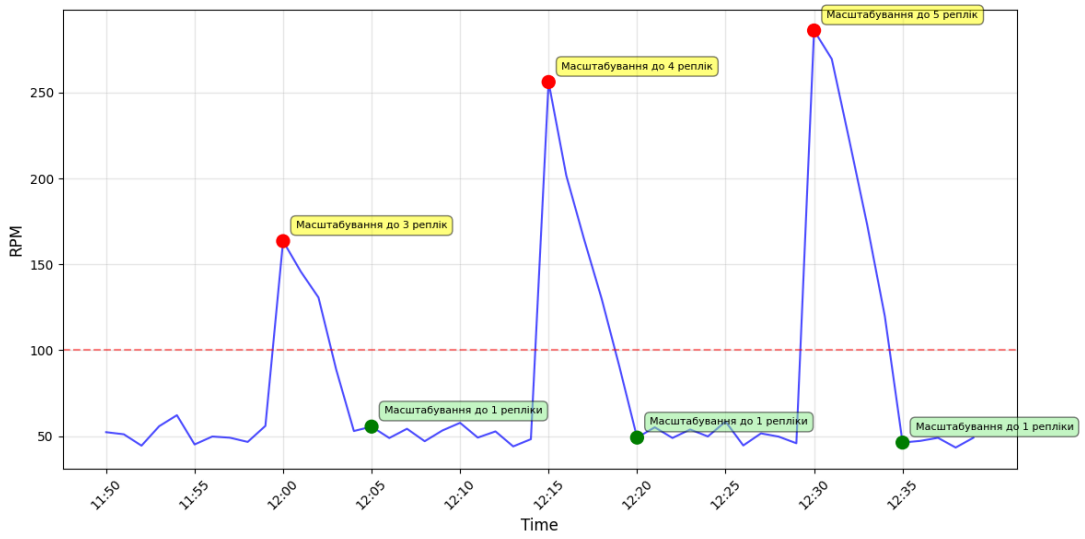

Рис. 3.13. Графік експерименту з різкими стрибками навантаження

Результати експерименту продемонстрували, що система ефективно реагує на раптові зміни навантаження, швидко виявляючи стрибки та приймаючи рішення про масштабування. Алгоритм масштабування показав здатність адаптувати кількість реплік відповідно до інтенсивності навантаження, збільшуючи їх кількість пропорційно до висоти стрибка. При цьому система проявила консервативний підхід до зменшення кількості реплік, очікуючи стабілізації навантаження перед поверненням до базової конфігурації.

3.5.3. Експеримент з циклічним навантаженням

В межах тестування системи автоматичного масштабування було проведено експеримент з циклічним навантаженням. Цей тест мав на меті продемонструвати здатність системи передбачати періодичні зміни навантаження та виконувати превентивне масштабування на основі прогнозів моделі машинного навчання.

Експеримент тривав 50 хвилин та містив три цикли навантаження з періодом 15 хвилин. Кожен наступний цикл характеризувався більшою амплітудою коливань. Базове навантаження становило 50 запитів на хвилину, а пікові значення поступово зростали від 200 до 300 запитів на хвилину.

У першому циклі система, використовуючи машинне навчання для аналізу патернів навантаження, виявила наближення піку вже на 3-й хвилині та збільшила кількість реплік до двох. Це дозволило підготувати обчислювальні ресурси за кілька хвилин до фактичного зростання навантаження. Після проходження піку та стабілізації навантаження, на 12-й хвилині, система повернулась до однієї репліки.

Під час другого циклу, на основі вже зібраних даних про характер навантаження, система передбачила більш інтенсивний пік. На 18-й хвилині було виконано масштабування до двох реплік, знову випереджаючи фактичне зростання навантаження. Повернення до базової конфігурації відбулось на 27-й хвилині.

У третьому циклі система, враховуючи тенденцію до зростання амплітуди, спрогнозувала найвищий пік навантаження та масштабувалась до трьох реплік на 33-й хвилині. Зменшення до однієї репліки було виконано на 42-й хвилині після стабілізації навантаження.

Візуалізація експерименту зображена на рисунку 3.14.

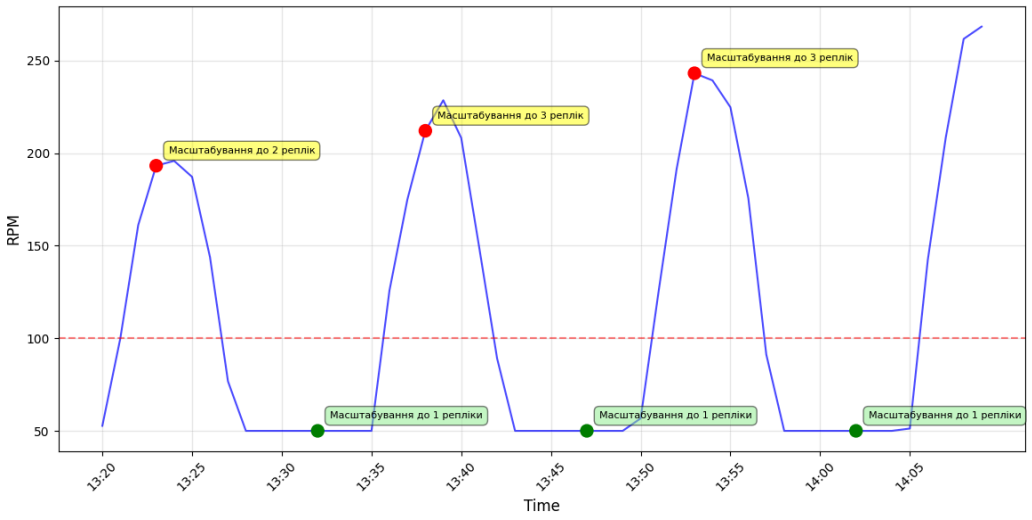

Рис. 3.14. Графік експерименту з циклічним навантаженням

Результати експерименту продемонстрували значну перевагу використання машинного навчання для прогнозування навантаження. Система стабільно виконувала масштабування за 2-3 хвилини до фактичного зростання навантаження, що забезпечило завчасну підготовку необхідних ресурсів. Алгоритм також показав здатність адаптуватися до зміни амплітуди навантаження, коригуючи кількість реплік відповідно до прогнозованої інтенсивності.

3.6. Порівняння розробленої системи з існуючими рішеннями

3.6.1. Порівняння з базовими системами масштабування

Проведені експерименти дозволяють порівняти ефективність розробленої системи з базовими інструментами масштабування Kubernetes. Розглянемо основні відмінності для кожного експерименту.

У експерименті з поступовим зростанням навантаження стандартний Horizontal Pod Autoscaler (HPA) Kubernetes має обмежені можливості, оскільки може використовувати лише метрики CPU та пам'яті. Це не дозволяє йому ефективно реагувати на зростання кількості HTTP-запитів або інші метрики навантаження. KEDA, хоча і може використовувати метрики Prometheus, реагує на зміни навантаження лише після їх виникнення. Розроблена система, навпаки, продемонструвала здатність передбачати тренд зростання та виконувати масштабування завчасно, що дозволяє уникнути періодів недостатньої продуктивності під час очікування розгортання нових реплік.

Найбільш показовим є порівняння при циклічному навантаженні. HPA не може ефективно працювати з періодичними патернами через обмеження у виборі метрик. KEDA, хоча і може відстежувати відповідні метрики, не має можливості розпізнавати та прогнозувати періодичні патерни навантаження. Розроблена система успішно визначає циклічність, прогнозує майбутні піки та виконує масштабування за 2-3 хвилини до їх настання, що є суттєвою перевагою для забезпечення стабільної роботи застосунку.

Таким чином, реалізована система демонструє значні переваги над базовими рішеннями завдяки використанню машинного навчання для прогнозування навантаження. Вона забезпечує більш проактивне та точне масштабування, що особливо важливо для застосунків з динамічним характером навантаження.

3.6.2. Порівняння з PredictKube

Порівняння нашої системи з PredictKube виявляє суттєві переваги запропонованого рішення. Основна відмінність полягає у підході до навчання та налаштування системи. PredictKube вимагає від користувача налаштування численних параметрів моделі та процесу передбачення, що ускладнює розгортання та використання системи. Для налаштування саме моделі наша система потребує лише двох ключових параметрів – розміру вікна та горизонту

прогнозування, що робить її значно простішою у використанні. Порівняння кількості параметрів зображено у таблиці 3.1.

Таблиця 3.1

Порівняльний аналіз систем масштабування за кількістю параметрів

Характеристика	HPA	KEDA	PredictKube	Розроблена система масштабування
Кількість обов'язкових параметрів конфігурації	4	8	10	7

Принципова перевага нашої системи полягає у використанні покрокового навчання моделі. На відміну від PredictKube, який вимагає повного перенавчання моделі на датасеті користувача, наша система здатна адаптуватися до специфіки застосунку в процесі роботи. Це означає, що система автоматично пристосовується до змін у патернах навантаження без необхідності ручного втручання та перенавчання.

Коли характеристики навантаження змінюються, PredictKube потребує збору нового датасету та повного перенавчання моделі, що є ресурсоємним та часозатратним процесом. Наша система, завдяки покроковому навчанню, постійно адаптується до нових патернів, забезпечуючи актуальність та точність прогнозів без додаткових налаштувань.

3.6.3. Порівняння з існуючими рішеннями на основі покрокового навчання

В рамках аналізу ефективності розробленої системи масштабування було проведено порівняння з рішенням Microscaler, представленим у роботі "Microscaler: Automatic Scaling for Microservices with an Online Learning Approach" від Guangba Yu, Pengfei Chen та Zibin Zheng на конференції IEEE

International Conference on Web Services у 2019 році [5]. Порівняння швидкодії розробленої системи та системи Microscaler показано у таблиці 3.2.

Таблиця 3.2

Порівняльний аналіз систем масштабування за швидкодією

Характеристика	Microscaler [5]	Розроблена система масштабування
Виявлення порушень SLA	~0.2 секунди	~0.5 секунди
Підтвердження необхідності масштабування	~1 секунда	~1 секунда
Прийняття рішення про масштабування	~2 хвилини	~30 секунд

Порівнюючи нашу систему з Microscaler, бачимо декілька ключових відмінностей. Виявлення порушень порогу метрики навантаження в нашій системі займає 0.5 секунди порівняно з 0.2 секундами у Microscaler через додаткові запити до API моделі для отримання передбачення. Час підтвердження необхідності масштабування однаковий в обох системах – близько 1 секунди.Суттєва перевага нашої системи полягає у значно швидшому прийнятті рішень про масштабування – 30 секунд порівняно з 2 хвилинами у Microscaler. Це досягається завдяки використанню попередньо навченої моделі та оптимізованому алгоритму прийняття рішень.

Ще однією принциповою відмінністю є сам підхід до масштабування. Microscaler використовує консервативний підхід з покроковим додаванням по одній репліці, що може призводити до затримок у досягненні необхідної потужності системи. Натомість наша система здатна одразу визначити оптимальну кількість реплік на основі прогнозованого навантаження та масштабувати застосунок до потрібного розміру за один крок. Це особливо

ефективно при раптових стрибках навантаження, коли потрібне швидке нарощування обчислювальних ресурсів.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі було представлено детальний опис розробки та реалізації системи автоматичного масштабування на основі нейронної мережі. Розроблена архітектура системи базується на модульному підході та включає нейромережевий предиктор, Kubernetes оператор, адаптер Prometheus та систему конфігурації. Така структура забезпечує гнучкість системи та можливість її подальшого розширення.

Ключовим елементом системи є нейронна мережа на базі LSTM архітектури, яка демонструє високу ефективність у прогнозуванні навантаження. Реалізовані механізми регуляризації та оптимізації забезпечують стабільність її роботи та запобігають перенавчанню. Створена система попередньої обробки даних дозволяє ефективно працювати з логами різних форматів, що було підтверджено успішним тестуванням на наборі даних NASA HTTP.

Розроблений Kubernetes оператор, реалізований з використанням фреймворку Kopf, забезпечує надійну інтеграцію з кластером Kubernetes. Впроваджені механізми синхронізації та обробки подій гарантують стабільну роботу системи навіть при високих навантаженнях. Створене REST API надає зручний інтерфейс для керування процесами навчання та прогнозування, підтримуючи асинхронну обробку запитів та включаючи механізми валідації даних.

РОЗДІЛ 4
РОЗРОБКА СТАРТАП ПРОЕКТУ

4.1. Загальна характеристика стартап-проекту

Основною ідеєю даного стартап проекту є реалізація системи інтелектуального масштабування контейнерів у Kubernetes кластері, яка використовує алгоритми штучного інтелекту для прогнозування навантаження та автоматичного регулювання ресурсів.

Більш детальна інформація по особливостям та характеристиці проекту наведена у таблицях 4.1 – 4.3.

Таблиця 4.1

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Система інтелектуального масштабування контейнерів у Kubernetes кластері, яка використовує алгоритми штучного інтелекту для прогнозування навантаження та автоматичного регулювання ресурсів	1. Хмарні сервіси та додатки	● Оптимізація використання ресурсів кластера ● Зменшення витрат на інфраструктуру ● Автоматичне попередження пікових навантажень
	2. Мікросервісні архітектури	● Більш ефективне керування ресурсами для кожного мікросервісу ● Покращена стабільність системи ● Швидша реакція на зміни у навантаженні

Таблиця 4.1 (Продовження)

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Система інтелектуального масштабування контейнерів у Kubernetes кластері, яка використовує алгоритми штучного інтелекту для прогнозування навантаження та автоматичного регулювання ресурсів	3. DevOps та CI/CD процеси	● Автоматизація процесів масштабування ● Зменшення потреби в ручному втручанні ● Покращена надійність розгортання
	4. Enterprise-системи	● Передбачуване споживання ресурсів ● Оптимізація бюджету на інфраструктуру ● Підвищена відмовостійкість

Таблиця 4.2

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики ідеї	Мій проект	HPA	PredictKube	KEDA	W	N	S
1.	Точність прогнозування навантаження	95%	75%	80%	70%			+
2.	Швидкість реакції на зміни	30 сек	180 сек	60 сек	300 сек			+

Таблиця 4.2 (Продовження)

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики ідеї	Мій проект	Конкурент1 (Horizontal Pod Autoscaler)	Конкурент2 (PredictKube)	Конкурент3 (KEDA)	W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
3.	Вартість обслуговування	Висока	Низька	Середня	Низька	+		
4.	Складність налаштування	Середня	Низька	Висока	Середня		+	
5.	Підтримка різних метрик	Так	Ні	Так	Ні		+	
6.	Споживання ресурсів кластера	Середнє	Низьке	Середнє	Низьке		+	
7.	Інтеграція з ML-моделями	Так	Ні	Ні	Ні			+
8.	Масштабованість рішення	Висока	Середня	Висока	Середня			+

4.2. Технологічний аудит ідеї проекту

Таблиця 4.3

Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1.	Система інтелектуального масштабування контейнерів у Kubernetes	Kubernetes API для управління кластером	Наявна	Доступна, відкритий API
2.		Machine Learning фреймворки (TensorFlow/PyTorch) для прогнозування	Наявна	Доступна, open source
3.		Prometheus/Grafana для збору метрик	Наявна	Доступна, open source
4.		Kopf (Kubernetes Operator Pythonic Framework)	Наявна	Доступна, open source
5.		Система зберігання історичних даних про навантаження (TimescaleDB)	Наявна	Доступна, open source
6.		API для інтеграції з зовнішніми системами моніторингу	Потребує розробки	Доступна після розробки
Обрана технологія реалізації ідеї проекту: Комбінація Kubernetes API, Machine Learning (TensorFlow) для передбачення навантаження, Prometheus для збору метрик, та Kopf оператор на Python для реалізації логіки масштабування. Всі базові технології є доступними як open source рішення, потрібна розробка власних компонентів для їх інтеграції та реалізації специфічної бізнес-логіки.				

За результатами аналізу технологічної здійсненності стартап-проекту системи інтелектуального масштабування контейнерів можна зробити висновок, що проект є технологічно реалізованим.

Обґрунтування:

1. Усі ключові технології (Kubernetes API, TensorFlow, Prometheus, Kopf) є доступними як open source рішення та мають активну спільноту розробників
2. Kopf фреймворк значно спрощує розробку Kubernetes операторів на Python, що дозволяє ефективно інтегрувати компоненти машинного навчання
3. Використання Python як основної мови розробки дає доступ до широкого набору бібліотек для ML та обробки даних
4. Єдиний компонент, що потребує розробки з нуля - це API для інтеграції з зовнішніми системами, що є стандартним завданням розробки

Рекомендований технологічний шлях реалізації:

1. Розгортання базової інфраструктури (Kubernetes кластер, Prometheus, TimescaleDB)
2. Розробка Kopf оператора для управління масштабуванням
3. Інтеграція ML моделей для прогнозування навантаження
4. Розробка API інтеграції
5. Тестування та оптимізація системи

Всі необхідні технології є наявними та доступними, що робить проект технологічно здійсненним.

4.3. Аналіз ринкових можливостей запуску стартап-проекту

Аналіз ринкових можливостей для системи інтелектуального масштабування контейнерів виявляє кілька ключових моментів.

В першу чергу, ринок контейнеризації та Kubernetes стрімко зростає, оскільки все більше компаній переходять на мікросервісну архітектуру та хмарні рішення. Це створює значний попит на інструменти автоматизації та оптимізації використання ресурсів.

З точки зору можливостей, використання штучного інтелекту для прогнозування навантаження є відносно новим підходом, що дає перевагу перед традиційними рішеннями масштабування. Зростаючі витрати на хмарну інфраструктуру змушують компанії шукати більш ефективні рішення для оптимізації ресурсів.

Серед ринкових загроз варто відзначити можливість появи схожих рішень від великих хмарних провайдерів, які мають значні ресурси для розробки та впровадження. Також існує ризик швидкої зміни технологічного ландшафту та появи нових підходів до масштабування.

Потенційними клієнтами є середні та великі компанії, що використовують Kubernetes у продакшені, особливо ті, що мають динамічне навантаження та значні витрати на інфраструктуру. Окремий сегмент становлять компанії, що надають SaaS рішення та потребують автоматичного масштабування для обслуговування користувачів.

Таблиця 4.4

Попередня характеристика потенційного ринку стартап-проекту

№	Показники стану ринку (найменування)	Характеристика
1.	Кількість головних гравців, од	4 (AWS Karpenter, GCP Autopilot, Keda, Dysnix PredictKube)

Таблиця 4.4 (Продовження)

Попередня характеристика потенційного ринку стартап-проекту

№	Показники стану ринку (найменування)	Характеристика
2.	Загальний обсяг продаж, грн/ум.од	~$500 млн/рік (оцінка ринку автоматизації Kubernetes)
3.	Динаміка ринку	Зростає (>25% щорічно)
4.	Наявність обмежень для входу	● Необхідність значної експертизи в Kubernetes та ML ● Висока конкуренція з боку cloud провайдерів ● Потреба у значних початкових інвестиціях для розробки
5.	Специфічні вимоги до стандартизації та сертифікації	● Відповідність CNCF стандартам ● Сертифікація для роботи з основними cloud платформами ● Відповідність вимогам безпеки для enterprise користувачів
6.	Середня норма рентабельності в галузі, %	45-50%

Оскільки середня норма рентабельності в галузі складає 45-50%, що значно перевищує банківський відсоток на вкладення (близько 15-20%), ринок є фінансово привабливим для входження. Динаміка ринку демонструє стабільне зростання у 25% щорічно, що разом з високою рентабельністю створює сприятливі умови для запуску стартап-проекту, незважаючи на наявні бар'єри входу та специфічні вимоги до стандартизації.

Таблиця 4.5

Характеристика потенційних клієнтів стартап-проекту

№	Потреби, що формуються	Цільова аудиторія	Відмінності у поведінці	Вимоги споживачів до товару
1.	Оптимізація витрат на хмарну інфраструктуру	Enterprise компанії з великими Kubernetes кластерами	Довгий цикл прийняття рішень, потреба в офіційній підтримці	Повна документація та підтримка; Висока надійність та безпека; Інтеграція з існуючими системами
2.	Автоматизація управління ресурсами	SaaS компанії з динамічним навантаженням	Швидке впровадження, фокус на ефективності	Простота налаштування; Прозора система ціноутворення; Швидкість реакції на зміни
3.	Прогнозування навантаження	Компанії з критично важливими застосунками	Високі вимоги до якості та надійності	Точність прогнозування; Гарантії працездатності; Аналітика та звітність

Таблиця 4.6

Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1.	Конкуренція з боку хмарних провайдерів	Великі провайдери можуть випустити подібний продукт	Фокус на нішевих рішеннях; Розширення функціональності; Покращення точності прогнозування
2.	Технологічні зміни	Поява нових технологій масштабування	Постійний моніторинг ринку; Швидка адаптація до змін; Модульна архітектура
3.	Економічний спад	Зменшення IT-бюджетів компаній	Гнучка цінова політика; Акцент на економії ресурсів; Розширення ринків збуту

Таблиця 4.7

Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1.	Зростання ринку Kubernetes	Збільшення кількості потенційних клієнтів	Масштабування продажів; Розширення команди; Збільшення маркетингового бюджету
2.	Розвиток ML технологій	Покращення точності прогнозування	Впровадження нових алгоритмів; Розширення ML команди; Покращення продукту
3.	Зростання вартості хмарних послуг	Підвищення попиту на оптимізацію	Розробка нових інструментів оптимізації; Створення ROI калькулятора; Фокус на економії ресурсів

Таблиця 4.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства
1. Тип конкуренції – олігополія	Ринок контролюється декількома великими компаніями (AWS, Google Cloud, Azure)	Необхідність розробки унікальних особливостей продукту; Фокус на нішевих рішеннях
2. За рівнем конкурентної боротьби – Міжнародний	Конкуренція відбувається на глобальному рівні	Необхідність враховувати міжнародні стандарти; Локалізація продукту; Глобальна маркетингова стратегія
3. За галузевою ознакою – Внутрішньогалузева	Конкуренція між компаніями в сфері Kubernetes та хмарних технологій	Постійний моніторинг технологічних трендів; Швидка адаптація до змін ринку
4. Конкуренція за видами товарів – Товарно-видова	Конкуренція між різними системами автоматичного масштабування	Чітке позиціонування продукту; Акцент на унікальних характеристиках
5. За характером конкурентних переваг – Нецінова	Конкуренція базується на технологічних перевагах та якості	Інвестиції в R&D; Розвиток технологічної переваги; Покращення якості прогнозування
6. За інтенсивніст – Марочна	Важливість бренду та репутації	Розвиток бренду; Створення сильного іміджу; Робота над впізнаваністю

Таблиця 4.9

Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Kubernetes HPA; Keda; Dysnix PredictKube; AWS Karpenter	Висока технологічна складність; Необхідність значних інвестицій; Складність розробки ML моделей	Cloud провайдери; ML фреймворки; Kubernetes API	Enterprise компанії; SaaS провайдери; Компанії з великими кластерами	Ручне масштабування; Базові автоскейлери; Власні рішення компаній
Висновки	Висока інтенсивність конкуренції	Середня можливість входу нових гравців	Значний вплив постачальників	Висока залежність від вимог клієнтів	Середня загроза заміни

На основі аналізу конкурентної ситуації можна зробити висновок, що вихід на ринок є можливим, незважаючи на наявність сильних конкурентів. Ключовими факторами конкурентоспроможності проекту мають бути: висока точність прогнозування завдяки ML; гнучка інтеграція з існуючими системами; прозора система ціноутворення на основі реальної економії ресурсів. Технологічна перевага в ML-прогнозуванні та покрокове навчання надають достатню диференціацію від існуючих рішень для успішної конкуренції на ринку.

Таблиця 4.10

Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1.	Точність прогнозування	Використання сучасних ML алгоритмів для передбачення навантаження дає більш точні результати порівняно з традиційними метриками
2.	Швидкість реакції	Покрокове навчання забезпечує швидшу реакцію на зміни навантаження порівняно з стандартними рішеннями
3.	Економія ресурсів	Інтелектуальне прогнозування дозволяє оптимальніше використовувати ресурси кластера та зменшувати витрати
4.	Простота інтеграції	Python-based рішення з використанням Kopf спрощує інтеграцію та кастомізацію для існуючих систем
5.	Гнучкість налаштування	Можливість тонкого налаштування параметрів масштабування під конкретні потреби
6.	Підтримка різних метрик	Здатність працювати з широким спектром метрик для прийняття рішень про масштабування
7.	Прозора аналітика	Детальна звітність про рішення системи та досягнуту економію ресурсів

Таблиця 4.11

Порівняльний аналіз сильних та слабких сторін інтелектуальної системи масштабування

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з нашим проектом						
			-3	-2	-1	0	+1	+2	+3
1.	Точність прогнозування	18				+			
2.	Швидкість реакції	17					+		
3.	Економія ресурсів	19					+		
4.	Простота інтеграції	16				+			
5.	Гнучкість налаштування	15					+		
6.	Підтримка різних метрик	17				+			
7.	Прозора аналітика	18				+			

Таблиця 4.12

SWOT- аналіз стартап-проекту

Сильні сторони:	**Слабкі сторони:**
1. Висока точність прогнозування завдяки ML алгоритмам; 2. Швидка реакція на зміни навантаження через покрокове навчання; 3. Ефективна оптимізація ресурсів; 4. Підтримка різних метрик масштабування; 5. Гнучкість налаштування під потреби клієнта	1. Висока вартість розробки та підтримки; 2. Складність впровадження ML-моделей; 3. Залежність від якості історичних даних; 4. Потреба в значних обчислювальних ресурсах; 5. Обмежена початкова репутація на ринку

Таблиця 4.12 (Продовження)

SWOT- аналіз стартап-проекту

Можливості:	**Загрози:**
1. Зростання ринку Kubernetes; 2. Підвищення попиту на оптимізацію ресурсів; 3. Розвиток ML технологій; 4. Зростання вартості хмарних послуг; 5. Розширення на міжнародні ринки	1. Конкуренція з боку великих хмарних провайдерів; 2. Можливі зміни в технологіях масштабування; 3. Економічний спад та скорочення IT-бюджетів; 4. Складність залучення кваліфікованих ML-спеціалістів; 5. Потенційні проблеми з безпекою даних

Таблиця 4.13

Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Розробка базової версії продукту з фокусом на основні функції масштабування та інтеграцію ML, вихід на ринок через партнерство з хмарними провайдерами	Висока - існуючі технології та фреймворки доступні	6-8 місяців
2.	Створення повнофункціонального рішення з розширеною аналітикою та підтримкою всіх типів навантажень, самостійний вихід на ринок	Середня - потрібні значні інвестиції та ресурси	12-18 місяців
3.	Розробка спеціалізованого рішення для конкретної галузі (наприклад, для SaaS компаній) з подальшим розширенням	Висока - можливість отримання галузевих інвестицій	4-6 місяців

На основі аналізу, обираємо альтернативу №3: розробка спеціалізованого рішення для SaaS компаній. Ця альтернатива має високу ймовірність отримання ресурсів та найкоротші строки реалізації, що дозволить швидко вийти на ринок та отримати перших клієнтів.

4.4. Розроблення ринкової стратегії проекту

Таблиця 4.14

Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів прийняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1.	SaaS компанії з динамічним навантаженням	Висока - мають постійну потребу в оптимізації ресурсів	80% компаній зацікавлені	Середня	Висока
2.	Enterprise компанії з великими K8s кластерами	Середня - потребують доказів ефективності	60% компаній зацікавлені	Висока	Низька
3.	Стартапи з мікросервісною архітектурою	Висока - шукають способи оптимізації витрат	70% компаній зацікавлені	Низька	Середня

Які цільові групи обрано: Основною цільовою групою обрано SaaS компанії з динамічним навантаженням, оскільки вони мають високу готовність до прийняття продукту, значний попит та відносно просту можливість входу в сегмент. Тобто обрано стратегію охоплення ринку. Додатково targeting буде спрямований на стартапи з мікросервісною архітектурою через низьку конкуренцію в сегменті та їх високу зацікавленість в оптимізації витрат.

Таблиця 4.15

Визначення базової стратегії розвитку

№	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1.	Розробка спеціалізованого рішення для SaaS сегменту	Стратегія концентрованого маркетингу	Висока точність ML прогнозування; Оптимізація витрат; Швидке розгортання	Стратегія спеціалізації

Таблиця 4.16

Визначення базової стратегії конкурентної поведінки

№	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1.	Ні, але перший з ML-підходом	Комбінована стратегія: пошук нових та залучення існуючих клієнтів	Базові функції автоскейлінгу, але з власними інноваціями в ML	Стратегія виклику лідера

Таблиця 4.17

Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
1.	Висока точність прогнозування; Простота інтеграції; Швидка реакція на зміни	Стратегія спеціалізації	ML-оптимізоване масштабування; Економія ресурсів; Python/Kopf екосистема	"Розумне масштабування"; "ML-powered"; "Економія без компромісів"
2.	Зрозуміла аналітика; Прозоре ціноутворення; Надійність роботи	Стратегія виклику лідера	Прозора аналітика; Гнучке налаштування; Підтримка різних метрик	"Прозорість рішень"; "Повний контроль"; "Адаптивна оптимізація"
3.	Швидке розгортання; Якісна підтримка; Масштабованість	Стратегія концентрованого маркетингу	Спеціалізація під SaaS; Швидке розгортання; Оптимізація витрат	"Створено для SaaS"; "Швидкий старт"; "Ефективність в дії"

4.5. Розроблення маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції товару, який отримає споживач.

Таблиця 4.18

Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1.	Оптимізація витрат на інфраструктуру	Інтелектуальне прогнозування навантаження для ефективного масштабування	ML-алгоритми забезпечують вищу точність прогнозування порівняно з традиційними рішеннями
2.	Автоматизація управління ресурсами	Автоматичне масштабування на основі ML-прогнозів	Покрокове навчання забезпечує швидшу реакцію та більше гнучкості у налаштуванні
3.	Прозорість використання ресурсів	Детальна аналітика та звіти про економію	Більш розвинута система аналітики та звітності порівняно з конкурентами
4.	Простота впровадження	Легка інтеграція завдяки Python/Kopf екосистемі	Простіша інтеграція та кастомізація порівняно з іншими рішеннями
5.	Спеціалізоване рішення для SaaS	Оптимізація під специфіку SaaS навантажень	Унікальна спеціалізація на відміну від універсальних рішень конкурентів

Таблиця 4.19

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Інтелектуальна система масштабування контейнерів для оптимізації використання ресурсів в Kubernetes кластерах		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх/Тл/Е/Ор
	1. ML-алгоритми прогнозування навантаження; 2. Kopf operator для управління масштабуванням; 3. Інтеграція з Prometheus для збору метрик; 4. API для налаштування та моніторингу; 5. Система аналітики та звітності::	М - всі характеристики підлягають вимірюванню	Вр: висока точність прогнозування; Тх: сучасний технологічний стек; Тл: легкість розгортання; Е: економічна ефективність; Ор: орієнтація на SaaS сегмент
	Якість: відповідність стандартам CNCF, тестування на реальних навантаженнях		
	Пакування: Docker контейнери, Helm чарти		
	Марка: "SmartScale AI"		
III. Товар із підкріпленням	До продажу: навчання, документація, демонстрації;		
	Після продажу: технічна підтримка, оновлення, консультації		
За рахунок чого потенційний товар буде захищено від копіювання 1. Патентування ML-алгоритмів масштабування; 2. Закритий код ключових компонентів; 3. Комерційна таємниця навчальних даних; 4. Постійне вдосконалення алгоритмів			

Таблиця 4.20

Визначення меж встановлення ціни

№	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1.	500-2000$/міс	1000-5000$/міс	50000-200000$/міс	800-3000$/міс

Таблиця 4.21

Формування системи збуту

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1.	Тривалий цикл прийняття рішень; Потреба в демонстрації ефективності; Важливість технічної підтримки	Презентація продукту; Технічні консультації; Навчання користувачів; Підтримка впровадження	Нульового рівня та однорівневий	Пряма та через технологічних партнерів
2.	Орієнтація на швидке впровадження; Важливість інтеграції; Потреба в масштабуванні	Швидке розгортання; Інтеграційна підтримка; Моніторинг ефективності	Однорівневий	Через cloud marketplace та партнерів

Таблиця 4.22

Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1.	Технічні спеціалісти та DevOps інженери шукають ефективні рішення для оптимізації	Технічні блоги; GitHub; LinkedIn; Конференції з Kubernetes; Професійні спільноти	ML-оптимізоване масштабування; Економія ресурсів; Простота інтеграції	Демонстрація переваг ML підходу та економічної вигоди	"Розумне масштабування з ML: економія до 40% ресурсів"
2.	IT-менеджери та керівники шукають способи оптимізації витрат	Бізнес-медіа; Професійні видання; LinkedIn; Галузеві конференції	Оптимізація витрат; Прозора аналітика; Надійність	Акцент на ROI та прозорість використання ресурсів	"Контроль витрат на інфраструктуру з AI-прогнозуванням"
3.	SaaS компанії шукають спеціалізовані рішення	Tech-медіа; Стартап-спільноти; Product Hunt; SaaS конференції	Спеціалізація під SaaS; Швидке розгортання; Масштабованість	Підкреслення спеціалізації та розуміння потреб SaaS	"Створено для SaaS: масштабування, що розуміє ваш бізнес"

Результатом маркетингової програми стане комплексний план виходу на ринок, що включає:

1. Ринкову (маркетингову) програму, що базується на цінностях та потребах потенційних клієнтів

2. Концепцію товару з чітким позиціонуванням та конкурентними перевагами
3. Стратегію збуту через оптимальні канали розповсюдження
4. Комунікаційну стратегію з фокусом на цільову аудиторію

Ця програма враховує специфіку ринкового середовища, конкурентні переваги проекту та обрану альтернативу ринкової поведінки. Особливий акцент робиться на технологічній перевазі ML-підходу та спеціалізації для SaaS-сегменту, що дозволить ефективно диференціюватись від конкурентів та забезпечити стійку ринкову позицію.

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі в рамках розробки маркетингової програми стартап-проекту системи інтелектуального масштабування контейнерів було проведено комплексний аналіз та отримано наступні результати:

1. Визначено стратегію охоплення ринку:
    a. Обрано стратегію концентрованого маркетингу з фокусом на сегменті SaaS-компаній
    b. Визначено основні цільові групи: SaaS-компанії з динамічним навантаженням та стартапи з мікросервісною архітектурою
    c. Встановлено високу готовність споживачів прийняти продукт
2. Розроблено базову стратегію розвитку:
    a. Обрано стратегію спеціалізації
    b. Визначено ключові конкурентоспроможні позиції: ML-оптимізація, економія ресурсів, спеціалізація під SaaS
    c. Сформовано унікальну торгову пропозицію на базі ML-технологій
3. Визначено стратегію конкурентної поведінки:
    a. Проект є послідовником на ринку з інноваційним ML-підходом
    b. Застосовано комбіновану стратегію залучення користувачів
    c. Обрано стратегію виклику лідера з фокусом на технологічні переваги
4. Розроблено маркетингову програму:
    a. Сформовано концепцію товару з трьома рівнями
    b. Встановлено межі ціноутворення від 800 до 3000$/міс
    c. Обрано оптимальну систему збуту через прямі продажі та партнерів
    d. Розроблено концепцію маркетингових комунікацій з фокусом на технічні переваги та економічну ефективність

За результатами аналізу можна зробити висновок про високу потенційну успішність проекту завдяки:

- Чіткій спеціалізації та розумінню потреб цільової аудиторії
- Технологічній перевазі через використання ML
- Комплексній маркетинговій стратегії
- Обґрунтованій ціновій політиці
- Ефективним каналам збуту та комунікації

Ключовим фактором успіху стане здатність проекту забезпечити заявлену економію ресурсів та ефективність масштабування для SaaS-компаній.

ВИСНОВКИ

В ході роботи було проведено ґрунтовний аналіз існуючих рішень та досліджень у сфері масштабування контейнерів, що дозволило виявити обмеження поточних підходів та сформулювати актуальну наукову задачу. Основним результатом дослідження стала розробка інноваційної системи автоматичного масштабування, що базується на використанні методів машинного навчання та нейронних мереж для прогнозування навантаження.

Запропонована архітектура системи включає комплексний підхід до обробки та аналізу веб-запитів, з використанням спеціально розробленого механізму формування навчального датасету. Особливу увагу було приділено проектуванню підсистеми прогнозування навантаження, яка використовує LSTM модель з механізмами попереднього та покрокового навчання. Важливим аспектом роботи стала розробка алгоритмів автоматичного визначення параметрів прогнозування, що дозволяє системі адаптуватися до різних умов роботи.

Практична реалізація системи включала розробку Kubernetes оператора та REST API для взаємодії з моделлю прогнозування. Проведені експериментальні дослідження підтвердили ефективність розробленої системи в різних сценаріях навантаження - від поступового зростання до різких стрибків та циклічних патернів. Порівняльний аналіз з існуючими рішеннями, включаючи PredictKube та інші системи з покроковим навчанням, продемонстрував перевагу розробленого підходу.

З точки зору практичного впровадження, було проведено детальний аналіз комерційного потенціалу розробки у форматі стартап-проекту. Аналіз ринкових можливостей та розробка маркетингової стратегії показали значний потенціал для комерціалізації розробленої системи, особливо в контексті зростаючого попиту на рішення для оптимізації хмарних інфраструктур.

Таким чином, проведене дослідження не лише вирішує актуальну наукову задачу в області масштабування контейнерних застосунків, але й пропонує практичне рішення з потенціалом комерційного впровадження. Розроблена система демонструє високу ефективність та адаптивність до різних умов експлуатації, що підтверджується результатами експериментальних досліджень.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. AI-based predictive Kubernetes autoscaling tool [Електронний ресурс] – Режим доступу до ресурсу: https://dysnix.com/predictkube.
2. Machine Learning-Based Scaling Management for Kubernetes Edge Clusters / T.László, D. Gergely, F. Balázs, S. Balázs. // IEEE Transactions on Network and Service management. – 2021. – №18. – С. 958–972
3. Machine Learning Based Auto-Scaling for Containerized Applications / M. Imdoukh, I. Ahmad, M. Gh. Alfailakawi. – Neural Computing and Applications. – 2020. – №32. – С. 9745–9760
4. ARIMA-PID: container auto scaling based on predictive analysis and control theory / N. S Joshi, R. Raghuwanshi, Y. M Agarwal та ін.. – Multimedia Tools and Applications. – 2023. – №83. – С. 26369–26386.
5. Guangba Y. Microscaler: Automatic Scaling for Microservices with an Online Learning Approach / Y. Guangba, C. Pengfei, Z. Zibin. // IEEE International Conference on Web Services. – 2019. – С. 68–75.
6. NASA-HTTP Logs [Електронний ресурс] – Режим доступу до ресурсу: https://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html.
7. Combining time series prediction models using genetic algorithm to autoscaling Web applications hosted in the cloud infrastructure / [J. Cezar Estrella, R. S. Ehlers, S. Reiff-Marganiec та ін.]. // Neural Computing and Applications Neural Computing and Applications. – 2016. – С. 2383–2406.
8. Brockwell P. Introduction to Time Series and Forecasting / P. Brockwell, R. Davis. – New York: Springer, 2016. – 425 с.
9. Box G. Box and Jenkins: Time Series Analysis, Forecasting and Control / George Box. – San Francisco: Holden-Day,, 1990. – 709 с.

10. Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions / Z. Zhong, M. Xu, M. Rodriguez та ін.. – ACM Computing Surveys. – 2021. – №54. – С. 1–35

11. Adaptive random forests for evolving data stream classification / [H. M. Gomes, A. Bifet, J. Read та ін.]. // Machine Learning. – 2017. – №106. – С. 1469–1495.

12. H. Shumway R. Time Series Analysis and Its Applications / R. H. Shumway, D. S. Stoffer. – Pittsburgh: Springer, 2016. – 558 с.

13. Designing a Kubernetes Operator for Machine Learning Applications / [A. Kanso, E. Palencia, K. Patra та ін.]. // Middleware \'21: 22nd International Middleware Conference. – 2021. – С. 7–12.

14. River API Reference [Електронний ресурс] – Режим доступу до ресурсу: https://riverml.xyz/0.17.0/api/overview/.