ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Метод імплементації алгоритмів штучного інтелекту для підвищення функціональності та продуктивності хмарних технологій»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Іван ПОЛТАВСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Іван ПОЛТАВСЬКИЙ

Керівник: Владислав ЯСКЕВИЧ
канд. техн. наук., доц.

Рецензент:
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Полтавському Івану Андрійовичу

1. Тема кваліфікаційної роботи: «Метод імплементації алгоритмів штучного інтелекту для підвищення функціональності та продуктивності хмарних технологій»
   керівник кваліфікаційної роботи Владислав ЯСКЕВИЧ, канд. техн. наук, доц., затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.
2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з хмарних обчислень та прогнозування часових рядів, реальні фрагменти телеметричних даних хмарних сервісів (CPU, пам’ять, диск), згенеровані синтетичні набори навантажень різних типів, параметри статистичних, ML- та DL-моделей прогнозування, вимоги до точності визначення навантаження та стабільності масштабування.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Аналіз предметної галузі
   2. Проєктування методу імплементації алгоритмів штучного інтелекту для прогнозування навантаження в хмарних сервісах
   3. Розробка адаптивного методу прогнозування навантаження у хмарних сервісах та порівняльний аналіз його результативності

5. Перелік ілюстративного матеріалу: *презентація*
   1. Порівняльна характеристика методів.
   2. Визначення вимог для методу.
   3. Математична модель методу.
   4. Архітектура використаних моделей прогнозування.
   5. Схематичне представлення роботи методу.
   6. Аналіз ефективності розробленого методу.
   7. Практичний результат.

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10-07.11.2025	
2	Вивчення сучасних методів прогнозування часових рядів та AI алгоритмів класифікації	08.11-12.11.2025	
3	Дослідження статистичних, ML- та DL моделей для прогнозування навантаження	13.11-18.11.2025	
4	Аналіз впливу типів навантаження на точність різних моделей прогнозування	19.11-25.11.2025	
5	Розробка адаптивного методу з AI класифікацією та вибором оптимальної моделі	26.11-04.12.2025	

6	Тестування методу на різних типах навантаження та оцінка його ефективності	05.12-07.12.2025	
7	Оформлення роботи: вступ, висновки, реферат	08.12-16.12.2025	
8	Розробка демонстраційних матеріалів	17.12-19.12.2025	

Здобувач вищої освіти _____ Іван ПОЛТАВСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 102 стор., 1 табл., 26 рис., 37 джерел.

Мета роботи – підвищення функціональності та продуктивності хмарних технологій за рахунок інтеграції алгоритмів штучного інтелекту для прогнозування навантаження.

Об'єкт дослідження – процес прогнозування навантаження у хмарних технологіях.

Предмет дослідження – алгоритми штучного інтелекту на базі нейронних мереж, для прогнозування навантаження у хмарних технологіях в умовах динамічних та неоднорідних режимів роботи.

У роботі використано різноманітні методи, такі як статистичне моделювання, алгоритми машинного навчання, глибинні нейронні мережі та методи інтелектуальної класифікації часових рядів.

Проведено аналіз сучасних підходів до прогнозування навантаження у хмарних сервісах, а також досліджено їхні переваги та обмеження в умовах динамічних і нерегулярних режимів роботи.

Розроблено та оптимізовано адаптивний метод прогнозування, який поєднує результати ШІ-класифікації з оцінкою точності моделей для автоматичного вибору оптимального алгоритму.

Проведено експерименти для оцінювання ефективності запропонованого методу на різних типах навантажень, що дозволило підтвердити його переваги над традиційними підходами.

КЛЮЧОВІ СЛОВА: ХМАРНІ ТЕХНОЛОГІЇ, ПРОГНОЗУВАННЯ НАВАНТАЖЕННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, СТАТИСТИЧНІ МОДЕЛІ, ГЛИБИННІ НЕЙРОННІ МЕРЕЖІ, КЛАСИФІКАЦІЯ ЧАСОВИХ РЯДІВ, АДАПТИВНИЙ МЕТОД, АВТОМАТИЧНЕ МАСШТАБУВАННЯ, ТЕЛЕМЕТРІЯ.

ABSTRACT

Text part of the master's qualification work: 102 pages, 1 table, 26 figures, 37 sources.

The purpose of the work is to enhance the functionality and performance of cloud technologies by integrating artificial intelligence algorithms for load forecasting.

The object of the research is the process of load forecasting in cloud technologies.

The subject of the research is artificial intelligence algorithms based on neural networks for load forecasting in cloud technologies under dynamic and heterogeneous operating conditions.

The work employs various methods, such as statistical modeling, machine learning algorithms, deep neural networks, and intelligent time-series classification methods.

An analysis of modern approaches to load forecasting in cloud services has been conducted, along with an examination of their advantages and limitations under dynamic and irregular operating conditions.

An adaptive forecasting method has been developed and optimized, combining AI classification results with model accuracy assessment for automatic selection of the optimal algorithm.

Experiments were performed to evaluate the effectiveness of the proposed method across different types of workloads, confirming its advantages over traditional approaches.

KEYWORDS: CLOUD TECHNOLOGIES, LOAD FORECASTING, ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, STATISTICAL MODELS, DEEP NEURAL NETWORKS, TIME-SERIES CLASSIFICATION, ADAPTIVE METHOD, AUTOMATIC SCALING, TELEMETRY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI – Штучний інтелект (Artificial Intelligence).

API – Програмний інтерфейс застосунків (Application Programming Interface).

ARIMA – Авторегресивна інтегрована модель ковзного середнього (AutoRegressive Integrated Moving Average).

AWS – Хмарна платформа Amazon Web Services.

CPU – Центральний процесор (Central Processing Unit).

DL – Глибинне навчання (Deep Learning).

GCP – Хмарна платформа Google Cloud Platform.

HPA – Автоматичне горизонтальне масштабування у Kubernetes (Horizontal Pod Autoscaler).

K8s – Kubernetes, система оркестрації контейнерів.

LSTM – Довготривала короткочасна пам’ять (Long Short-Term Memory).

MAE – Середня абсолютна помилка (Mean Absolute Error).

MAPE – Середня абсолютна відносна помилка (Mean Absolute Percentage Error).

ML – Машинне навчання (Machine Learning).

MSE – Середньоквадратична помилка (Mean Squared Error).

NN – Нейронна мережа (Neural Network).

PCA – Аналіз головних компонентів (Principal Component Analysis).

RAM – Оперативна пам’ять (Random Access Memory).

RF – Багатолісовий регресор (Random Forest).

RMSE – Корінь середньоквадратичної помилки (Root Mean Squared Error).

RNN – Рекурентна нейронна мережа (Recurrent Neural Network).

SARIMA – Сезонна ARIMA-модель (Seasonal AutoRegressive Integrated Moving Average).

SVM – Машина опорних векторів (Support Vector Machine).

CPU Load – Навантаження на процесор.

Time Series – Часовий ряд даних.

MAE Weight – Вага моделі, отримана на основі помилки MAE.

ВСТУП

Актуальність теми дослідження зумовлена швидким зростанням використання хмарних технологій та необхідністю забезпечення стабільної роботи сервісів за умов непередбачуваних, динамічних та змішаних навантажень. Низька точність прогнозування навантаження призводить до неефективного масштабування, зниження продуктивності та збільшення витрат. Тому виникає потреба у методах, здатних адаптивно аналізувати тип навантаження та обирати найбільш ефективну модель прогнозування із залученням алгоритмів штучного інтелекту.

Метою магістерської кваліфікаційної роботи є підвищення функціональності та продуктивності хмарних технологій за рахунок інтеграції алгоритмів штучного інтелекту для прогнозування навантаження.

Для досягнення поставленої мети були визначені наступні завдання дослідження:

1. Проаналізувати існуючі методи та моделі прогнозування часових рядів у хмарних середовищах.
2. Дослідити властивості різних типів навантаження (сезонного, стохастичного, пікового та змішаного).
3. Розробити AI-класифікатор для автоматичного визначення типу навантаження на основі ознак часового ряду.
4. Реалізувати моделі прогнозування SARIMA, Random Forest та LSTM для різних структур даних.
5. Розробити метод адаптивного вибору оптимальної моделі прогнозування з урахуванням результатів AI-класифікатора та оцінки похибок.
6. Провести експериментальну перевірку методу та порівняти його ефективність з існуючими підходами.

Об'єктом дослідження є процес прогнозування навантаження у хмарних технологіях.

Предметом дослідження є алгоритми штучного інтелекту на базі нейронних мереж та статистичних моделей, застосовані для прогнозування навантаження в умовах динамічних та неоднорідних режимів роботи.

У дослідженні використано наступні методи: аналіз літературних джерел та існуючих рішень, методи математичного аналізу та аналізу часових рядів, моделювання архітектури системи, налаштування статистичних моделей, навчання алгоритмів машинного навчання та глибинних нейронних мережі, реалізація методу мовою Python в середі JupyterLab.

Наукова новизна одержаних результатів полягає у:

1. Вперше розроблено адаптивний метод прогнозування навантаження у хмарних сервісах, який, на відміну від існуючих підходів, поєднує AI класифікацію типу часового ряду з динамічним вибором оптимальної моделі прогнозування.
2. Удосконалено процес визначення типу навантаження за рахунок використання алгоритмів машинного навчання (Random Forest) для аналізу статистичних та спектральних ознак, що забезпечує точнішу класифікацію сезонних, стохастичних, пікових та змішаних режимів.
3. Дістало подальшого розвитку застосування комбінованих моделей прогнозування шляхом обґрунтування ефективності використання різних моделей (SARIMA, Random Forest, LSTM) для різних типів навантаження та інтеграції результатів їх роботи в єдину адаптивну систему.

Практичне значення одержаних результатів полягає у можливості підвищення точності прогнозування навантаження у хмарних сервісах, зменшенні кількості помилкових масштабувань та оптимізації використання ресурсів, можливості інтеграції розробленого методу у системи оркестрації контейнерів, такі як Kubernetes, забезпеченні більш стабільної роботи сервісів у динамічних режимах навантаження.

Апробація результатів магістерської роботи. Основні положення та результати дослідження доповідалися та обговорювалися на науково-практичних конференціях та семінарах з питань інформаційних технологій в освіті.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1. Загальний огляд хмарних сервісів

Хмарні сервіси - це технології, які забезпечують зберігання, обробку та доступ до даних через інтернет без необхідності мати фізичні сервери. Вони поділяються на три основні моделі: інфраструктура як послуга (IaaS), платформа як послуга (PaaS) та програмне забезпечення як послуга (SaaS). Кожна з цих моделей пропонує унікальні можливості для користувачів залежно від їхніх потреб та вирішує різні проблеми на різних рівнях[1].

Модель IaaS (Infrastructure as a Service) забезпечує доступ до базових обчислювальних ресурсів, таких як віртуальні машини, сховища даних і мережі. Це дозволяє організаціям масштабувати інфраструктуру залежно від завантаження, мінімізуючи витрати на фізичне обладнання. Наприклад, компанії можуть орендувати віртуальні сервери для запуску своїх додатків без необхідності будувати власні дата-центри. [1].

Модель PaaS (Platform as a Service) надає платформу для розробки, тестування та розгортання програмного забезпечення. Розробники отримують доступ до інструментів, фреймворків та середовищ виконання без необхідності турбуватися про управління інфраструктурою. Наприклад, сервіси на базі PaaS, такі як Google App Engine, дозволяють швидко створювати веб-додатки, зосереджуючись лише на коді[1].

Модель SaaS (Software as a Service) пропонує готові програмні рішення, доступні через інтернет. Сервіси, як-от Google Workspace або Salesforce, надають можливість використовувати потужне програмне забезпечення без необхідності його встановлення чи підтримки. SaaS підходить для широкого спектру завдань - від офісних додатків до систем управління клієнтськими відносинами (CRM). [1].

Кожна з цих моделей відповідає різним потребам і дозволяє організаціям ефективніше керувати своїми ресурсами, адаптуючи технології під специфічні вимоги.

Amazon Web Services (AWS) є одним із найвідоміших постачальників хмарних сервісів, який розпочав свою діяльність у 2006 році[1]. AWS пропонує більше ніж 200 сервісів, що охоплюють зберігання даних, обчислювальні ресурси, аналітику та інтеграцію з штучним інтелектом [3]. Перевагами AWS є велика кількість доступних рішень, висока масштабованість та глобальна інфраструктура. Завдяки цьому компанія забезпечує високу продуктивність навіть для найвимогливіших клієнтів. Однак складність налаштування для новачків і висока вартість сервісів можуть стати перешкодою для малих підприємств, які не мають достатнього досвіду роботи з хмарними технологіями. AWS також допомагає організаціям вирішувати такі проблеми, як управління великими обсягами даних, автоматизація процесів і забезпечення безпеки. Їхні сервіси, зокрема Amazon SageMaker, дозволяють створювати та навчати моделі машинного навчання без потреби в потужній локальній інфраструктурі [4].

Microsoft Azure, офіційно запущений у 2010 році, спеціалізується на корпоративних рішеннях із глибокою інтеграцією в екосистему Windows [5]. Azure пропонує широкий вибір інструментів для побудови, розгортання та управління додатками в хмарі. Перевагою Azure є тісна інтеграція з іншими продуктами Microsoft, такими як Office 365 і Dynamics 365, що робить його зручним для підприємств, які вже використовують ці платформи. Водночас складність розробки кастомізованих рішень та відносно висока вартість можуть створити труднощі для стартапів і малих компаній. Azure вирішує такі проблеми, як оптимізація бізнес-процесів, управління даними та підвищення продуктивності завдяки використанню штучного інтелекту. Наприклад, сервіс Azure Cognitive Services надає можливості для аналізу тексту, зображень і мовлення [6].

Google Cloud, який з’явився у 2008 році, виділяється своєю орієнтацією на обробку великих даних і розвиток машинного навчання [7]. Google Cloud надає такі сервіси, як BigQuery для аналізу великих обсягів даних і TensorFlow для побудови моделей ШІ. Головною перевагою є високий рівень продуктивності і доступ до провідних технологій у галузі машинного навчання. Проте складність інтеграції з іншими системами та обмежена кількість дата-центрів порівняно з AWS можуть бути викликами для клієнтів. Google Cloud дозволяє компаніям вирішувати такі проблеми, як прогнозування поведінки користувачів, автоматизація рутинних задач та підвищення точності бізнес-аналітики. [8,9]

IBM Cloud, заснований у 2011 році, орієнтований на гібридні хмари та індустріальні додатки. Він пропонує широкий спектр сервісів, зокрема Watson AI, який використовується для аналізу тексту, зображень і даних [1]. Перевагою IBM Cloud є висока надійність і можливість адаптації до специфічних галузевих потреб, наприклад, у фінансовому секторі чи охороні здоров'я. Однак порівняно з конкурентами IBM Cloud має дещо менше сервісів і менш масштабовану інфраструктуру, що може обмежити його застосування для великих міжнародних корпорацій. Хмарний сервіс IBM допомагає вирішувати такі завдання, як покращення обслуговування клієнтів, автоматизація управління даними та створення інтелектуальних рішень для бізнесу.

Хмарні сервіси широко використовуються в електронній комерції, фінансах, охороні здоров'я, освіті та виробництві. Вони сприяють прискоренню інновацій, забезпечують доступність складних технологій і знижують витрати на управління інфраструктурою.

1.2. Основні підходи до впровадження ШІ в хмарні сервіси

Інтеграція ШІ в хмарні сервіси є важливою складовою сучасних технологічних змін, що дозволяють значно покращити ефективність роботи з даними, оптимізувати ресурси та підвищити рівень безпеки. Впровадження ШІ в

хмари здійснюється через низку основних завдань, які можуть бути вирішені за допомогою новітніх технологій штучного інтелекту.

1.2.1. Оптимізація управління ресурсами

Керування ресурсами у хмарних середовищах є критично важливим для підтримки стабільності роботи серверів, систем та додаткових сервісів. Однак зростання навантажень на сервери, постійні коливання у вимогах користувачів та зміна умов роботи часто спричиняють такі проблеми:

недоцільне або надмірне використання ресурсів;

неефективна еластичність ресурсів, що може спричиняти відставання в роботі системи;

високі грошові витрати на ресурси, які залишаються невикористаними в періоди пікових навантажень [9].

ШІ здатний аналізувати поточний стан серверів, здійснювати постійний моніторинг завантаження, прогнозувати піки та динамічно коригувати обсяг ресурсів у режимі реального часу. Алгоритми машинного навчання можуть автоматично налаштовувати масштабування відповідно до фактичних потреб, досягаючи максимальної ефективності використання.

Наприклад:

Прогнозування попиту на потужності: Моделі ШІ аналізують історичні дані про навантаження для передбачення майбутніх змін, що дозволяє заздалегідь нарощувати або скорочувати потужності.

Адаптивне масштабування: ШІ може автоматично регулювати кількість обчислювальних ресурсів, спираючись на поточний рівень завантаження [10].

Для успішної реалізації цього підходу у хмарних екосистемах потрібно впровадити системи моніторингу для збору даних про завантаження та використання ресурсів. Необхідно розробити або інтегрувати алгоритми для аналізу цих даних та передбачення динаміки навантажень. Також потрібно

налаштувати механізми автоматичного масштабування через хмарні платформи (наприклад, AWS, Google Cloud чи Azure), беручи до уваги отримані прогнози.

1.2.2. Підвищення безпеки

Зі збільшенням кількості користувачів та обсягів даних, що обробляються у хмарних системах, значно зростає і потенційна загроза атак. Традиційні методи захисту часто виявляються неспроможними оперативно реагувати на новітні вектори загроз. Основні виклики полягають у виявлені нових аномалій та зловмисних дій у режимі реального часу. Також, обмежена прогностична здатність традиційних систем безпеки, які не можуть заздалегідь ідентифікувати майбутні загрози. І до того, значна кількість помилкових спрацьовувань у системах захисту, що спричиняє "хибні тривоги" та перевантажує групи безпеки.

ШІ дає змогу розпізнавати нетипову поведінку у реальному часі, аналізуючи колосальні обсяги даних. Завдяки застосуванню алгоритмів глибокого навчання та машинного навчання можна досягти наступного:

ідентифікація аномалій: ШІ може бути навчений розпізнавати нетипові шаблони, які свідчать про потенційні ризики, такі як DDoS-атаки, несанкціонований доступ чи незвичне використання ресурсів;

прогнозування кібератак: Через моделювання та аналіз історичних даних про інциденти, ШІ може передбачати потенційні атаки або небезпечні ситуації на основі змін у загальній поведінці системи;

автоматизація реагування на інциденти: За допомогою ШІ можна створювати системи автоматизованого реагування, що дозволяють невідкладно вживати заходів для блокування шкідливої активності.

Для інтеграції ШІ в механізми захисту хмарних послуг необхідно використовувати методи машинного навчання для аналізу великих обсягів інформації щодо активності користувачів та серверів, розгорнути моніторингові інструменти та сенсори для збору та аналізу трафіку й операцій у реальному часі. Також доцільно розробити чи інтегрувати системи автоматичного реагування,

які використовуватимуть отримані аналітичні дані для превентивного запобігання атакам.[11]

1.2.3. Обробка даних

Хмарні сервіси обробляють величезні обсяги даних, що постійно зростають. Проте обробка таких обсягів без залучення передових технологій може бути неефективною. Основні проблеми:

- висока складність аналізу великих даних;
- потреба в швидкому аналізі для прийняття рішень в реальному часі;
- проблеми з автоматизацією створення звітів і прогнозів.

ШІ дозволяє ефективно працювати з великими обсягами даних, виявляти закономірності, прогнозувати майбутні тенденції та автоматизувати створення звітів:

Аналіз даних: Використання алгоритмів машинного навчання для автоматичного виявлення корисної інформації з великих масивів даних.

Прогнозування тенденцій: ШІ здатний прогнозувати майбутні тренди на основі історичних даних.

Автоматизація звітності: Алгоритми ШІ можуть автоматично формувати звіти, що дозволяють приймати бізнес-рішення без участі людини [11].

Для інтеграції ШІ в цей процес необхідно інтегрувати аналітичні платформи з використанням алгоритмів ШІ для обробки та аналізу даних. Також можна використовувати моделі прогнозування для виведення важливих інсайтів. Або ж розробити системи для автоматичного формування та розсилки звітів на основі даних, що надходять.

1.2.4. Персоналізація послуг

Для хмарних сервісів важливо забезпечити користувачам персоналізований досвід. Це дозволяє підвищити задоволеність та ефективність використання послуг. Проблеми полягають у:

- неадекватному адаптуванні послуг до конкретних потреб користувачів;

- проблемах з налаштуванням інтерфейсів та функціоналу для широкого кола користувачів;

- занадто високих витратах на створення персоналізованих сервісів вручну.

ШІ дає змогу створювати персоналізовані сервіси для кожного користувача, адаптуючи їх в реальному часі на основі взаємодії з системою, а саме:

Аналіз поведінки користувача: алгоритми ШІ можуть відслідковувати поведінку користувачів і пропонувати їм послуги, що найкраще відповідають їхнім вподобанням.

Адаптивні рекомендації: завдяки технологіям рекомендованих систем, ШІ може персоналізувати пропозиції та контент, що користувачі можуть знайти цікавими.

Автоматичне налаштування інтерфейсів: ШІ здатен аналізувати взаємодію з користувачами та динамічно змінювати інтерфейси сервісів.

Для інтеграції ШІ в процес персоналізації послуг можна використовувати алгоритми машинного навчання для збору даних про користувацьку поведінку та їх переваги. Варіантом також є розробка адаптивних системи персоналізації, які будуть змінювати контент і сервіси залежно від інтересів користувача. Або ж впровадити системи рекомендацій на основі аналізу великих даних та уподобань користувачів[12,13,14].

Інтеграція ШІ в хмарні сервіси дає можливість не тільки підвищити ефективність і безпеку, але й адаптувати сервіси до потреб користувачів, що є важливим для конкурентоспроможності на сучасному ринку. Шлях до успішного впровадження ШІ полягає в розробці правильних алгоритмів, використанні даних та інтеграції з існуючими системами хмарних платформ.[13]

1.3. Приклади реалізацій ШІ в хмарних сервісах

Сучасні хмарні платформи, як-от Amazon Web Services, Microsoft Azure, Google Cloud та інші, активно інтегрують штучний інтелект у власні внутрішні процеси. Якщо на ранніх етапах розвитку хмарних технологій ШІ застосовували переважно як окремі сервіси для кінцевих користувачів, то сьогодні штучний інтелект дедалі частіше стає підґрунтям внутрішньої інфраструктури та ключовим засобом для підвищення продуктивності, надійності й економічності хмарних систем. Хмарні провайдери використовують алгоритми машинного навчання для оптимізації майже кожного етапу функціонування своїх платформ: від прогнозування завантаження та управління ресурсами до підтримки безпеки, балансування трафіку та оптимізації енергоспоживання.

Одним із найбільш важливих напрямків застосування ШІ є прогнозування навантаження на обчислювальні ресурси. У великих розподілених системах завантаження завжди нестабільне. Воно змінюється залежно від часу доби, сезону, поведінки користувачів та зовнішніх факторів, як от маркетингові кампанії чи глобальні події. Традиційні системи оркестрації ресурсів часто працювали за статичними правилами. Якщо завантаження процесора долає певний поріг, система автоматично збільшує кількість серверів, але такий підхід не враховує складних патернів навантаження і може реагувати занадто повільно або ж, навпаки, передчасно. Саме тому великі хмарні провайдери дедалі частіше застосовують алгоритми прогнозного аналізу для передбачення змін навантаження наперед.

AWS інтегрує моделі машинного навчання у свій сервіс Predictive Scaling[2,3]. Ці моделі аналізують історичні дані, визначають часові закономірності, виявляють періоди сезонності та надають оркестратору прогноз на кілька хвилин, годин або навіть днів вперед. Завдяки цьому AWS може масштабувати ресурси ще до того, як навантаження зросте, що дозволяє уникати перевантажень, зменшувати затримки та економити обчислювальні потужності.

Google Cloud застосовує схожий підхід у своєму оркестраторі Borg, який є основою Kubernetes [7,8]. Borg використовує алгоритми глибинного навчання для аналізу поведінки контейнерів і прогнозування майбутнього споживання ресурсів. Системи аналізують десятки параметрів, включаючи використання пам'яті, процесорного часу, мережевої активності та особливості конкретних робочих навантажень. Модель враховує повторюваність навантаження, нетипові стрибки та взаємозалежності між різними сервісами, що дозволяє Borg приймати рішення щодо переміщення контейнерів між фізичними машинами, щоб уникати перевантажених вузлів і забезпечувати високу щільність розміщення, що, своєю чергою, оптимізує витрати на інфраструктуру, простоту обслуговування, тощо. Тобто в цьому випадку ШІ допомагає Google Cloud забезпечувати не лише високу продуктивність, а й стабільно низьку латентність для користувачів по всьому світу [7,8].

Критичним напрямком використання ШІ є управління енергоспоживанням та оптимізація роботи дата-центрів. Великі хмарні провайдери витрачають мільярди доларів на охолодження серверів і підтримання їх у робочому стані. ШІ дає змогу значно зменшити ці витрати. Найвідомішим прикладом є співпраця Google з компанією DeepMind. Алгоритми DeepMind аналізують температуру, вологість, швидкість вентиляторів, стан компресорів та інші параметри у режимі реального часу і на основі цих даних ШІ визначає оптимальні параметри роботи систем охолодження, що дозволило Google знизити витрати на електроенергію майже на 40 відсотків. Рішення працює автономно і самонавчається, постійно підвищуючи точність своїх рекомендацій. У результаті дата-центри Google стали одними з найбільш енергоефективних у світі. Подібні підходи застосовують й інші провайдери, хоча їх рішення рідше публікуються, адже такі технології часто є внутрішніми конкурентними перевагами[7,8].

Штучний інтелект також відіграє важливу роль у підвищенні надійності хмарних платформ. У великих інфраструктурах кожна секунда простою може коштувати мільйони доларів, тому швидке виявлення аномалій та інцидентів є

ключовим завданням. Хмарні системи генерують величезну кількість логів, подій, метрик та повідомлень про стан і людина фізично не здатна обробляти такі обсяги даних, тому провайдери використовують AIOps – концепцію, у якій алгоритми ШІ замінюють значну частину аналітичної роботи технічних команд. AWS має сервіс DevOps Guru, який використовує машинне навчання для аналізу логів та телеметрії, виявлення незвичних патернів поведінки та раннього попередження про потенційні інциденти. Система не просто виявляє аномалії, але й пропонує рекомендації щодо усунення проблем, аналізуючи схожі випадки з минулого[8]. Azure застосовує подібний підхід у Monitor та Sentinel, де моделі аналізують сигнали з різних систем, виділяють значущі події та зменшують кількість хибних сповіщень. Google Cloud також активно використовує машинне навчання для пошуку кореляцій між збоєм в одному сервісі та сповільненням роботи в іншому, що дозволяє швидко локалізувати першопричину інцидентів.

Окремим напрямком є оптимізація мережевого трафіку. Мережа є основою хмарної інфраструктури, і її ефективність визначає швидкість роботи додатків по всьому світу. Через складність глобальних мереж традиційні алгоритми маршрутизації часто не справляються з постійними змінами у навантаженні, затримках та пропускній здатності каналів. Тому провайдери дедалі частіше застосовують ШІ для того, щоб визначати найоптимальніші маршрути передачі даних у реальному часі. Google використовує систему B4, яка управляє внутрішнім трафіком між дата-центрами компанії. Алгоритми машинного навчання моделюють навантаження та прогнозують, які лінки найближчим часом можуть перевантажитися. Це дозволяє системі заздалегідь перерозподіляти трафік, забезпечуючи безперебійну роботу сервісів. AWS застосовує аналогічні підходи у Global Accelerator, де ШІ аналізує маршрути, визначає їх якість, час відгуку та змінює напрям трафіку таким чином, щоб мінімізувати затримки для кінцевого користувача. Azure використовує інтелектуальні алгоритми у Traffic Manager, який динамічно перенаправляє запити користувачів на найменш завантажені або фізично ближчі сервери[6].

Важливою сферою є забезпечення безпеки хмарних платформ. Безпека у хмарних системах є надзвичайно складним завданням через багаторівневу архітектуру, розподілені ресурси, величезну кількість взаємодій і потребу обробляти мільярди подій на секунду, саме тому алгоритми ШІ стали невід'ємною частиною сучасних систем кіберзахисту хмарних провайдерів.

AWS GuardDuty застосовує машинне навчання для аналізу трафіку, журналів доступу, DNS-запитів та інших даних, виявляючи аномальну поведінку, шкідливі дії чи загрози. Система працює у режимі постійного самонавчання.

Azure Sentinel об'єднує алгоритми глибинного навчання з методами кореляційного аналізу, щоб виявляти складні ланцюжки атак, які складно розпізнати класичним методам.

Google Chronicle аналізує трафік глобального масштабу, застосовуючи ШІ до величезних наборів даних, що дає змогу знаходити загрози на ранніх етапах. Завдяки використанню ШІ хмарні провайдери можуть значно швидше і точніше виявляти атаки, зменшуючи час реакції до мінімуму.

ШІ використовується для оптимізації систем зберігання даних. Хмарні платформи пропонують різні класи сховищ, що відрізняються за ціною, швидкістю доступу та призначенням. Автоматичне визначення того, які дані слід зберігати в "гарячому" швидкому сховищі, а які можна перемістити до "холодного" або архівного рівня, є складною задачею. Тому провайдери застосовують алгоритми машинного навчання, які аналізують частоту доступу до даних, їх важливість та інші параметри.

Наприклад, AWS у своєму сервісі S3 Intelligent-Tiering використовує ШІ для автоматичного переміщення об’єктів між рівнями сховища, залежно від того, як часто до них звертаються[3]. Це дозволяє клієнтам суттєво економити на витратах, не жертвуючи продуктивністю. Google Cloud застосовує подібний підхід у Auto Tiering, а Azure Storage Insights використовує аналіз метаданих і прогнозування, щоб забезпечити найефективніше розміщення файлів.

Усі ці приклади демонструють, що провідні хмарні провайдери активно застосовують ШІ у своїй інфраструктурі не лише як інструмент для кінцевих користувачів, а й як ключовий компонент внутрішніх механізмів.

ШІ допомагає прогнозувати навантаження, оптимізувати енергоспоживання, підвищувати надійність, забезпечувати безпеку, оптимізувати трафік і ефективно управляти сховищами. Впровадження інтелектуальних алгоритмів у хмарну інфраструктуру дозволяє розв’язувати завдання, які раніше вимагали значних людських зусиль, а іноді взагалі були недосяжними через масштаб або складність систем.

Цей напрямок активно розвивається, і очікується, що в майбутньому роль ШІ в управлінні хмарними ресурсами лише зростатиме. Саме ці тенденції створюють основу для розробки нових методів імплементації алгоритмів штучного інтелекту, які здатні підвищувати ефективність та надійність сучасних хмарних сервісів і стають важливою частиною сучасних досліджень у цій галузі.

2 ПРОЄКТУВАННЯ МЕТОДУ ІМПЛЕМЕНТАЦІЇ АЛГОРИТМІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ПРОГНОЗУВАННЯ НАВАНТАЖЕННЯ В ХМАРНИХ СЕРВІСАХ

2.1. Аналіз сучасних підходів до прогнозування навантаження у хмарних середовищах

Стабільність роботи хмарної інфраструктури в першу чергу залежить від здатності системи своєчасно реагувати на зміни навантаження та адаптувати наявні ресурси відповідно до потреб користувачів. У хмарному середовищі це завдання є значно складнішим, ніж у традиційних локальних або корпоративних мережах. Якщо у класичних дата-центрах потоки запитів частіше мають відносно передбачуваний характер, то у хмарних системах навантаження формується великою кількістю факторів, які не лише різноманітні, а й постійно змінюються у часі. Хмарні сервіси працюють у глобальному масштабі, обслуговуючи мільйони користувачів, що створює поведінку системи, яку вже не можна описати простими моделями.

Хмарні середовища характеризуються високою географічною розподіленістю. Користувачі взаємодіють із сервісами з різних континентів, з різним доступом до мережі, з різними часовими поясами, що створює унікальну і нерівномірну активність протягом доби. Навантаження на один і той самий сервіс у той самий момент часу може суттєво відрізнятися залежно від регіону, що унеможливлює використання єдиної універсальної моделі поведінки для конкретних, наприклад, країн[14].

Природа застосунків у хмарі дуже різноманітна. Одні сервіси створюють рівномірне, відносно стабільне навантаження, інші, наприклад, потокові системи, інтернет-магазини, банківські платформи чи системи обробки подій, мають виражену піковість. Різні категорії застосунків вимагають різного підходу до планування ресурсів. Наприклад, стрімінгові платформи генерують значне

навантаження у вечірні години та під час глобальних спортивних подій. Вебсайти електронної комерції можуть відчувати різкі сплески активності під час акцій або рекламних кампаній. Корпоративні інструменти, такі як CRM або ERP-системи, мають чітко виражену сезонність, пов’язану з робочими циклами компаній[15].

Попри загальноприйняті закономірності, діяльність користувачів, як правило, демонструє стохастичний характер. Людська поведінка не завжди піддається прогнозуванню, тому й запити, що надходять до хмарних платформ, часто бувають випадковими за своєю суттю. Трапляються ситуації, коли обсяг навантаження може зрости у кілька разів за лічені хвилини – наприклад, після виходу популярного відео, старту рекламної кампанії, впровадження нової можливості у застосунку, або навіть через непередбачувані зовнішні соціальні події. Такі обставини ускладнюють і прогнозування, адже класичні моделі, що спираються на усталені патерни, не завжди здатні їх ідентифікувати.

Складність профілю навантаження також посилюється через внутрішню організацію сучасних хмарних програм. Чимало сервісів збудовані на мікросервісній архітектурі, де кожен окремий елемент може спричиняти власні піки навантаження, котрі впливають на загальну динаміку системи. Наприклад, зростання кількості звернень до сервісу аутентифікації автоматично тягне за собою збільшення навантаження на сховища даних, кеш-системи та мережеві маршрутизатори. Це породжує каскадний ефект навантажень, який може бути вкрай важко передбачити навіть для найсучасніших систем моніторингу[15,16].

Крім того, хмарні екосистеми функціонують у режимі реального часу та вимагають миттєвого відгуку. У середовищах, де навіть незначні коливання у завантаженні процесора або затримка у мілісекундах можуть позначитися на роботі тисяч клієнтів, хибне передбачення може спричинити збої, недоступність ресурсів або, навпаки, їх надмірне резервування, що фінансово невигідно. Це перетворює завдання прогнозування не просто на бажаний елемент оптимізації, а на життєво важливий складник функціонування усієї інфраструктури[17].

Певні труднощі створює і політика масштабування, оскільки хмарні постачальники пропонують різні методи управління ресурсами, але більшість із них оперують за реактивними принципами або на основі простих порогових налаштувань. Наприклад, якщо завантаження процесора перетинає певний поріг, система додає нові ресурси, що не є оптимальним, а інколи навіть надто пізно, адже, наприклад, для банківського сектору вирішальне значення мають секунди, тоді як розширення системи за рахунок так званих «великих інстансів» може займати десятки хвилин. Отже, реактивний підхід завжди є запізнілим. Щоб інфраструктура справді працювала ефективно, вона мусить реагувати не на факт зростання навантаження, а на його перспективу. Це вимагає прогнозування з достатнім часовим горизонтом – від кількох хвилин до кількох годин, залежно від специфіки програми.[17]

Ще однією важливою особливістю є взаємозв'язок між різними показниками навантаження. Наприклад, зростання кількості запитів користувачів може спричинити не лише збільшення завантаження ЦП, а й підвищення споживання оперативної пам’яті, мережевого трафіку, кількості звернень до баз даних, кількості оброблюваних логів. Усі ці параметри змінюються по-різному і у різний час. Це формує багатовимірний образ, який традиційні одновимірні моделі часових рядів вже не можуть повністю охопити[18].

Сучасні хмарні платформи є надзвичайно гнучкими й динамічними. Сервіси здатні запускатися та зупинятися на вимогу, створювати нові дублікати, мігрувати між вузлами, змінювати конфігурації, адаптуючись до активності користувачів. Це створює ситуацію, коли навантаження може трансформуватися не лише через дії користувачів, а й через функціонування самої інфраструктури.

Усі вищезгадані чинники разом формують складне, багатошарове середовище, де прості методи прогнозування вже не дають належних результатів. Саме тому передбачення навантаження у хмарних оточеннях еволюціонувало у високоспеціалізовану науково-практичну галузь. Вона

інтегрує прийоми зі статистики, машинного навчання, глибоких нейронних мереж, обробки сигналів та оптимізації ресурсів. Кожен метод має свої переваги, але водночас і суттєві обмеження, які вимагають розробки нових рішень, здатних забезпечити високу точність та гнучкість у різних умовах.

2.1.1. Статистичні методи часових рядів

Статистичні методики прогнозування часових рядів лягли в основу перших систем планування навантаження і тривалий час домінували в аналізі поведінки обчислювальних комплексів. Їхня популярність зумовлена доступністю математичного інструментарію, відносною легкістю впровадження та достатньою точністю там, де навантаження має впорядковану структуру. Однак у хмарних середовищах, з їхньою багатофакторною та непередбачуваною суттю, ці моделі демонструють як свої сильні сторони, так і вагомі вади[15].

Статистичне передбачення ґрунтується на засаді, що майбутні значення ряду можна описати, використовуючи попередні дані та внутрішні закономірності. Головні припущення цих моделей полягають у стаціонарності, наявності лінійних взаємозв'язків або циклічності патернів. Коли комп'ютерні системи ще не були настільки глобально рознесені, а поведінка користувачів була більш передбачуваною, такі методи були об'єктивно найкращими, оскільки дозволяли ефективно моделювати типові навантаження, наприклад, на сервери баз даних чи локальні мережеві вузли.

Однією з найвідоміших і найчастіше застосовуваних моделей є ARIMA. Вона поєднує авторегресійні складові, компонент ковзного середнього та інтегруючу частину, що нівелює нестабільність ряду. ARIMA широко використовували для короткострокового передбачення завантаження ЦП, кількості запитів до вебсерверів та активності дисків у ранніх системах розподіленого моніторингу. Її перевага полягає у здатності формувати досить точні прогнози у випадках, коли часове навантаження переважно слідує лінійній динаміці або не має різких структурних змін. Проте ARIMA вимагає попередньої

верифікації на стаціонарність, аналізу функції автокореляції та ручного добору параметрів, що знижує її універсальність[15,19,20]. Основні компоненти моделі вказані на рисунку 2.1.

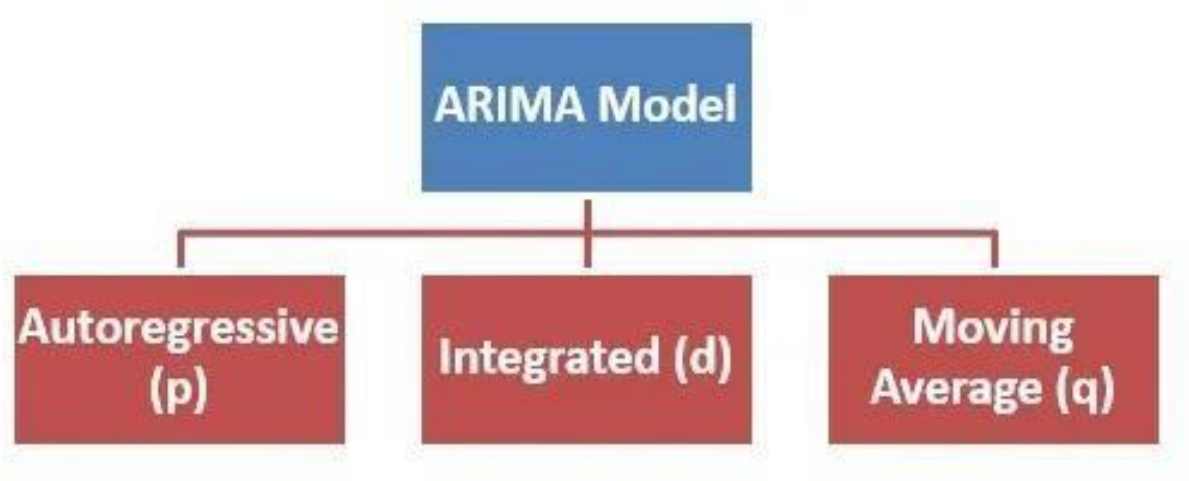


Рис. 2.1 Компоненти моделі ARIMA

Аби зменшити недоліки ARIMA, була створена SARIMA, яка враховує сезонність. У багатьох сервісах, наприклад у хмарних поштових службах, платформах електронної комерції чи корпоративних системах, навантаження підпорядковується денним, тижневим, а то й річним циклам. SARIMA здатна моделювати цю циклічність, враховуючи як короткочасні, так і тривалі коливання. Сезонні моделі демонструють добру результативність у стабільних середовищах, але стикаються з труднощами при зіткненні з непередбачуваними змінами. У хмарній інфраструктурі поява несподіваних подій, різких стрибків чи «аномальних днів» знижує точність сезонних моделей[19].

Наступний важливий напрям у статистичному прогнозуванні представлений методами, заснованими на експоненційному згладжуванні. Ці моделі були розроблені для оперативного реагування на коливання тренду та сезонності. Суть експоненційного згладжування полягає в тому, що новіші значення ряду отримують більшу вагу, аніж старі. У сфері передбачення навантаження це вкрай корисно, оскільки дозволяє швидше адаптуватися до нових тенденцій. Модель Хольта-Вінтерса стала класикою у цьому напрямку. Вона дозволяє працювати з рядом, де присутні й тренд, і сезонність, тому часто

застосовувалася для прогнозування веб-трафіку, використання оперативної пам’яті та кількості активних користувачів.

Незважаючи на свою обчислювальну ефективність та простоту, методи експоненційного згладжування погано справляються в умовах раптових змін. У ситуаціях, коли навантаження може зрости багаторазово за короткий період, ці моделі реагують із запізненням. Це особливо помітно у хмарних комплексах, де стрибки навантаження можуть бути спровоковані зовнішніми чинниками, такими як публікація контенту, різкі сплески активності користувачів чи внутрішні інфраструктурні події.

Prophet, розроблений Meta для моделювання прогнозів на великих датасетах із нестабільними та складними сезонними патернами, використовує адитивну модель із гнучким підходом до опису сезонності та святкових періодів. Це робить його ефективним у середовищах, де навантаження має помітний ритм, але також підвладне впливу аномальних подій. У багатьох випадках Prophet показує вищу точність, ніж традиційні статистичні моделі, особливо у завданнях із нестандартними чи мінливими сезонними патернами. Його здатність коректно працювати з відсутніми даними та нерівномірними інтервалами підвищує його практичну цінність.

Проте, попри широке використання, статистичні моделі мають три корінні обмеження, які стають критичними саме у хмарних середовищах:

Вони виходять з припущення, що динаміка часового ряду є хоча б частково стаціонарною. У хмарних системах це рідкісне явище. Мінливість навантаження, непередбачувані сплески та залежність від зовнішніх факторів перетворюють стаціонарність на виняток, а не на правило.

Статистичні моделі переважно є лінійними. Вони не можуть моделювати нелінійні взаємозв'язки між різними складовими навантаження, наприклад, між мережею, базами даних, дисковою підсистемою та процесором. Зміна одного параметра впливає на всю систему. Лінійні моделі це ігнорують.

Статистичні підходи, як правило, оперують в межах одного часового ряду та не залучають додаткові змінні. У хмарних системах профіль навантаження часто залежить від десятків змінних: типу запиту, часу доби, звичок користувачів, географії, внутрішньої взаємодії сервісів. Ігнорування цих факторів суттєво знижує точність передбачень.

Підсумовуючи, хоча статистичні методи залишаються цінним інструментом для базового аналізу та швидкої оцінки, їхня придатність у динамічних, масштабованих і непередбачуваних хмарних екосистемах значно обмежена. Вони добре працюють там, де навантаження має фіксовану структуру, проте виявляються недостатніми у системах з високою мінливістю. Саме ці обмеження стали ключовим поштовхом для переходу від статистичних моделей до підходів машинного та глибинного навчання, які будуть розглянуті далі.

2.1.2. Методи машинного та глибинного навчання

Поява машинного навчання стала переломним моментом у розвитку підходів до прогнозування навантаження в інформаційних системах. На відміну від статистичних методів, які за своєю природою виходять із передумови стаціонарності та лінійності часових рядів, моделі машинного та глибинного навчання здатні моделювати значно складніші залежності та патерни.

Саме тому вони швидко стали одним із ключових інструментів у прогнозуванні навантаження в сучасних хмарних середовищах. Їх застосування дає змогу аналізувати не лише попередні значення метрики, а й широкий набір супутніх параметрів, що формують навантаження: активність користувачів, контекстні ознаки, внутрішні взаємодії сервісів, топологію розгортання, стан мережі та інші фактори.

У хмарних системах навантаження часто формується з великої кількості взаємопов’язаних причин, які важко або навіть неможливо виявити за допомогою класичних моделей. Наприклад, збільшення навантаження на процесор може бути не лише наслідком зростання кількості зовнішніх запитів, а й результатом

внутрішніх процесів сервісу, таких як активація нових модулів, переміщення контейнерів між вузлами, активація механізмів реплікації бази даних чи зміна структури кешування. Моделі машинного навчання здатні враховувати комплексну природу таких сценаріїв[21].

Одним із найважливіших здобутків машинного навчання є здатність працювати з багатовимірними даними. Замість того, щоб аналізувати лише часовий ряд CPU або RAM, моделі можуть брати до уваги десятки або сотні параметрів одночасно. Наприклад, у прогнозуванні навантаження застосовують такі ознаки, як час доби, день тижня, регіон користувачів, середній розмір запиту, кількість успішних і неуспішних відповідей сервера, швидкість обробки транзакцій, стан мережевих інтерфейсів, температура вузлів, інтенсивність роботи бази даних і багато інших. Це дозволяє формувати складні, високоточні прогнози навіть у випадках, де структура навантаження є нелінійною та нестабільною.

На практиці одним із найпоширеніших інструментів стали моделі дерев рішень і їх ансамблі під назвою Random Forest[19]. Він поєднує множину дерев рішень і завдяки цьому забезпечує надійність, стійкість до шумів і здатність виявляти складні взаємозалежності. У задачах прогнозування навантаження його часто використовують як базову модель через його стабільність і високу інтерпретованість результатів. Він добре працює там, де складність патернів є високою, але обчислювальна економічність залишається важливим критерієм.

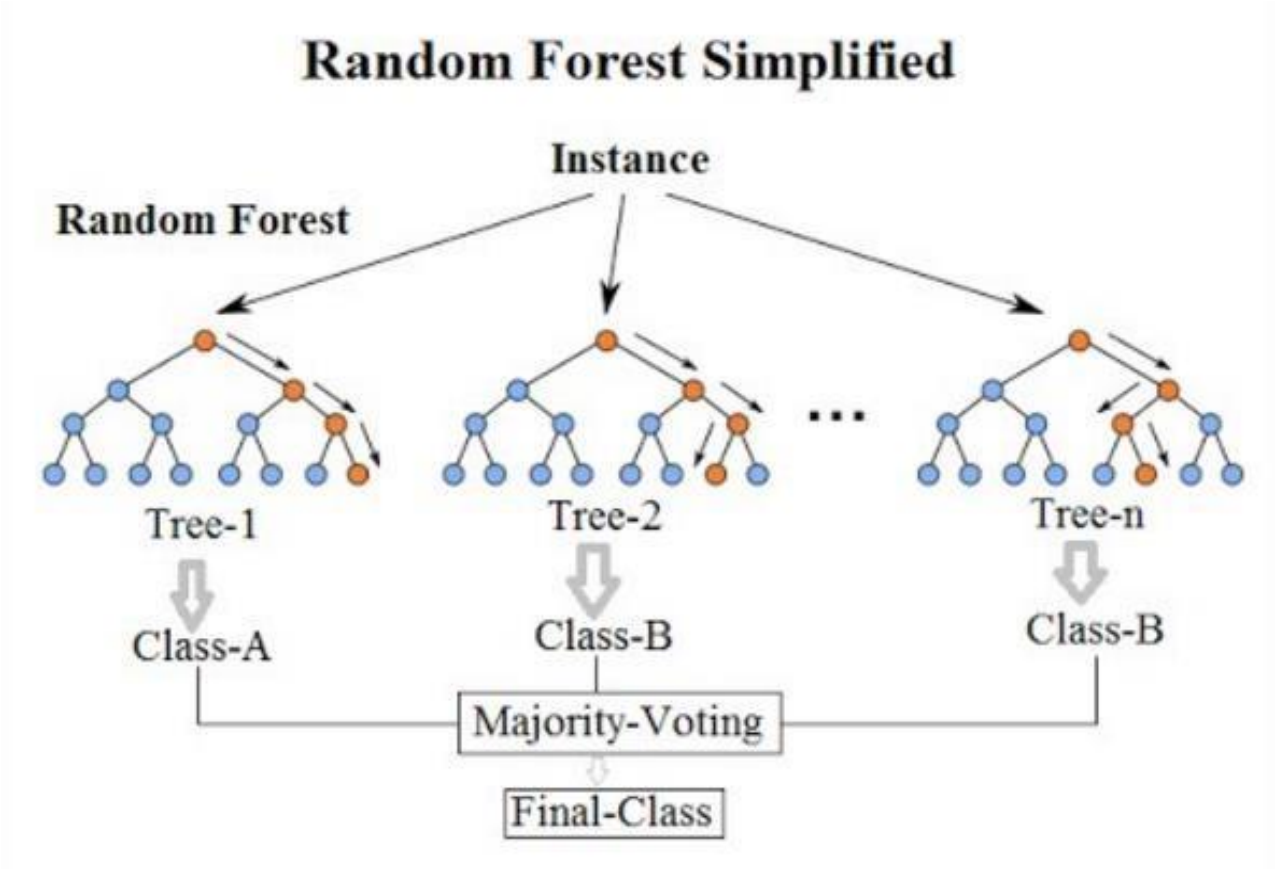


Рис 2.2. Візуалізація моделі Random Forest

Ще точніші результати демонструють моделі Gradient Boosting, такі як XGBoost, LightGBM та CatBoost. Gradient Boosting побудований на ітеративному навчанні, де кожна наступна модель намагається компенсувати помилки попередньої. Це робить ці моделі надзвичайно потужними у виявленні непрямих залежностей та невидимих закономірностей у даних. Саме через це моделі бустингу широко використовуються у прогнозуванні навантаження у хмарних платформах, де необхідна точність і швидка адаптація до різних типів поведінки користувачів[19]. Проте вони є досить важкими в налаштуванні та навчанні, тому в роботі була використана модель Random Forest.

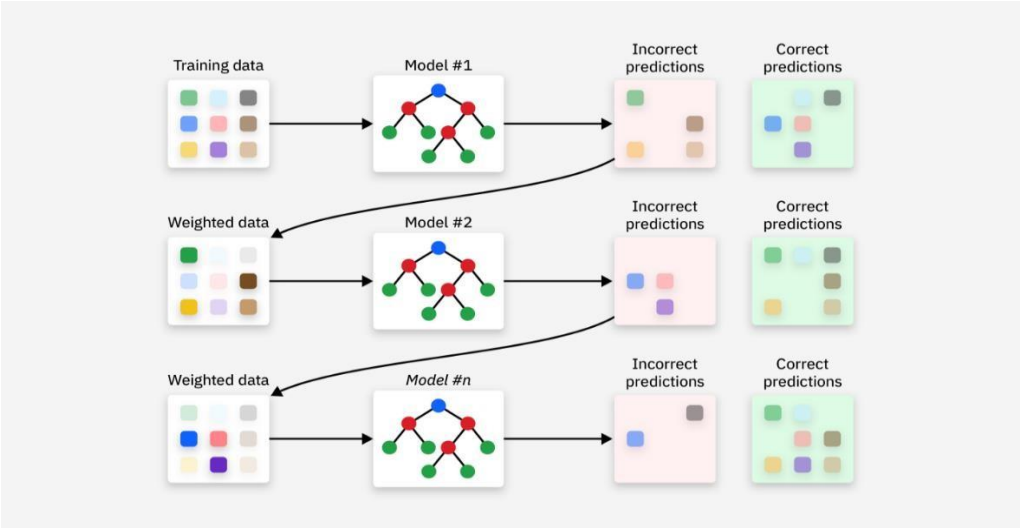


Рис. 2.3 Принцим роботи моделі Gradient Boosting

Завдяки своїй структурі Gradient Boosting добре справляється з ситуаціями, де навантаження залежить від поєднання багатьох факторів і не має стабільної сезонності. Наприклад, у системах електронної комерції активність користувачів може залежати від часу проведення акцій, маркетингових кампаній, поведінки конкурентів чи навіть погоди в окремих регіонах. Такі фактори створюють складні часові патерни, які складно описати традиційними статистичними моделями.

У той час, коли класичні ML-моделі стали відповіддю на складні нелінійні залежності, розвиток глибинного навчання привів до появи ще потужніших інструментів. Глибинні моделі, такі як LSTM та GRU, здатні аналізувати часові ряди з врахуванням довготривалої пам'яті. Це означає, що вони можуть вловлювати залежності між подіями, які розділені значним проміжком часу, що є критично важливим у багатьох хмарних системах. Наприклад, зміна поведінки користувачів може бути результатом події, яка сталася не хвилину тому, а кілька годин або днів тому. LSTM здатні моделювати такі залежності, що робить їх дуже цінними у прогнозуванні нестабільних та хаотичних патернів[19,20,21].

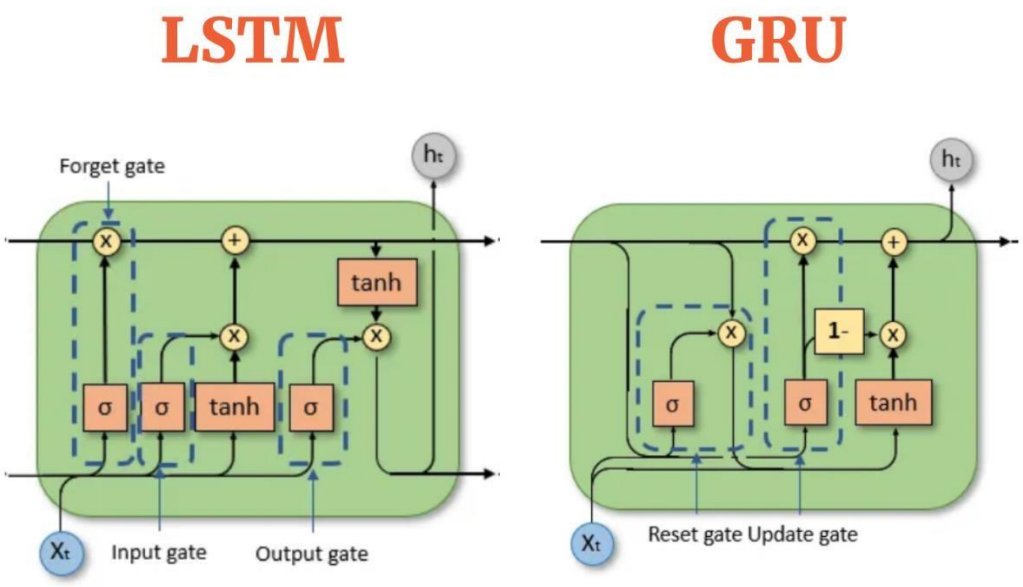


Рис. 2.4 Схеми роботи моделей LSTM та GRU

GRU є більш спрощеним варіантом LSTM, але зберігає майже усю ефективність. Завдяки меншій кількості параметрів GRU моделі працюють швидше і менш схильні до перенавчання. У хмарних системах, де швидкість реакції є критично важливою, GRU часто використовують як альтернативу LSTM для швидкого прогнозування короткострокових змін[21].

Наступною великою інновацією стали моделі на основі механізму уваги, які дозволяють моделі самостійно визначати, які частини часового ряду є важливими для прогнозу. Моделі трансформерів, такі як Informer або Autoformer, демонструють особливо високі результати на великих часових рядах. Вони здатні обробляти довгі послідовності без втрати продуктивності та можуть виявляти залежності у дуже складних структурах даних. Це має величезне значення у хмарних системах, де зміни навантаження можуть мати як локальні, так і глобальні тригери[20].

Проте глибинні моделі мають свої особливості, які роблять їх непростими у використанні. Вони потребують значних обчислювальних ресурсів, великих обсягів даних для навчання, ретельної оптимізації та постійного контролю якості. У хмарних системах це може стати проблемою, оскільки використання складних

моделей може збільшити затримку при прийнятті рішень. Крім того, характер навантаження може змінюватися настільки швидко, що модель, навчена вчора, вже не буде ефективною сьогодні. У таких випадках LSTM або трансформери можуть демонструвати гірші результати, ніж простіші ML-моделі або навіть класичні статистичні підходи.

Ще одна проблема полягає у переважно «чорному» характері глибинних моделей. Для багатьох систем критично важливо розуміти, чому саме модель прийняла певне рішення. Наприклад, у банківських або медичних застосунках, що працюють у хмарній інфраструктурі, інтерпретованість прогнозів є частиною вимог безпеки. Глибинні моделі не завжди задовольняють ці вимоги, тому моделі цього типу не будуть використані в роботі, але мають право на згадування.

Незважаючи на ці складнощі, методи машинного та глибинного навчання стали ключовими у прогнозуванні навантаження в хмарних середовищах. Їхня здатність працювати з великими обсягами даних, враховувати багатовимірні залежності, виявляти складні патерни та адаптуватися до нових умов робить їх незамінними у сучасних системах. Однак, як і статистичні методи, вони не позбавлені обмежень. Жодна окрема модель не може бути універсальною для всіх типів навантаження. Це створює потребу у нових, гібридних та адаптивних підходах, здатних об'єднати переваги різних методів і компенсувати їхні недоліки.

2.1.3. Підходи прогнозування навантаження у хмарних провайдерів

Прогнозування завантаження у провідних хмарних постачальників послуг є не просто додатковим інструментом, а наріжним каменем усього комплексу керування інфраструктурою. Гіганти на кшталт Amazon, Microsoft та Google оперують сотнями мільйонів активних клієнтів та тисячами високоінтенсивних сервісів, що вимагає надзвичайно високої точності передбачення майбутніх змін у їхній роботі. Похибка, що сягає лише кількох відсотків, може обернутися для них додатковими витратами у десятки мільйонів доларів або спричинити

серйозні збої у продуктивності. Таким чином, прогнозування навантаження у хмарі - це не лише інженерне завдання, але й критичний економічний аспект, від якого залежить прибутковість бізнесу постачальника та якість сервісів, що надаються користувачам.

У хмарній царині прогнозування вбудоване в кожен рівень архітектурної структури. Воно починається з фізичних дата-центрів, де необхідно точно планувати витрати енергії та потужності охолодження серверів, і доходить до найвищих рівнів сервісів, де масштабування відбувається на рівні контейнерів, функцій та мікросервісів. Кожен із цих провайдерів розробив власні системи для моніторингу, аналітики та прогнозування, які функціонують у режимі реального часу, охоплюючи величезні, розподілені інфраструктурні комплекси.

Amazon Web Services (AWS) інтегрує прогнозування завантаження у такі свої сервіси, як Auto Scaling та Predictive Scaling[3]. Класичні механізми масштабування AWS працювали на реактивному принципі: система збільшувала чи зменшувала кількість інстансів EC2 лише після того, як поточне навантаження досягало встановленого ліміту. Однак, зі зростанням обсягів та ускладненням навантажень, такий підхід став недостатнім. Predictive Scaling вирішує цю проблему за допомогою машинного навчання для аналізу історичних показників та формування передбачень. AWS використовує моделі, які здатні враховувати довгострокові тренди, сезонні коливання, періодичні піки та короткочасні непередбачувані сплески. Система готує прогноз заздалегідь і масштабує інфраструктуру ще до фактичного надходження пікового навантаження. Це уможливлює уникнення збоїв у роботі послуг і водночас знижує витрати на утримання надлишкових ресурсів.

Варто підкреслити, що AWS не розкриває детально, які саме моделі використовуються, проте численні дослідження та спостереження за роботою системи вказують на застосування ансамблів моделей - регресійних, імовірнісних та мереж типу LSTM. Рішення від AWS демонструє хорошу ефективність для широкого спектра клієнтів, але має загальну слабкість: воно не

бере до уваги індивідуальні особливості конкретного типу навантаження. Predictive Scaling добре справляється там, де динаміка навантаження є відносно впорядкованою, але його точність відчутно падає у випадках нестабільних чи дуже мінливих сценаріїв.

Microsoft Azure використовує свої власні методи у сервісах Autoscale та Azure Monitor. Оскільки Azure орієнтується переважно на корпоративних замовників, його методи прогнозування краще враховують більш формалізовані цикли навантаження. Корпоративні системи зазвичай демонструють чітко визначений робочий час та час відпочинку, усталені цикли обробки транзакцій та регулярні пікові періоди. Azure Autoscale аналізує ці закономірності та надає рекомендації для оптимального масштабування. Однією з важливих переваг Azure є можливість налаштовувати правила масштабування на основі складних комбінацій метрик, що дозволяє частково нівелювати недоліки самого прогнозування через гнучкішу організацію (оркестрацію)[6].

Сервіс Azure Advisor застосовує машинне навчання для виявлення відхилень та передбачення потреби у ресурсах, однак його успішність прямо залежить від сталості патернів. Він може бути менш корисним у тих середовищах, де навантаження формується через несподівані події, характерні для сучасних, швидкозмінних вебдодатків. Як наслідок, Azure схиляється до вбудовування прогнозування скоріше як системи рекомендацій, аніж як суворого механізму автоматичного масштабування.

Google Cloud Platform (GCP) використовує унікальний підхід до прогнозування завантаження завдяки своїй системі оркестрації Borg, на якій згодом було засновано Kubernetes. Borg накопичував дані з мільйонів контейнерів, і ці дані стали основою для машинного навчання, яке прогнозувало, скільки ресурсів споживатиме кожен окремий контейнер. Моделі Google враховують десятки різних характеристик, включаючи вид завдання, тривалість життя контейнера, структуру обчислень та поведінку кінцевих користувачів. Проте Borg також базується на універсальній моделі, яка не враховує специфіку

різних типів навантаження. Він формує загальний прогноз, який добре працює для середньостатистичного контейнера, але може виявитися неточним для навантажень зі складною нелінійною динамікою[8].

Іншим прикладом є Google Autopilot - система, що автоматично визначає необхідний обсяг ресурсів для контейнерів у Kubernetes. Вона завбачливо передбачає піки навантаження та заздалегідь резервує потрібні потужності. Однак, і Autopilot також функціонує на загальній моделі, яка не пристосована до різних класів навантажень.

Усі згадані провайдери використовують системи прогнозування як ключовий компонент у своїх механізмах автоматичного розширення (autoscaling), але всі вони стикаються з однією суттєвою проблемою: відсутністю здатності адаптивно обирати модель. Універсальні підходи, що застосовуються наразі, намагаються бути достатніми для повного спектру сценаріїв, проте не є оптимальними для жодного конкретного класу навантаження. Одна модель не може однаково успішно функціонувати як при регулярних циклах, так і при стохастичних піках чи хаотичних патернах, притаманних архітектурам на основі мікросервісів.

Ще однією перешкодою є питання масштабування самих цих систем. Хмарні оператори не можуть дозволити собі створювати індивідуальну модель для кожного клієнта або кожного сервісу, оскільки це спричинило б надмірні обчислювальні витрати. Через це їхні методи являють собою певний компроміс між універсальністю та точністю. Як наслідок, прогнозування демонструє хороші результати лише за середніх умов, але може втрачати свою ефективність у випадках екстремальних або нестандартних ситуацій.

Окрім того, провайдери часто вбудовують прогнозування лише як частину ширшого механізму прийняття рішень. Навіть за наявності високоточної моделі автоскейлінгу, фінальне рішення може бути відкориговане через програмні обмеження політики, налаштування мінімальної кількості інстансів (екземплярів), правила розгортання, час запуску ресурсів, економічні міркування

щодо масштабування та інші технічні й фінансові фактори. Це означає, що навіть найбільш точний прогноз має обмежений реальний вплив, якщо система керування ресурсами не здатна до гнучкості, або якщо політика масштабування накладає жорсткі обмеження.

Попри усі ці складнощі, підходи, втілені хмарними провайдерами, встановили галузевий стандарт. Вони продемонстрували, що машинне навчання є ефективним засобом для прогнозування, і що інтеграція прогнозних моделей у системи оркестрації є стратегічно важливою. Водночас, обмеження цих систем чітко вказують на потребу у розробці нових методів, які б не тільки моделювали майбутнє навантаження, але й динамічно пристосовували свій підхід відповідно до його типу, особливостей та змін з часом. Це закладає основу для створення більш гнучких та самонавчальних мультимодельних систем, здатних подолати недоліки існуючих рішень.

2.1.4. Проблеми та обмеження існуючих рішень

Незважаючи на значний прогрес у сфері розробки технологій передбачення завантаження, актуальні підходи все ще мають низку визначальних прогалин, що ускладнюють їх ефективне функціонування у реальних хмарних середовищах. Ці проблеми мають різну природу: деякі кореняться у математичних властивостях самих моделей, інші – у конструктивних особливостях хмарних платформ, а треті – у специфіці динаміки користувачів та додатків у масштабованих конфігураціях. Вкрай важливо детально розібрати кожне з цих обмежень, адже саме вони обумовлюють потребу у створенні нового, адаптивного підходу, здатного нівелювати недоліки існуючих рішень.

Однією з найбільш вагомих вад сучасних методик є їхня вразливість перед мінливим характером робочого навантаження. У хмарних системах завантаження не лише змінюється з плином часу, а часто трансформує свою сутність кардинально. Ресурс, що демонстрував чіткі добові коливання протягом тижня, може раптово перейти до хаотичної (стохастичної) активності після

випуску нової опції чи старту маркетингової акції. У подібній ситуації модель, побудована на архівних даних, втрачає здатність коректно передбачати майбутні значення. Вона спирається на припущення, що майбутнє буде дзеркальним відображенням минулого, хоча насправді поведінка клієнтів або внутрішня логіка застосунку можуть зазнати повної перебудови.

Чимало актуальних моделей просто не спроєктовані для таких різких зсувів, що призводить до втрати точності прогнозу в критичний момент, коли система вимагає миттєвого нарощування чи, навпаки, скорочення ресурсів. Ці трансформації можуть бути зумовлені діями користувачів, випуском оновлень, рекламними кампаніями, запровадженням нових можливостей, зміною інтенсивності запитів чи іншими подіями. Прогнозні моделі, налаштовані на певний тип даних, часто демонструють погіршення точності, коли структура навантаження зазнає несподіваних змін. Це стосується як статистичних методів, які виходять із відносної сталості параметрів часового ряду, так і моделей на базі машинного чи глибокого навчання, які можуть бути надмірно "перенавчені" історичними трендами. Як наслідок, прогноз стає недостатньо надійним якраз тоді, коли інфраструктура потребує швидкої та точної реакції, ставлячи під загрозу стабільність функціонування сервісів.

Іншим серйозним ускладненням є обмеженість сфери застосування більшості моделей до конкретних видів навантаження. Кожен алгоритм має свою "оптимальну зону", де він показує найкращі результати. Скажімо, статистичні підходи ефективні при роботі з навантаженням, яке має усталену циклічність, але вони не спрацьовують у випадках нерегулярної зміни патернів або при наявності численних нетипових сплесків. Так само, моделі глибокого навчання добре опрацьовують нерівномірні дані, проте потребують тривалого архіву спостережень і можуть працювати некоректно, якщо доступний масив даних є коротким або його структура динамічно змінюється. У підсумку, моделі, які є відмінними у певних локальних сценаріях, виявляються непридатними в інших, і жодна з наявних платформ не вміє автоматично переключатися між різними

засадами прогнозування залежно від поточної ситуації. Нейронні мережі глибокого рівня (як-от LSTM чи GRU), здатні фіксувати довгострокові взаємозв'язки, вимагають великих обсягів навчального матеріалу та значних обчислювальних потужностей, а також можуть надмірно реагувати на шумові елементи або неточно відображати локальні специфіки. Через це жодна модель не може бути універсальною: алгоритм, що ідеально підходить для одного підтипу навантаження, може повністю провалитися на іншому. Це означає, що нинішні рішення завжди є компромісом: або вони сфокусовані на конкретному сценарії, або є загальними, але не надто точними [18].

Важливою перешкодою також слугує швидка втрата актуальності моделей у реаліях хмарних систем. Навіть найбільш ретельно налаштована модель, заснована на машинному чи глибокому навчанні, з часом починає втрачати свою інформативну цінність. Це викликано тим, що хмарні платформи - це надзвичайно динамічні середовища, де можуть змінюватися внутрішня архітектура, конфігурації, кількість сервісів, типи оброблюваних даних та методи їхнього опрацювання. Наприклад, заміна системи баз даних, впровадження нового механізму кешування або модернізація балансувальника може докорінно змінити характер робочого навантаження. Модель, тренована на даних, що передували цим змінам, більше не може адекватно відображати взаємозв'язки між різними показниками. Періодичне повторне навчання частково вирішує цю проблему, але не завжди гарантує необхідну оперативність. Зміни можуть настати неочікувано, і модель, яка ще вчора давала точні оцінки, сьогодні ризикує стати джерелом хибних рішень. В умовах, коли система вимагає негайного масштабування або оптимального розподілу ресурсів, такі помилки стають критично небезпечними [19].

Додаткові труднощі виникають через багатомірність хмарних навантажень. У дійсності навантаження формується внаслідок взаємодії великої кількості чинників: мережевих, процесорних, дискових, транзакційних, а також логічних зв'язків між різними сервісами. Скажімо, зростання кількості

HTTPзапитів може спричинити не тільки підвищення завантаження ЦП, але й активізацію роботи бази даних, збільшення обсягу журналювання, зміну стану кешу та посилення внутрішньої взаємодії між мікросервісами. Статистичні моделі як правило оперують одним часовим рядом, а тому не можуть охопити картину повністю. Навіть складніші ML-моделі можуть пропустити наростаючі залежності, якщо вони не відображені явно у наборі ознак. Високорівневі нейронні мережі частково зменшують це обмеження, але вони потребують гігантського обсягу даних для тренування і схильні концентруватися на одних патернах, ігноруючи рідкісні, але істотні зв'язки. У результаті прогноз може бути математично точним, але не відображати реальний стан системи, який формується завдяки взаємодії десятків процесів.

Важливою проблемою є також відсутність повноцінних механізмів самооновлення (самонавчання) у переважній більшості підходів. Хмарні системи функціонують у режимі реального часу, і поведінка сервісів може змінюватися щогодини. Однак прогнозні системи, що використовуються сьогодні, зазвичай оновлюються лише періодично або напівавтоматично. Це означає, що між циклами перенавчання модель оперує застарілою інформацією. Аномальні явища, що виникають у системі, не включаються в модель миттєво, а потрапляють до неї лише після чергового етапу перетренування. Це створює небезпечні часові "проміжки", коли механізм масштабування залежить від прогнозів, які вже не відповідають актуальним патернам навантаження. У багатьох випадках це може призвести до недостатнього виділення ресурсів або, навпаки, до надмірного резервування потужностей, що тягне за собою зростання фінансових витрат [24].

Викликом є і складність пояснення результатів роботи сучасних прогнозних моделей, оскільки у хмарних системах часто є критично важливим не лише мати сам прогноз, а й розуміти фактори, що його сформували. Наприклад, адміністратору може знадобитися обґрунтування, чому модель очікує стрімкого зростання завантаження: через пік трафіку, внутрішні операції,

архітектурні зміни чи інші чинники. Статистичні моделі пропонують високий рівень інтерпретованості, але їхня точність є невисокою; ML-моделі знаходять компроміс між точністю та можливістю пояснення, але їхні висновки часто є неповними. Найпотужніші ж моделі глибокого навчання практично неможливо інтерпретувати, що ускладнює їхнє застосування у сферах, де обґрунтованість рішень має високу вагу (наприклад, у фінансових чи медичних інформаційних комплексах).

Сучасні комерційні хмарні оператори також стикаються з обмеженням стандартизації своїх моделей. Через колосальну кількість користувачів і широкий спектр застосуваних додатків, вони змушені впроваджувати єдині системи передбачення для всіх клієнтів, незалежно від їхньої унікальної специфики. Це тягне за собою ситуацію, коли модель, ідеально працююча для типових корпоративних завдань, може бути абсолютно нерезультативною для специфічного сервісу зі складною, хаотичною динамікою навантаження, а відсутність індивідуального налаштування призводить до того, що універсальні алгоритми знижують якість передбачення в окремих, персоналізованих випадках.

Усі означені обмеження підкреслюють, що наявні методи передбачення завантаження не володіють достатньою гнучкістю, здатністю до адаптації та універсальністю. Хмарні інфраструктури потребують таких методологій, які мали б можливість враховувати не лише архівні показники, але й тип навантаження, контекст його виникнення, особливості зміни патернів та взаємозалежності між компонентами системи. Такий метод мав би мати здатність до автоматичного пристосування до змін, забезпечуючи при цьому високу точність прогнозів навіть у високодинамічних середовищах [24].

2.2 Класифікація типів навантаження у хмарних сервісах

Основними характеристиками робочих навантажень, що генеруються у хмарних сервісах, є їхня висока заплутаність та плинність. На противагу локальним установкам, де зміна навантаження часто є передбачуваною і залежить від обмеженого кола чинників, у хмарній інфраструктурі навантаження постає внаслідок взаємодії великої кількості незалежних та взаємопов'язаних процесів. До таких процесів віднесено таку диверсифікацію: патерни використання користувачами із різних часових поясів, внутрішню діяльність самих сервісів, параметри мережі, фізичне розташування обчислювальних центрів, механізми управління ресурсами, автоматичне розширення чи скорочення потужностей, а також зовнішні події, які неможливо безпосередньо виміряти чи включити до моделі. Ця багатогранність формує складне інформаційне поле, де навіть незначні флуктуації здатні спричинити суттєві модифікації у структурі навантаження.

У цьому контексті значущість набуває групування (класифікація) видів навантаження. Воно дає змогу виявити загальні закономірності, які можна експлуатувати для підвищення точності аналітичних моделей прогнозування. Навантаження може проявлятися як періодичні, трендові, випадкові (стохастичні), змішані чи імпульсні форми, і кожен із цих форматів вимагає іншого підходу до аналізу та передбачення. Однак, ускладнення полягає в тому, що фактичні часові послідовності, які відображають навантаження, рідко відповідають лише одному чіткому типу. Як правило, вони містять у собі декілька взаємовиключних типів поведінки, поєднуючи в собі як упорядкованість, так і непередбачуваність, циклічність із хаотичністю, плавні зміни з різкими стрибками.

Отже, необхідно не лише формально ідентифікувати типи навантажень, а й усвідомити сутність їхнього сумісного існування та взаємного впливу.

Однією з причин, що ускладнюють класифікацію, є масове впровадження компонентно-орієнтованої мікросервісної архітектури у хмарних програмах. На відміну від монолітних рішень, де навантаження в більшій чи меншій мірі розподіляється по всіх елементах системи, системи на базі мікросервісів складаються з десятків або сотень окремих модулів, кожен з яких формує свій власний часовий ряд навантаження. Ці ряди можуть мати кардинально різну природу: один модуль може демонструвати сталу періодичність, інший - випадкові коливання, третій реагує на зовнішні тригери великими піками, а четвертий показує поступове наростання тренду протягом тривалого часу. Додатково, між цими модулями існують складні логічні взаємозв'язки, і навантаження на один сервіс може прямо чи опосередковано впливати на інші.
За таких умов класифікація типів навантаження перетворюється не просто на допоміжний інструмент опису, а на базовий принцип для розробки прогнозних механізмів.

Групування є важливим і тому, що різні різновиди навантаження мають різний рівень можливості для передбачення. Регулярні цикли легко піддаються моделюванню за допомогою статистичних методик, тоді як стрибки та випадкові флуктуації вимагають застосування моделей глибокого навчання або ансамблевих підходів. Наявність гібридних структур ускладнює ситуацію ще більше, оскільки непередбачувана поведінка може накладатися на впорядковані цикли, створюючи заплутані конфігурації патернів. Відсутність класифікації веде до того, що прогностичні моделі застосовуються без належного врахування природи вхідних даних, а, отже, їхня результативність знижується. Якщо модель, оптимізована для сезонних рядів, буде використана для стохастичного навантаження, вона виявиться невідповідною, а застосування надто складної нейронної мережі до стабільного ряду лише спричинить непотрібні обчислювальні витрати без покращення точності[25].

Існує ще одна вагома причина для необхідності класифікації - це вплив різних форматів навантаження на механізми автомасштабування. Хмарні

платформи формують рішення про масштабування, спираючись на прогнозні моделі. Якщо прогноз хибний або модель не враховує реальної сутності навантаження, система може або надмірно розширити ресурси (що призведе до надлишкових фінансових витрат), або, навпаки, не додати їх (що може спричинити погіршення якості послуг та невдоволення користувачів). Наприклад, стрімкі піки вимагають швидких реактивних рішень та більшої опори на адаптивність, тоді як сезонні коливання дозволяють завчасно планувати розширення потужностей. Без усвідомлення типу навантаження розробка найефективнішої стратегії масштабування є нездійсненною.

Під час процесу групування слід брати до уваги не лише видиму форму часового ряду, але й його контекстуальне оточення. Деякі сервіси можуть мати патерни, що повторюються, але їхнє походження пов'язане із зовнішніми подіями. Приміром, соціальні медіа можуть спостерігати сплески активності під час великих спортивних змагань чи прем'єр кінофільмів. Інші системи можуть демонструвати сезонність, проте тут важливими є не самі періодичні коливання, а внутрішні бізнес-цикли, як-от регулярна звітність. Усе це впливає на вибір прогностичних моделей та методів, які слід застосовувати для коригування прогнозу в реальному часі[25].

Класифікація різновидів навантаження - це обов'язковий етап у розробці мультимодельного підходу до прогнозування, оскільки вона дає змогу визначити, яка саме модель буде найпродуктивнішою у конкретному випадку, наскільки глибокий аналіз даних є необхідним і наскільки швидко інфраструктура повинна реагувати на зміни. Без класифікації метод прогнозування не зможе бути ефективно впроваджений, особливо у змішаних та випадкових середовищах, характерних для сучасних хмарних платформ. Саме тому розуміння та формалізація типів навантаження становить наріжний камінь для запропонованого адаптивного методу, який передбачає можливість зміни моделі залежно від природи вхідних даних.

2.2.1. Основні типи навантаження

Сезонне й циклічне навантаження

Сезонне навантаження є одним із найбільш поширених типів для хмарних сервісів. Воно проявляється у вигляді регулярних, повторюваних циклів, які відображають закономірну поведінку користувачів або специфіку бізнеспроцесів. Добові цикли є типовими для вебсайтів, новинних платформ, соціальних мереж, корпоративних систем. У таких випадках активність зростає у робочі години та зменшується у нічний час. Тижнева сезонність характерна для бізнес-сервісів, де інтенсивність обробки транзакцій суттєво знижується у вихідні. Існує й річна сезонність, наприклад у торгівлі або системах бронювання.

Хоча сезонне навантаження на перший погляд є одним із найбільш передбачуваних типів, у реальних хмарних середовищах сезонність має складну природу. Амплітуда циклів може змінюватися залежно від зовнішніх факторів: святкових періодів, рекламних кампаній, подій у світі або навіть погодних умов. Крім того, сезонність у реальних застосунках часто змішується з іншими типами поведінки: добовий цикл може містити хаотичні коливання активності всередині пікових годин. Це створює значні труднощі для прогнозування, оскільки модель має розрізняти регулярні компоненти від нерегулярних[14,15].

Пікове та сплескове навантаження

Пікове навантаження є одним з найбільших викликів для хмарних платформ. Воно характеризується короткочасними, але дуже різкими підвищеннями активності. Ці піки можуть виникати практично миттєво та мати значну амплітуду, що становить ризик не лише для продуктивності окремого сервісу, але й для цілостності всієї інфраструктури. Наприклад, стрімінгові сервіси відчувають піки під час трансляцій популярних подій, інтернет-магазини – під час початку розпродажів, соціальні мережі - після вірусної публікації. Мікросервісні системи також створюють сплескові сценарії тоді, коли внутрішні вузли отримують неочікувано багато запитів через каскадні ефекти.

Особливо складною є природа походження піків. Часто пікові події не мають жодних попередніх індикаторів у часовому ряду. Вони виникають через зовнішні фактори, яких прогнозна модель просто не може побачити у даних. Через це навіть глибинні моделі, які здатні виявляти слабкі патерни, часто або пропускають пік, або неправильно оцінюють його амплітуду. Крім того, піки можуть мати різну тривалість і різні форми - від коротких імпульсів до тривалих хвиль навантаження. Це також ускладнює прогнозування[14,15].

Стохастичне та хаотичне навантаження

Стохастичні ряди відрізняються відсутністю чіткої структури - вони не мають ні сезонності, ні стабільного тренду. Така поведінка характерна для систем, де навантаження спричиняється подіями, що відбуваються у реальному часі, або внутрішніми механізмами, які генерують високий рівень шуму. До таких сервісів належать біржові торгові системи, ігрові сервери, подієвоорієнтовані платформи, а також частина мікросервісних архітектур, де залежності між компонентами створюють складне стохастичне середовище.

Прогнозування стохастичного навантаження є надзвичайно складним. У таких рядах немає чітких закономірностей, на які могла б спиратися модель. Нелінійні залежності часто мають слабку структуру, і навіть сучасні глибинні мережі можуть помилково інтерпретувати шум як закономірність або навпаки - пропускати важливі слабкі сигнали. Для такого виду навантаження найчастіше потрібні адаптивні, самонавчальні або ансамблеві моделі, які можуть перебудовуватися у режимі реального часу[15,16].

Змішані типи навантаження

Змішані ряди трапляються найчастіше та є найскладнішими для аналізу. У таких сценаріях різні сегменти часового ряду мають різні характеристики. В один проміжок часу дані демонструють сезонність, у інший – сталися стохастичні коливання, а під час зовнішніх подій виникають піки. Багато хмарних систем мають саме таку природу. Мікросервісні архітектури, наприклад, поєднують

регулярну активність користувачів із нерегулярною поведінкою внутрішніх механізмів, створюючи складні комбінації патернів.

Жодна окрема модель не здатна якісно працювати у змішаних сценаріях, оскільки вони вимагають гнучкого переходу між різними техніками прогнозування. У таких умовах надзвичайно важливою стає можливість класифікації типів навантаження та адаптивне перемикання алгоритмів - саме ця проблема зумовлює потребу у гібридному методі[15].

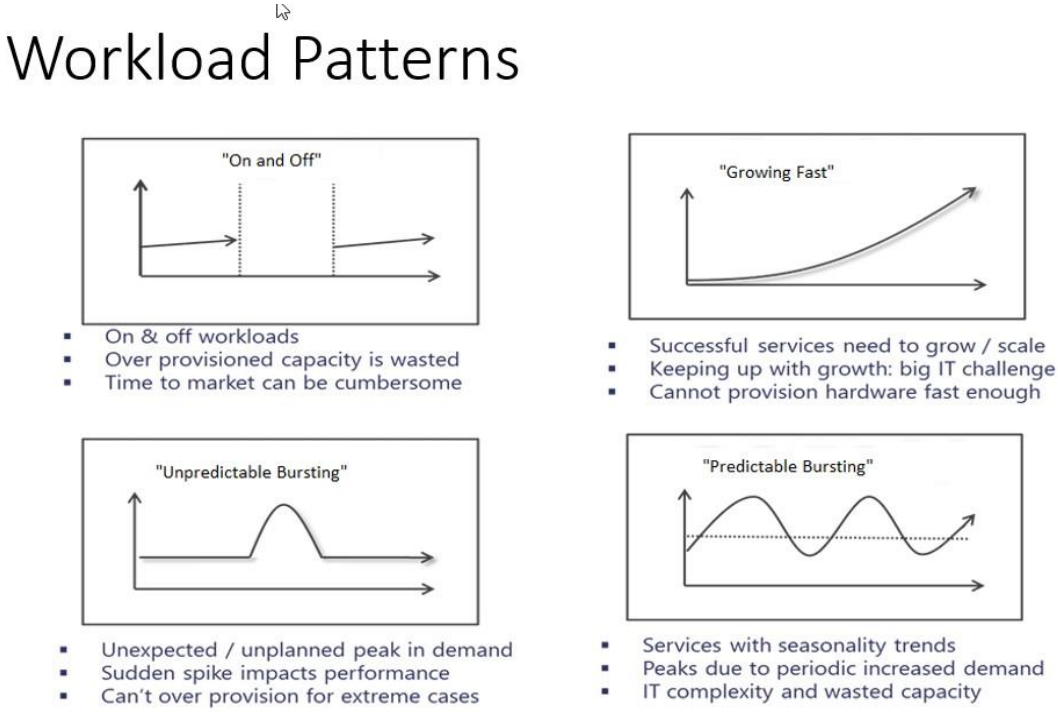


Рис 2.5 Типи навантажень в хмарному середовищі

2.3.2. Особливості прогнозування різних типів навантаження

Прогнозування стабільних рядів

Стабільні ряди є найлегшими для прогнозування з точки зору математичного моделювання. Такі ряди добре піддаються аналізу за допомогою лінійних моделей, методів згладжування або навіть простих фільтрів. Проте існують нюанси, які роблять працю зі стабільними рядами не завжди тривіальною. Навіть у стабільних сценаріях періодично виникають аномалії, які можуть суттєво вплинути на прогноз. Якщо модель не має механізму виявлення та корекції викидів, навіть один аномальний сплеск може спотворити прогноз на

тривалий період. Крім того, стабільне навантаження іноді маскує повільні тренди, і модель повинна мати можливість їх виявляти[15].

Прогнозування сезонного навантаження

У сезонних рядах найбільшу складність створює змінна сезонність. Амплітуда сезонних коливань може змінюватися залежно від внутрішніх або зовнішніх факторів. Якщо модель припускає, що сезонний патерн є сталим, прогноз буде помилковим у моменти зміни ритму роботи системи. Також важливо враховувати складну структуру сезонності - у деяких сервісів можуть одночасно існувати кілька циклів, наприклад добовий і тижневий. Для таких випадків необхідний гібридний підхід або моделі з можливістю окремого моделювання кількох сезонних компонентів[15].

Прогнозування пікових навантажень

Прогнозування пікових подій потребує моделей, здатних виявляти слабкі сигнали перед раптовими змінами. В окремих випадках такі сигнали є внутрішніми: задовго до піку можуть змінюватися вторинні метрики, наприклад час відповіді бази даних або поведінка черг. Глибинні моделі можуть їх розпізнавати, але часто помиляються, оскільки амплітуда сигналу слабка і не завжди очевидна. Найбільша проблема полягає в тому, що частина піків не містить жодних попередніх ознак, тому прогнозування у таких сценаріях є скоріше ймовірнісною оцінкою, ніж точним передбаченням. Тут важливу роль відіграє адаптивність і здатність системи швидко виявляти факт виникнення піку[15,16].

Прогнозування стохастичних рядів

Стохастичні ряди потребують моделей, які можуть виявляти слабкі нелінійні закономірності у шумі. Проблема полягає у тому, що надто складні моделі можуть почати «бачити» структуру там, де її немає, тобто перенавчатися на шум. Це викликає нестабільні прогнози, які можуть коливатися сильніше, ніж сам ряд. Також стохастичні ряди часто мають нерівномірні інтервали або кореляції з іншими параметрами, які важко виміряти. Для таких сценаріїв

оптимальною є мультимодельна стратегія, де прості моделі фільтрують загальну структуру, а складні - працюють лише у моменти структурних змін[15,16].

Прогнозування змішаних типів навантаження

Змішані ряди вимагають найбільш складного підходу. У таких сценаріях прогнозна модель повинна вміти адаптуватися до змін характеру навантаження, які можуть відбуватися у межах одного часу. Це означає, що модель повинна мати механізм аналізу локальних властивостей. Сучасні глибинні моделі частково справляються з цим завдяки здатності аналізувати довгі залежності, але вони не завжди можуть розрізнити сезонні та стохастичні компоненти. Адаптивний гібридний метод, що може перемикатися між різними алгоритмами або об'єднувати їхні прогнози, є найбільш підходящим рішенням для таких рядів[15,16].

2.3 Постановка задачі та вимоги до методу

Прогнозування навантаження у хмарних середовищах є ключовим елементом ефективного управління обчислювальними ресурсами. Від точності прогнозу залежить не лише стабільність роботи сервісів, а й економічна ефективність використання інфраструктури. З огляду на те, що сучасні хмарні системи характеризуються високою динамічністю, непередбачуваністю та складними багатовимірними залежностями між компонентами, постає необхідність у створенні нового методу, який здатний працювати у широкому спектрі сценаріїв і адаптуватися до різних типів навантаження.

Попередній аналіз існуючих рішень показав, що традиційні підходи - статистичні моделі, алгоритми машинного навчання та глибинні нейронні мережі хоча й мають свої сильні сторони, не здатні забезпечити необхідний рівень універсальності та адаптивності. Особливої ваги набуває здатність моделі коригувати свою поведінку відповідно до типу навантаження, яке спостерігається у поточний момент. Системи хмарного масштабування

потребують прогнозів, які не просто відображають можливе майбутнє, а враховують контекст, характер зміни патернів та взаємозалежності між метриками.

Постановка задачі у рамках цього дослідження полягає у розробці методу, який забезпечує високу точність прогнозування незалежно від характеру навантаження - сезонного, стохастичного, пікового чи змішаного. Метод повинен уміти автоматично обирати модель прогнозування, найбільш відповідну до конкретної ситуації, враховувати зміни у структурі історичних даних та адаптувати свої параметри під впливом нових патернів.

Метод повинен забезпечувати не лише точність прогнозів, а й стабільність рішень, щоб уникати ситуацій надмірного масштабування сервісів або зайвих відмов від ресурсів.

У контексті сучасних вимог до хмарних платформ метод має бути не лише точним, а й ефективним в обчисленні. Надмірно складні моделі глибинного навчання, хоча й забезпечують високу точність, не завжди можуть бути застосовані в реальному масштабуванні через обмеження швидкості прийняття рішень. Тому метод повинен включати механізм балансування між точністю та обчислювальною складністю прогнозу. Рішення мають прийматися у реальному часі, не створюючи додаткових затримок у системі.

Постановка задачі зводиться до необхідності створення адаптивного, мультимодельного методу прогнозування навантаження, який може враховувати різні типи патернів, обирати найкращий алгоритм у конкретний момент часу, реагувати на зміни структури даних та бути інтегрованим у процес автоматичного масштабування хмарних ресурсів. У наступних підрозділах детально описано формальну постановку задачі, вимоги до методу та критерії оцінювання його ефективності[18].

2.3.1. Формалізація задачі прогнозування

Задача прогнозування навантаження формулюється як задача передбачення майбутніх значень певної метрики на основі попередніх спостережень та супутніх ознак. У загальному випадку система моніторингу хмарної інфраструктури генерує часові ряди, що відображають різні характеристики навантаження: використання процесорного часу, споживання оперативної пам'яті, кількість запитів на секунду, затримку мережевих операцій, активність дисків та інші параметри. Нехай задано часовий ряд

$$X = \{x_t\}, \tag{2.1}$$

де значення x_t вимірюється у моменті часу t. Необхідно побудувати функцію прогнозування $F(x_t,x_{t-1},...,x_{t-n})$, яка повертає оцінку навантаження у майбутній момент $t+k$. Однак у хмарних системах задача виходить далеко за межі однієї метрики. У реальності навантаження формується через взаємодію десятків параметрів, що утворюють багатовимірний часовий ряд. Тому задача набуває вигляду

$$X = \{x_t^1, x_t^2, \dots, x_t^m\}, \tag{2.2}$$

де кожне $x_t{}^i$ представляє окрему метрику. Прогнозування вимагає побудови багатовимірного відображення, здатного враховувати різні типи залежностей - як короткострокові, так і довгострокові. У цьому контексті важливою вимогою є здатність методу працювати з різними типами навантаження, які мають власну структуру і властивості, а також визначати характеристики поточного ряду перед побудовою прогнозу.

Задача також ускладнюється потребою у врахуванні масштабу часу. У деяких випадках прогноз має бути обчислений на наступні хвилини, тоді як в інших - на години або доби наперед. Інфраструктура масштабування ресурсів формує різні запити залежно від потреб сервісу, тому метод повинен бути гнучким щодо часових горизонтів і адаптувати механізм прогнозування відповідно до них.

2.3.2. Функціональні та нефункціональні вимоги до методу

Функціональні вимоги до методу прогнозування навантаження визначають, якою мірою він має відповідати характеру реальної хмарної інфраструктури. Однією з ключових вимог є здатність працювати з різними типами навантаження, оскільки реальна поведінка системи може суттєво варіюватися від стабільних і плавних циклів до різких, стохастичних коливань. Метод повинен включати механізм аналізу форми часового ряду, виявляти наявність сезонності, стохастичних коливань або трендових змін, а також автоматично підлаштовуватися до кожної з цих структур. Це потребує попередньої класифікації навантаження, яка буде базуватися на статистичних або машинних ознаках, таких як поведінка автокореляційної функції, розподіл амплітуд коливань або ступінь нелінійності. Такий механізм має бути інтегрований у метод як самостійний блок, який передує вибору моделі прогнозування.

Ключовими функціональними вимогами також є гнучкість та здатність до пристосування. Хмарні середовища не є константами; вони швидко зазнають трансформацій на рівні апаратного забезпечення, програмного забезпечення та моделі експлуатації користувачами. Отже, метод має володіти здатністю самостійно коригувати власні параметри відповідно до нововиявлених закономірностей, що виникають під час експлуатації. Це передбачає імплементацію циклу самонавчання або адаптації, який дає змогу прогностичним моделям оновлювати ваги, переобчислювати ключові змінні чи реконструювати внутрішню логіку на підставі новопоступлених даних. Без такого механізму неминуче настане стрімка втрата точності, оскільки модель, сформована на базі минулих даних, із часом перестане адекватно репрезентувати мінливу природу навантаження [25].

Крім того, важливо, аби метод мав потенціал масштабування відповідно до комплексності системи. У конфігураціях, де одночасно оперує сотнями чи тисячами сервісів чи контейнерів, кількість часових розрізів даних може бути надзвичайно великою. Метод повинен ефективно обробляти значні обсяги

інформації, не знижуючи своєї продуктивності у сценаріях, де необхідно одночасно проводити передбачення навантаження для множинних елементів. Це суттєва вимога, зважаючи на те, що хмарні платформи нерідко координують величезні пули сервісів, кожен з яких демонструє унікальну динаміку [26].

До нефункціональних вимог належить питання обчислювальної потужності. Навіть якщо моделі глибокого навчання забезпечують вищу точність, вони можуть вимагати надмірних ресурсів. У системах автоматичного регулювання обсягу (автоскейлінгу) час є критичним фактором: рішення про зміну масштабу має бути миттєвим, і будь-яка відстрочка у процесі передбачення може спричинити зниження якості роботи або перевантаження сервісу. Тому метод зобов'язаний забезпечувати рівновагу між достовірністю прогнозу та швидкістю обчислень, застосовуючи складніші моделі лише тоді, коли це обґрунтовано, а в інших випадках використовуючи більш швидкодійні алгоритми [27].

2.3.3. Критерії якості прогнозування

Оцінка достовірності прогнозу є невіддільною складовою розробки та впровадження будь-якої методології. Точність моделі відіграє першочергову роль, адже будь-яке розходження між передбачуваним і фактичним завантаженням може призвести до серйозних наслідків. Якщо прогноз виявиться занадто оптимістичним і недооцінить рівень навантаження, система масштабування не встигне своєчасно збільшити обчислювальні ресурси, що призведе до погіршення якості сервісу, зростання затримок або навіть тимчасової непрацездатності. Якщо ж прогноз буде надмірно обережним і система виділить надлишок ресурсів, це потягне за собою фінансові втрати, особливо у великих масштабах. Тому точність оцінюється за допомогою метрик MAE та RMSE та інших індикаторів, що відображають рівень середнього чи максимального відхилення. Важливо, щоб ці показники брали до уваги як середній рівень

помилки, так і крайні значення - максимальні чи різкі відхилення, які можуть мати найбільший вплив на стійкість [25, 26, 27].

Проте критерій точності не є єдиним. Для хмарних систем критично важливо, щоб прогноз демонстрував стабільність у часі. Модель, яка має невелику середню похибку, може виявитися практично непридатною, якщо прогноз значно "стрибає" між послідовними часовими інтервалами. Такі коливання створюють обставини, коли механізм автоскейлінгу безперервно збільшує або зменшує кількість ресурсів, що спричиняє ефект "мерехтіння" - нестабільне та хаотичне масштабування. Це не лише дестабілізує роботу сервісу, але й підвищує фінансові витрати, оскільки хмарні постачальники часто тарифікують самі операції зміни масштабу. Отже, модель повинна демонструвати не лише точність, але й плавність, послідовність та передбачуваність своєї поведінки [25].

Важливим мірилом є здатність моделі коректно працювати з рідкісними чи нетиповими подіями. Пікові рівні навантаження, хоч і трапляються нечасто, є найбільш значущими для забезпечення стабільності сервісів. Багато моделей демонструють хороші результати у стандартних умовах, але втрачають достовірність у моменти різких змін. Це робить їх малопридатними для застосувань, де пікові показники мають вирішальне значення, наприклад, в інтернет-магазинах під час розпродажів або у стрімінгових платформах під час трансляції важливих подій. Тому модель повинна вміти якісно ідентифікувати аномалії, розрізняти природні різкі сплески від шуму та формувати прогнози, стійкі до таких коливань [28].

Ще один значущий критерій - це адаптивність прогнозу. Якісна модель повинна змінювати свою динаміку залежно від трансформацій у вхідних даних. Якщо структура навантаження еволюціонує, модель мусить оперативно переналаштовувати свої внутрішні параметри. Це означає, що оцінка якості має включати не лише фактичну помилку прогностичних вибірок, але й часову характеристику - швидкість реагування моделі на зміни. Моделі, які адаптуються

повільно, можуть мати прийнятну середню точність, але бути фактично некорисними у реальних експлуатаційних умовах [29].

Окрім того, важливо оцінювати обчислювальну потужність. Хмарні системи не можуть толерувати модель, яка є точною, але надто повільною. Відтак, час, потрібний для формування прогнозу, швидкість обробки даних та спроможність працювати в режимі реального часу є не менш важливими критеріями, ніж сама точність. У багатьох випадках компроміс між точністю і швидкістю є неминучим, і модель мусить забезпечувати оптимальну рівновагу між цими атрибутами [28].

2.4. Концептуальна архітектура запропонованого методу

Створення методології передбачення завантаження, яка забезпечує узгоджену взаємодію між усіма ключовими елементами системи - збором даних, їх первинною обробкою, класифікацією типу завантаження, вибором найбільш відповідної моделі - є обов'язковою умовою. На відміну від традиційних підходів, де використовується єдина модель незалежно від властивостей даних, запропонована концепція передбачає можливість динамічної зміни між різними моделями.

Розробка такої архітектури є відповіддю на фундаментальні обмеження, які присутні у сучасних методах прогнозування. Як було показано вище, жодна існуюча модель не здатна однаково ефективно функціонувати з усіма різновидами навантаження. Статистичні підходи чудово справляються з циклічними та стабільними рядами, але втрачають достовірність на хаотичних даних. Машинне навчання ефективно моделює нелінійні залежності, проте іноді погано враховує сезонність. Моделі глибокого навчання здатні ідентифікувати складні патерни, проте вони обтяжливі з точки зору обчислень і схильні до надмірного тренування. У хмарному середовищі навантаження часто буває

змішаним, тобто окремі його компоненти мають різні структурні характеристики.

Запропонована архітектура усуває цей недолік за допомогою впровадження мультимодельного принципу, де застосування різноманітних алгоритмів передбачення залежить від конкретної структури часового ряду. Такий підхід поєднує в собі високу точність, здатність до пристосування та універсальність. Він не лише сприяє поліпшенню якості передбачень, але й суттєво підвищує стійкість системи управління ресурсами. Архітектура також охоплює механізм зворотного зв’язку, який контролює ефективність задіяних моделей, здійснює оновлення їхніх параметрів та коригує вибір моделей на основі нової інформації.

2.4.1. Загальна структурна моделей методу

Загальна структурна моделей відображає логічний шлях, який проходять дані - від моменту їх надходження з системи моніторингу до формування остаточного прогнозу та його передання у систему автоматичного масштабування. Цей шлях складається з кількох послідовних та взаємопов’язаних компонентів.

У першому блоці система отримує телеметричні дані, що надходять у режимі реального часу. Ці дані мають різні частоти оновлення, можуть бути агрегованими або сирими, а також містити шуми чи пропуски. Важливо зазначити, що сучасні хмарні сервіси генерують величезні обсяги даних, тому перший етап архітектури має забезпечувати ефективне збирання даних з мінімальним навантаженням на систему. Другий блок - попередня обробка яка усуває недоліки сирих даних: нормалізує, агрегує, фільтрує та перетворює їх у формат, придатний для подальшого прогнозування. На цьому етапі формується структурована множина ознак, що згодом дозволяє моделі краще інтерпретувати інформацію.

Третім ключовим етапом є визначення типу навантаження за допомогою ШІ модулю. Саме цей блок надає системі можливість адаптації: він аналізує структуру даних, виявляє присутність сезонності, рівень шуму, пікові чи стохастичні флуктуації, та класифікує ряд. Результат цього етапу впливає на весь подальший алгоритм.

Далі працює ядро системи - модуль вибору та застосування моделі. На цьому етапі відбувається або запуск однієї моделі, або паралельне виконання кількох моделей із наступним комбінуванням результатів. Модель обирається залежно від типу навантаження, що дозволяє уникнути універсальних, але неточних прогнозів.

Останнім елементом є модуль самонавчання. Він аналізує похибки прогнозування, порівнює результати різних моделей, оновлює внутрішні параметри і забезпечує довгострокову адаптивність системи.

2.4.2. Модуль збору та попередньої обробки даних

Збір даних є першим і фундаментальним етапом роботи методу. Хмарна інфраструктура генерує дані з високою частотою і великою кількістю параметрів. У цьому середовищі важливо не просто зібрати дані, а зібрати їх правильно. Дані можуть надходити із систем моніторингу CPU, пам’яті, мережі, дисків, черг запитів, логів тощо. Кожне джерело має свої особливості у форматі, часі збору, процедурі агрегації. Нерідко дані приходять із затримками або з пропусками, тому модуль збору повинен узгодити часові мітки, синхронізувати частоти вибірок та забезпечити цілісність даних[28].

Після збору даних необхідно провести попередню обробку. Вона включає очищення від шуму, заміну або інтерполяцію пропущених значень, фільтрацію випадкових сплесків, що не несуть цінності. На цьому етапі виконується нормалізація значень, масштабування, а також формування похідних ознак. Наприклад, можна додати числові ознаки, що характеризують час доби, день тижня, тривалість останнього тренду, похідні від зміни навантаження.

Складність модуля полягає в тому, що він повинен працювати в режимі реального часу. Попередня обробка має бути достатньо швидкою, щоб не створювати затримок, але і достатньо повною, щоб забезпечити якісні дані для прогнозування.

2.4.3. Модуль аналізу типу навантаження

Модуль аналізу типу навантаження визначає структуру часового ряду, що є вхідними даними. Він працює як механізм класифікації на основі наявних моделей ШІ-класифікаторів (RandomForestClassifier), який автоматично встановлює, які характеристики переважають у поточний момент.

Модуль включає кілька рівнів аналізу. На першому рівні проводиться базова статистична перевірка: оцінюється варіативність, автокореляція, наявність тренду, сезонності, спектральні особливості ряду. На другому рівні застосовуються методи машинного навчання, які працюють із набором статистичних ознак, що були сформовані на попередньому етапі.

Якщо часовий ряд демонструє регулярні цикли, система класифікує його як сезонний. Якщо відсутні цикли, але присутні слабкі кореляції - як трендовий. Якщо основною ознакою є раптові сплески - як піковий. Якщо ряд має високий рівень шуму - як стохастичний. Найскладніший варіант - змішані ряди, де присутні кілька типів патернів. У таких випадках система може визначити домінантний тип або позначити ряд як змішаний, що вимагає застосування ансамблевих моделей[30].

Результат цього модуля визначає траєкторію всього подальшого прогнозування, тому точність класифікації є критичною.

2.4.4. Модуль прогнозування та адаптивного вибору моделі

Цей модуль є ядром усього методу. Він відповідає за застосування моделей прогнозування та адаптацію вибору залежно від структури навантаження.

Модуль містить бібліотеку моделей, які різняться між собою за типом математичного апарату та складністю:

- статистичні моделі для сезонних і стабільних рядів;
- алгоритми машинного навчання для рядів з нелінійними залежностями;
- глибинні нейронні мережі для складних, хаотичних або змішаних сценаріїв;
- ансамблеві методи, які поєднують переваги кількох моделей.

Після отримання класифікації від попереднього модуля система автоматично визначає, яку модель слід застосувати. Наприклад, для сезонних рядів краще підходять SARIMA або Prophet, для хаотичних - XGBoost, для складних залежностей - LSTM чи трансформери. У випадках змішаного ряду система запускає кілька моделей і комбінує їх прогноз через ансамблевий підхід, наприклад через зважену середню або метамодель[15,29].

Цей модуль також враховує попередню ефективність моделей. Якщо певний алгоритм постійно демонструє низьку точність у поточних умовах, його вага автоматично зменшується. Таким чином, відбувається динамічна адаптація, яка дозволяє підтримувати точність прогнозування навіть у змінних умовах.

2.5. Математичне та алгоритмічне обґрунтування методу

Розробка мультимодельного адаптивного методу прогнозування навантаження вимагає наявності строгого математичного апарату, який би пояснював як сам процес прогнозування, так і механізми перемикання між моделями. На відміну від класичних алгоритмів, де структура моделі є статичною і не змінюється під час роботи, запропонований метод передбачає динамічну адаптацію. Це означає, що математична основа повинна враховувати не лише властивості окремих моделей, але й формалізувати взаємодію між ними, механізм вибору, об’єднання прогнозів і процедури самооновлення параметрів.

Також важливо врахувати той факт, що хмарні навантаження є нестійкими, нерівномірними та структурно змінними. Тому математична постановка повинна включати можливість роботи з часовими рядами, які змінюють свій клас у часі, тобто функція прогнозування має адаптуватися до нової структури даних. Це потребує введення механізмів класифікації, оцінювання похибок і оновлення параметрів.

2.5.1. Математична постановка задачі

Навантаження у хмарній системі можна уявити як послідовність значень, які ми спостерігаємо у часі. Наприклад: кількість запитів за секунду, завантаження CPU, використання пам’яті, кількість активних користувачів.

Тобто, у найпростішому випадку ми працюємо з послідовністю:

$$X = \{x_1, x_2, x_3, \ldots, x_t\}, \tag{2.3}$$

де x_t – значення метрики у момент часу t.

Ця формула означає, що ми маємо послідовність вимірювань, яка відображає поведінку навантаження у часі[16]. Прогнозування зводиться до оцінювання наступних значень цього ряду. Функція прогнозування задається як:

$$x_{T+k} = F(X_t), \tag{2.4}$$

де x_{T+k} – прогнозоване значення, а F – модель, яка обчислює прогноз на основі історії. Цей означає, що майбутнє значення визначається функцією від попередніх спостережень. Однак в умовах хмарних систем навантаження часто є багатовимірним, тому ряд представлено у вигляді векторів:

$$\overrightarrow{x_t} = (\overrightarrow{x_t^1}, \overrightarrow{x_t^2}, \ldots, x_t^m), \tag{2.5}$$

де кожний компонент $\overrightarrow{x_t^i}$ описує окрему метрику (CPU, пам’ять, трафік тощо). Така формалізація дозволяє моделям враховувати взаємозв’язки між різними параметрами. У хмарних середовищах структура ряду може змінюватися, тому вводимо функцію класифікації:

$$C_t = G(X_t), \tag{2.6}$$

де C_t - визначений тип навантаження у момент часу t, а X_t - вектор характеристик навантаження у момент часу t. Ця формула формалізує важливу ідею: перед тим як прогнозувати, система повинна зрозуміти, з яким саме типом даних вона працює. Тоді прогнозування набуває умовного вигляду:

$$x_{T+k} = F_{Ct}(X_t), \tag{2.7}$$

де x_{T+k} - прогнозоване значення навантаження через k кроків у майбутнє, а F_{Ct} - модель прогнозування, яка обирається залежно від класу навантаження.

Така адаптивність необхідна, оскільки різні типи рядів потребують різних методів прогнозування.

2.5.2 Формалізація моделей у мультимодельному підході

У методі використовується кілька типів моделей, кожна з яких описана компактними формулами, але має різні властивості та сфери застосування. Тому їх варто поділити на три основні групи - статистичні, моделі машинного навчання та глибинні нейронні мережі.

Говорячи про статистичні моделі, можна описати її роботу за допомогою формули:

$$x_{T+1} = a_1 x_T + a_2 x_{T-1} + \cdots + a_n x_{T-n}, \tag{2.8}$$

де x_{T+1} - прогнозоване значення часового ряду на один крок вперед, $x_T, x_{T-1}, \ldots, x_{T-n}$ - попередні значення часового ряду, які використовуються моделлю для аналізу динаміки, а a_1, a_2, a_n - коефіцієнти моделі. Ця формула є базовим виглядом лінійного прогнозу. Вона відображає, що прогноз залежить від попередніх значень, кожне з яких має свою вагу. Коефіцієнти a_i визначають вплив кожного минулого значення, при цьому модель припускає певну лінійність у поведінці ряду і добре підходить для стабільних або сезонних рядів.

$$x_{T+1} = ML(X_t), \tag{2.9}$$

де x_{T+k} - прогнозоване значення навантаження через k кроків у майбутнє, ML - будь-який алгоритм машинного навчання, а X_t - вектор характеристик

навантаження у момент часу t. Це узагальнена формула для групи моделей машинного навчання, яка вказує, що прогноз формується через алгоритм ML, який вчиться на історичних даних. Модель отримує набір ознак (як вихідних, так і похідних), на основі навчання вона встановлює зв'язки між параметрами, і формує прогноз на наступний момент часу. Такі моделі краще працюють із нелінійними залежностями[19].

У контексті глибинних нейронних мереж можна скористатись формулою

$$x_{T+1} = NN(X_t), \tag{2.10}$$

де x_{T+k} - прогнозоване значення навантаження через k кроків у майбутнє, NN - нейронна мережа а X_t - вектор характеристик навантаження у момент часу t. Це запис нейронної мережі як чорної скриньки, що приймає дані та повертає прогноз. Мережа може враховувати довгі залежності, здатна виявляти складні нелінійні патерни і при цьому оптимально працює зі стохастичними та хаотичними рядами.

2.5.3. Алгоритм адаптивного вибору моделі

Оскільки різні моделі ефективні у різних сценаріях, метод включає формальний механізм вибору. Тобто є три етапи як метод визначає, яку модель вибрати - визначення типу навантаження, оцінювання якості моделі та вибір оптимальної моделі.

Етап визначення типу навантаження можна описати формулою, яку ми використали вище:

$$C_t = G(X_t), \tag{2.11}$$

де C_t - визначений тип навантаження у момент часу t, а X_t - вектор характеристик навантаження у момент часу t. Ця функція визначає структуру сигналу. Вона потрібна для того, щоб обрати модель, яка найкраще відповідає поточному типу ряду. Тобто система аналізує ряд для визначення чи є повторювані цикли, чи є

багато шуму, чи є різкі стирибки, чи змінюється поведінка протягом дня, тощо, і таким чином визначається клас C_t[18].

Оцінювання якості моделі відбувається шляхом зберігання її середньої похибки.

$$E_j = \text{середня помилка моделі } j, \tag{2.12}$$

де j – вибрана модель. Ця величина використовується як показник довіри до моделі, тобто менша похибка означає вищу якість.

Вибір оптимальної моделі відбувається за допомогою формули:

$$M_t = \operatorname{argmin}_j E_j, \tag{2.13}$$

де $\operatorname{argmin}_j E_j$ - оператор, який повертає індекс моделі з найменшою помилкою.

Тобто обирається модель, яка показала найкращі результати на подібних даних у минулому. Це забезпечує адаптивність та мінімізацію ризиків неправильного прогнозу.

3 РОЗРОБКА АДАПТИВНОГО МЕТОДУ ПРОГНОЗУВАННЯ НАВАНТАЖЕННЯ У ХМАРНИХ СЕРВІСАХ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ ЙОГО РЕЗУЛЬТАТИВНОСТІ

Створення методики передбачення навантажень у хмарних сервісах вимагає формування цілісного, внутрішньо узгодженого алгоритмічного рішення, яке відображає увесь комплекс процесів, описаних вище. Втілення такого методу є ключовим елементом дослідження, оскільки дозволяє визначити, яким чином концептуальні положення, сформовані під час аналізу сучасних хмарних технологій, принципів використання алгоритмів штучного інтелекту та характеристик динамічних часових рядів навантаження, втілюються у практичній моделі, здатній функціонувати в умовах реального або близького до реального інформаційного середовища.

Структура навантаження у хмарних системах є складною, багатокомпонентною та високоволатильною, бо природа таких навантажень визначається особливостями архітектури розподілених систем, поведінкою користувачів, інтенсивністю запитів, внутрішніми процесами балансування, масштабування та міграції контейнерів, а також значною чутливістю до зовнішніх чинників[32,33]. Відповідно, метод передбачення повинен враховувати не лише амплітудні характеристики ряду, але й тип його поведінки, стабільність, наявність сезонності, випадковості або пікової активності. Реалізація методу повинна об’єднувати ці елементи в єдину структуру, здатну оцінювати стан системи, передбачати майбутні значення навантаження та реагувати на зміну його динаміки. Багатокомпонентний характер хмарних навантажень обумовлює необхідність використання комплексного підходу, який поєднує статистичні, машинні та глибинні моделі, а реалізація методу передбачає створення архітектури, у межах якої ці моделі працюють як взаємодоповнювальні інструменти, що розв’язують задачі передбачення з різним

рівнем складності, різною чутливістю до шуму та різною здатністю до узагальнення. Статистичні моделі є найбільш ефективними для сезонних або відносно стабільних рядів[33]. Моделі машинного навчання здатні аналізувати багатовимірні закономірності, що виникають унаслідок взаємодії різних метрик навантаження. Глибинні моделі, у свою чергу, забезпечують можливість виявлення складних нелінійних структур, характерних для хаотичних або стохастичних часових рядів[33].

Розробка такого методу потребує побудови системи попередньої обробки даних, а оскільки телеметрія в хмарних сервісах нерідко має неповноцінну, асинхронну та зашумлену структуру, реалізація повинна включати масштабовані компоненти очищення даних, нормалізації, відновлення пропусків, дискретизації та формування ознак. Дані, що надходять із систем моніторингу, часто мають нерівномірний крок знімання, містять випадкові стрибки, пропуски або аномальні значення, які є артефактами роботи системи або результатом затримок у доставці телеметрії. І саме тому реалізаційна частина повинна забезпечити узгодження часових міток, стабілізацію трендів та формування узагальнених статистичних і спектральних характеристик, які є основою для подальшого визначення типу навантаження[34].

Невід’ємною складовою розробки методу є механізм класифікації, який визначає, до якого типу належить поточний часовий ряд. Класифікація виступає центральним елементом адаптивності: залежно від виявленої структури навантаження метод визначає, які прогнозні моделі доцільно використовувати, які параметри обробки є домінантними і яка стратегія передбачення буде найточнішою. У реалізації класифікатора враховуються властивості, описані раніше: автокореляційні характеристики, дисперсійна структура, локальні коливання, періодичність, ступінь піковості та амплітудні зміни. Усі ці параметри формується як сукупність ознак, яка забезпечує достатню інформативність для визначення типу ряду з мінімально можливою похибкою. Реалізація механізму класифікації передбачає формування набору ознак у

реальному часі або під час попереднього аналізу даних, що моделює роботу системи в умовах хмарного середовища. Подальший етап реалізації полягає у створенні бібліотеки прогнозних моделей, де кожна модель виконує власну функцію і застосовується для певного підтипу поведінки навантаження[32,33,34].

Статистичні моделі, такі як авторегресивні підходи, забезпечують інтерпретованість і велику точність там, де структура ряду є відносно стабільною. Моделі машинного навчання дозволяють аналізувати складні взаємозв'язки між метриками навантаження, які не можуть бути описані простими аналітичними співвідношеннями. Глибинні моделі, зокрема рекурентні нейронні мережі та моделі на основі механізму уваги, дозволяють виявляти віддалені залежності у часових рядах, відрізняються стабільністю у ситуаціях із високою нерегулярністю і гнучкістю у роботі з хаотичними патернами. Реалізація цього компоненту вимагає побудови уніфікованого інтерфейсу передбачення, який забезпечує однакову схему виклику для всіх моделей та дозволяє організувати їх паралельне виконання, що є критично важливим для багатомодельної структури[35].

Завершальним елементом реалізації є формування адаптивної системи вибору моделі. На практичному рівні цей компонент відповідає за оцінювання прогнозів, отриманих від різних моделей, їх аналіз за показниками точності, стабільності та розподілу похибок, а також за вибір моделі, яка є найбільш оптимальною у відповідних умовах. Важливо забезпечити можливість регулярного оновлення статистичних характеристик моделей, щоб мати можливість оцінювати моделі, а точніше їх ефективність, у динаміці та враховувати зміни поведінки навантаження. Реалізація цього елементу формує основу адаптивності, оскільки система здатна автоматично перелаштовувати ваги моделей, змінювати конфігураціюта коригувати процедури обробки залежно від нових даних. Для достовірної перевірки необхідно використовувати декілька джерел даних, які відображають широкий спектр можливих сценаріїв

поведінки навантаження. Дані можуть включати як реальні часові ряди з відкритих датасетів, так і синтетично згенеровані сигнали, що імітують поведінку систем із сезонними, піковими або стохастичними характеристиками. Результати передбачення порівнюються за стандартними метриками точності, такими як середня абсолютна помилка, середньоквадратична помилка та відносна похибка. Додатково оцінюється поведінка методу в умовах різких змін у динаміці навантаження, що дозволяє виявляти його стійкість до аномалій та його здатність коректно реагувати на нестабільність ряду[34,35].

Таким чином, реалізація методу включає створення архітектури збору та обробки даних, налаштування класифікатора, реалізацію набору моделей передбачення та формування адаптивного механізму вибору. Комплексність цього процесу забезпечує можливість моделювання роботи реальної хмарної системи та оцінки того, наскільки інтелектуальні методи передбачення здатні підвищити її ефективність, стабільність та передбачуваність.

3.1. Загальні принципи і підходи до реалізації методу

Створення гнучкого методу прогнозування навантаження у хмарних сервісах вимагає формування такої реалізаційної схеми, яка б здатна була достеменно відтворити логіку математичного зразка та забезпечити його стабільну роботу в умовах мінливої інформаційної течії. На противагу концептуальному викладу, що окреслює абстрактні механізми роботи системи, практичне втілення вимагає суворого структурування робочих процесів, визначення чітких взаємодійних точок (інтерфейсів) між складовими частинами, а також урахування технічних та алгоритмічних обмежень, притаманних фактичним або наближеним до реальних даним. Зважаючи на особливості хмарних теренів, архітектурне рішення має бути сфокусоване не лише на отриманні передбачення, а й на спроможності багаторазово та безперервно відтворювати повний ланцюжок обробки даних, включно з їхньою структурою,

відновленням, перетворенням та спрямуванням до наступного вузла системи. У цьому контексті, впровадження методології постає не просто як тлумачення алгоритму, а як інженерна формалізація цілісної обчислювальної послідовності, яка бере до уваги вимоги до швидкодії, узгодженості інформації, надійності та відтворюваності результатів[35]. Схема роботи методу зображена на рисунку 3.1.

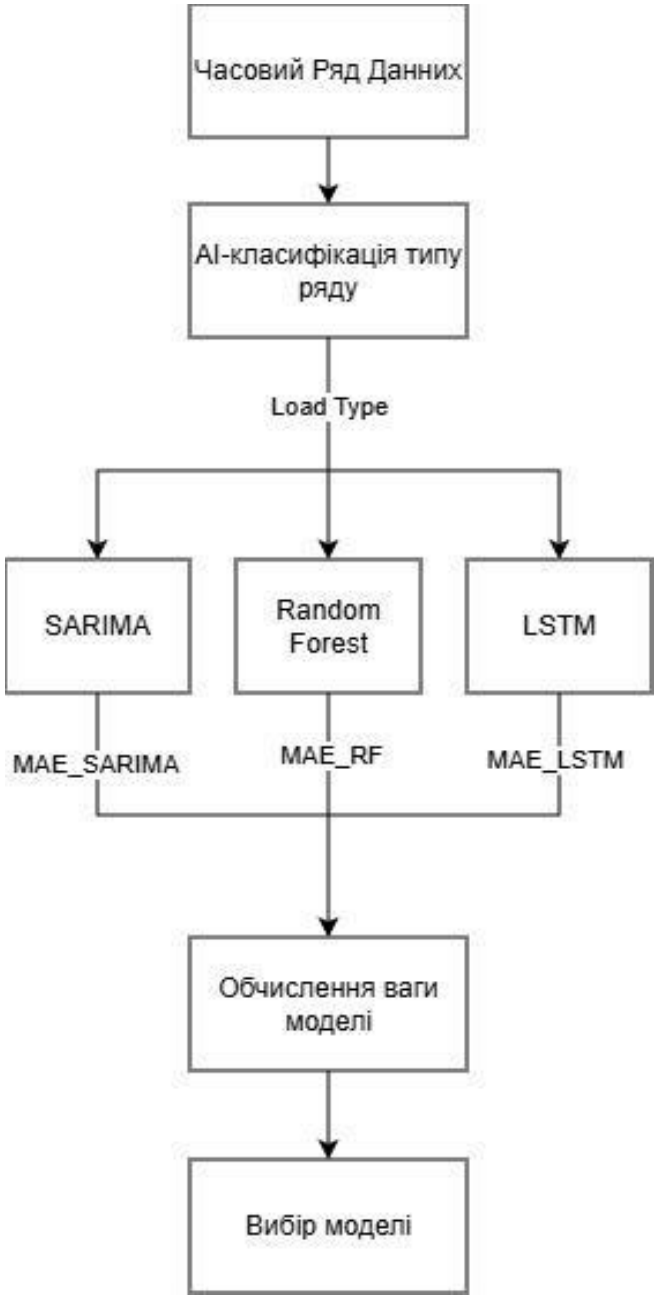


Рис. 3.1 Схема роботи адаптивного методу

Впровадження системи мусить гарантувати точне розділення обов’язків між окремими програмними блоками, що дає змогу інкапсулювати функції кожного елемента, убезпечуючи від надлишкових операцій та забезпечуючи автономність складових. Кожен модуль - від збору телеметричних даних аж до гнучкого комбінування прогнозів - зобов'язаний мати чітко визначені вхідні та вихідні дані, формальну характеристику, виконувати один чи кілька строго окреслених етапів алгоритму та бути структурно автономним від решти частин системи. Це не лише спрощує етапи розробки та перевірки, але й сприяє масштабуванню та заміні чи доповненню окремих елементів без необхідності переробляти усю архітектуру.

Надзвичайно важливим є те, що реалізація повинна адекватно реагувати на динамічний характер вхідної інформації. На відміну від незмінних задач, де вхідні дані закріплені, у хмарних комплексах навантаження змінюється безперервно, а його структура може трансформуватися кілька разів за короткий період. Це означає, що архітектура повинна бути спроможна обробляти дані послідовними, незалежними порціями, здатними функціонувати в умовах часових обмежень. Важливим аспектом також є забезпечення захищеності від нерівномірності вхідних даних, випадків їх втрати, шуму та тимчасових відхилень, оскільки ці явища є типовими для збору телеметрії у хмарних комплексах[14,27,31].

Однією з ключових ознак реалізації є потреба у підтримці алгоритмічної гнучкості. Це передбачає, що архітектура має надавати змогу моделі трансформувати свою поведінку залежно від ідентифікованого типу завантаження. Необхідно передбачити механізм регулярного оновлення статистичних параметрів, зменшення накопичених помилок та коригування вагових коефіцієнтів моделей. Усі ці дії мають бути інтегровані як частина внутрішнього робочого циклу системи, який повторюється кожного разу, коли надходять новенькі дані.

Запропонована методологія має забезпечувати достовірність передбачень для різноманітних типів навантаження, відтак реалізація повинна підтримувати одночасну роботу багатьох алгоритмічних підходів. Це вимагає зведення такої архітектури, де традиційні статистичні, методи машинного навчання та глибокі моделі є рівноправними складовими системи, що здійснюють прогнозування на одному й тому ж наборі попередньо оброблених даних. Важливо усвідомлювати, що кожен клас моделей має власні ліміти: статистичні моделі вимагають сталості (стаціонарності), ML-моделі потребуюють формування релевантних ознак, а глибокі моделі суттєво вимагають обчислювальних ресурсів.

Тому архітектура впровадження мусить створювати умови, за яких кожна з моделей отримає оптимальний набір вхідних даних і зможе працювати з максимальною віддачею. Окрім того, реалізація мусить брати до уваги можливість майбутньої інтеграції в системи автоматизованого масштабування. Навіть якщо у рамках поточного дослідження інтеграція представлена лише на рівні імітації, важливо забезпечити теоретичну та структурну готовність методу до взаємодії з елементами управління (оркестрації). Це означає, що архітектура мусить мати чітко окреслений спосіб передачі прогнозних значень, здатність розширювати формат вихідної інформації та узгоджувати алгоритми прогнозування з вимогами систем моніторингу[25,35].

Отже, загальні принципи реалізації обіймають структурне розбиття на модулі, роботу з часовими даними високої мінливості, забезпечення гнучкості, підтримку різнорідних моделей та узгодження алгоритмічних вимог з умовами подальшої експлуатації методу. У межах цієї підрозділу детально розглядаються вимоги до архітектури, що гарантують стабільність, здатність до розширення та ефективність втілення запропонованого адаптивного методу.

3.1.1. Архітектурні вимоги до реалізації адаптивного методу прогнозування

Архітектурні вимоги до впровадження адаптивного методу передбачення навантаження формуються на перетині математичної моделі, описаної раніше, та специфічних особливостей роботи хмарних обчислювальних оточень. Реалізація повинна забезпечити відтворення усіх ключових частин методології, зберігаючи при цьому структурну гнучкість, здатність до розширення та спроможність протистояти непередбачуваним змінам у даних, що надходять із систем спостереження. Хмари становлять собою комплексні утворення, де навантаження є багатофакторним, нерегулярним та схильним до стрімких трансформацій, тому архітектура алгоритму повинна бути розрахована на роботу в умовах значної динаміки, неповноти даних та нестабільності сигналу.

Впровадження методу має базуватися на модульній парадигмі, згідно з якою кожен блок виконує власну функцію в обробці даних та взаємодіє з іншими блоками винятково через чітко окреслені зовнішні інтерфейси. Модульність дає можливість розміщувати складні обчислювальні послідовності у відокремлених секціях та усувати зв’язки, які перешкоджають розширенню або заміні частин системи. Сучасні системи прогнозування нерідко страждають від надмірної жорсткості зв’язку між модулями, що ускладнює їх розвиток; натомість архітектура гнучкої методології повинна створювати простір для оновлення моделей, додання нових алгоритмів чи змін у логіці обробки без порушення цілісності усього програмного комплексу.

Однією з фундаментальних вимог є спроможність системи оперувати потоковими даними. Телеметрія хмарних систем формується у режимі справжнього часу, проте з характеристиками, які суттєво відрізняються від класичних потоків інформації. Інтервали між зняттям показників можуть бути нерівномірними, дані можуть надходити із запізненням, і частина відомостей може бути тимчасово недоступною. Потік даних може міняти свою структуру залежно від активності користувачів, розподілу трафіку чи внутрішніх процесів

збалансування навантаження. Архітектура реалізації мусить не просто знайти спосіб для їхньої обробки, а впровадити механізм пристосування до цієї мінливості. Неможливо коректно будувати прогноз, якщо часові мітки не узгоджені, тому система повинна мати інтегровані засоби передискретизації, згладжування та коригування даних. Це є однією з найскладніших частин архітектури, оскільки саме від неї залежить, чи отримають моделі передбачення коректний та стабільний часовий ряд[36].

Узагальнюючи, узгодженість даних є наріжним каменем для всієї прогностичної системи.

Значно складнішим завданням є робота з провалами (пропусками). У моніторингу хмарних систем пропуски трапляються постійно; вони можуть виникати через мережеві затримки, перерозподіл навантажень, зміни налаштувань чи нетривалі збої. Реалізація повинна включати механізм виявлення пропусків та диференційовані методи їх заповнення: невеликі проміжки можна заповнювати інтерполяцією, тоді як довгі розриви потребують побудови коригувальних моделей або обробки на рівні окремих фрагментів ряду. Витонченість цього кроку визначає якість передбачення, адже пропуски несуть ризик спотворення структури сигналу і, як наслідок, зниження точності моделей[36,38].

Окремим архітектурним завданням є забезпечення збереження та оновлення внутрішнього стану системи. Метод передбачає накопичення та аналіз статистики похибок моделей, оцінку їхньої стабільності, обчислення вагових коефіцієнтів для ансамблю. Архітектура повинна передбачати сховище цих метрик, засоби їхнього періодичного оновлення та механізм доступу до них у реальному часі. Це дає можливість адаптувати поведінку моделі не лише на основі поточних даних, а й з урахуванням історичних характеристик, що значно підвищує її ефективність у довготривалій перспективі. Важливим є і те, що зберігання похибок дозволяє системі «пам’ятати» поведінку різних моделей у

подібних умовах і робити більш обґрунтований вибір у наступних ітераціях прогнозування.

Таким чином, архітектурні вимоги формують комплексну систему принципів, що забезпечують коректну реалізацію, стабільність роботи, адаптивність, масштабованість та прозорість процесів прогнозування. Реалізація має бути побудована таким чином, щоб вона могла відтворювати логіку адаптивного методу у повному обсязі, водночас залишаючись достатньо гнучкою для подальшого розвитку та інтеграції у реальні хмарні середовища.

3.1.2. Логічна структура програмних модулів

Логічна будова програмних блоків у рамках адаптивного методу передбачення навантаження є основою його втілення, адже саме вона керує послідовністю опрацювання відомостей, взаємодією елементів у системі, порядком пересилання даних між стадіями та способом формування кінцевого прогнозу. На противагу традиційним засобам прогнозування, що працюють у рамках єдиної обчислювальної процедури, описуваний метод характеризується багатоетапною ієрархією та сегментацією логіки на окремі модулі, кожен з яких функціонує як самодостатній вузол у загальній схемі обчислень. Ядром же такої структури служить поділ системи на функціональні рівні, де кожен рівень відповідає за окремий аспект аналізу даних. Перший рівень об’єднує механізми роботи з телеметричними даними, що надходять із хмарного оточення або застосовуються у моделюванні.

Саме тут визначається, у якому вигляді інформація дістанеться наступних ланок алгоритму. Потрібно, аби логіка модуля збору відомостей забезпечувала не лише отримання сирих даних, а й їх первинну впорядкованість - узгодження часових міток, перевірку правильності, валідацію формату та підготовку до подальших перетворень. Дані рідко надходять у досконалій формі, тому система приймання повинна мати здатність опрацьовувати не лише зведені показники, а

й окремі фрагменти, що виникають через накладання потоків, збій тактовості збору чи появу локальних відхилень.

Після первинного упорядкування відомості переходять до модуля попереднього опрацювання, який є одним із найважливіших складників системи. Тут відбувається вирівнювання даних, поповнення прогалин, формування рівномірної структури ряду та створення додаткових ознак, необхідних для функціонування класифікаційних і прогнозних моделей.

Цей блок доцільно сприймати як інтелектуальний фільтр, що готує дані для більш високорівневих етапів, мінімізуючи вплив шумів та структурних дефектів. У складних хмарних середовищах саме цей компонент виконує роль своєрідного буфера між необробленими телеметричними даними та математичними моделями передбачення. Одним із ключових етапів логічного ланцюга є модуль визначення типу навантаження. Його завдання полягає в тому, щоб з'ясувати властивості часової послідовності: чи має вона ознаки сезонності, є стохастичною, піковою чи комбінованою. У цьому модулі аналізуються статистичні характеристики ряду – автокореляція, рівень сезонної енергії, амплітудні коливання, локальні тенденції та ступінь нерівномірності. Модуль слугує логічним перемикачем, що дає змогу гнучко змінювати подальшу логіку роботи алгоритму. Якщо класифікаційний рівень помилиться у визначенні типу навантаження, увесь наступний ланцюг прогнозування ризикує втратити точність, тому критично важливо, щоб він працював надійно навіть за умов суттєвого шуму[23,37].

Наступним структурним елементом є блок мультимодельного прогнозування. На цій стадії стартують статистичні, машинно-навчальні та глибокі моделі, кожна з яких генерує власний прогноз для найближчого періоду чи заданого горизонту. Важливо, що моделі опрацьовуються незалежно, використовуючи ідентичний набір ознак та однакову, попередньо оброблену вибірку. Логіка цього модуля передбачає, що результати роботи моделей на цьому етапі не змішуються - вони лише збираються, утворюючи сукупність

альтернативних прогнозів. Усі моделі є рівноправними учасниками процесу: жодна з них не має "заводського" пріоритету, оскільки адаптивність досягається наступним етапом вибору, а не окремою моделлю.

Механізм вибору прогнозу на основі адаптації є важливою частиною логічної структури методу. У цьому модулі здійснюється оцінка якості прогнозів, сформованих кожною моделлю, визначається їхня минула результативність, стабільність та схильність до надмірної чи недостатньої відповідності. Модуль оперує зібраними даними про похибки кожної моделі, оцінює їхню поведінку на різних типах рядів, розраховує вагові коефіцієнти й створює інтегральний прогноз – ансамбль, який об’єднує індивідуальні оцінки в єдине значення. Логічна побудова цього модуля є заплутаною, оскільки він зобов'язаний перелаштовувати свою роботу при кожному оновленні статистики моделей, забезпечуючи безперервне пристосування системи до мінливих умов у хмарному оточенні[21].

Підсумовуючи логічну структуру, слід зазначити, що їй притаманні послідовність, відокремленість компонентів та внутрішня залежність між модулями. Кожен блок залежить від виходів попереднього, проте його функціонування не впливає на внутрішню логіку інших частин. Такий підхід створює ефект системної цілісності: від початку до кінця прогноз формується як наслідок проходження даних через комплексну, але структуровано впорядковану архітектуру, що відповідає вимогам платформ хмарних обчислень.

3.1.3. Середовище реалізації та технічні обмеження

Операційне середовище, обране для реалізації адаптивного методу передбачення навантаження у хмарних сервісах, формує технічну базу, на якій здійснюється робота усіх програмних блоків, викладених у попередніх частинах. Від вибору цього середовища залежить не лише швидкість обчислень, але й здатність алгоритмів опрацьовувати значні масиви динамічних даних, гарантувати узгодженість часових рядів, підтримувати складні процедури моделювання та

ефективно функціонувати з багатокомпонентною структурою системи. Втілення неможливо розглядати окремо від технологічного контексту: будь-яка складова методу - від попереднього опрацювання до адаптивного ансамблювання – вимагає стійкості ресурсів, можливості нарощування потужностей та високої надійності середовища, де він виконується.

У більшості випадків операційне середовище для прогнозних систем будується на базі платформ, що підтримують наукові та аналітичні інструменти. Найчастіше це платформи, де використовується мова програмування Python або її розширення, що надають доступ до репозиторіїв статистичного аналізу, машинного навчання та глибокого моделювання. Важливо, що обрана платформа повинна забезпечувати не лише змогу оперувати різними інструментами, але й гарантувати їхню взаємосумісність. Наприклад, при застосуванні глибоких моделей можуть залучатися TensorFlow чи PyTorch, тоді як статистичні моделі спираються на бібліотеки іншого рівня, як-от statsmodels. Співпраця цих елементів має бути сталою та передбачуваною[38].

З огляду на це, середовище виконання не може бути простим набором бібліотек. Воно має виконувати функції координації робочих потоків, опрацювання потокових даних, підтримки розподілених обчислень та паралельної роботи моделей. Робота з потоковими даними є одним із найскладніших аспектів втілення. Дані, що надходять у режимі реального часу, можуть бути нерівномірними, неповноцінними чи суперечливими. Це вимагає наявності інструментів для контролю цілісності, структурування та буферизації відомостей. У реальних системах подібні завдання вирішуються спеціалізованими сервісами: Apache Kafka, AWS Kinesis, Google Pub/Sub чи іншими аналогами. У моделюванні ці функції можуть бути частково відтворені у локальному оточенні, проте архітектурно реалізація має відповідати вимогам, які ці сервіси висувають до опрацювання даних.

Технічно принципово, аби операційне середовище давало змогу працювати з великими обсягами оперативної пам'яті. Часові ряди, що відображають

навантаження у хмарних системах, можуть містити десятки тисяч, а інколи й мільйони виміряних точок. Попереднє опрацювання, формування ознак та моделювання можуть вимагати значних обсягів оперативної пам'яті. Статистичні моделі можуть бути відносно "легкими", але моделі машинного навчання, зокрема бустингові, потребують значно більших ресурсів. Глибокі моделі, особливо ті, що працюють із тривалими часовими залежностями, застосовуючи рекурентні чи увагові архітектури, можуть потребувати апаратного підсилення, на кшталт графічних процесорів чи тензорних блоків. Таким чином, операційне середовище повинно мати можливість до масштабування згідно з обчислювальними потребами кожного компонента.

Середовище має забезпечувати сумісність різних режимів обчислень. Частина етапів виконується послідовно, проте моделі прогнозування здатні працювати паралельно. Це означає, що технологічна база мусить підтримувати багатопотоковість або розподілені обчислення. Нестача такої підтримки призведе до подовження часу виконання моделі, а отже, до можливих затримок у передбаченні, що суперечить принципам роботи у хмарних оточеннях, де рішення щодо розширення потужностей мають ухвалюватися негайно[37].

Питання технічних обмежень відіграє особливу роль. Найперше обмеження - це час. У хмарних системах рішення щодо розподілу ресурсів повинні прийматися практично миттєво. Якщо реалізація прогнозного методу потребує часу, який перевищує допустиме вікно реакції, метод втрачає практичне значення. Тому архітектура реалізації повинна забезпечувати оперативність обчислень. Інше обмеження - це обсяг доступних даних. У реальних сценаріях можуть бути відсутні довгі історичні ряди. Метод повинен коректно працювати навіть у ситуаціях, коли історична інформація є обмеженою або нерівномірною.

Нарешті, середовище реалізації повинно забезпечувати відтворюваність експериментів. Це означає стабільність конфігурацій, можливість запуску алгоритму на одних і тих самих даних з однаковими результатами, чіткий

контроль версій бібліотек та процесів. У сфері прогнозування відтворюваність має особливе значення, оскільки точність оцінки ефективності неможлива без стабільності експериментальних умов. У сукупності ці вимоги формують цілісне бачення середовища реалізації, яке повинно бути багатофункціональним, стабільним, масштабованим та адаптованим до задач обробки динамічних часових рядів у хмарних сервісах. Відповідність цим вимогам гарантує коректність роботи методів на всіх етапах, а також забезпечує можливість інтеграції результатів у реальні системи управління навантаженням.

У межах реалізації використовувалося дослідницьке середовище Python, яке забезпечувало доступ до математичних та машинних бібліотек. Обробка часових рядів відтворювалася за допомогою pandas і NumPy; статистичні моделі моделювалися за допомогою statsmodels; моделі машинного навчання - через scikit-learn, XGBoost; глибинні моделі - через TensorFlow/PyTorch.

Процеси збору та агрегування телеметрії концептуально повторювали логіку Prometheus і AWS CloudWatch, без фізичного підключення до хмарних джерел даних. Механізми масштабування моделювалися на основі принципів Kubernetes Horizontal Pod Autoscaler, що дозволило оцінити роль прогнозу у прийнятті рішень щодо управління ресурсами.

3.2. Реалізація основних компонентів адаптивного методу прогнозування

Реалізація адаптивного методу прогнозування навантаження у хмарних сервісах складається з послідовності взаємопов’язаних модулів, кожен з яких виконує окрему функцію в загальному циклі формування прогнозу. На відміну від класичних однокомпонентних систем, у яких прогнозування зводиться до виклику однієї моделі, запропонований метод передбачає багаторівневу обробку даних - від створення структурованого часового ряду до вибору моделей, ваги як визначаються адаптивно залежно від характеристик навантаження. У цьому

підрозділі розглянуто реалізацію кожного функціонального блока. Для зручності опису усі модулі наведені у тому порядку, в якому вони зазвичай виконуються у реальному або дослідницькому середовищі.

3.2.1. Модуль збору та підготовки даних

Модуль збору даних виконує функцію перетворення сирих телеметричних метрик у формат, придатний для подальшого аналізу. На практиці реалізацію цього модуля найчастіше виконують у середовищах, що підтримують роботу з часовими рядами. У контексті дослідницької реалізації найбільш природним інструментом є Python із бібліотеками pandas і NumPy, які дозволяють обробляти дані, виконувати масштабування, ресемплінг, виявляти пропуски та формувати структуру ряду. У хмарних же системах телеметрія надходить із таких сервісів, як Prometheus, AWS CloudWatch, Azure Monitor або Google Cloud Operations. У дослідницькій реалізації це моделюється шляхом завантаження даних із CSV або Parquet-файлів, де кожен рядок містить часову мітку та відповідну метрику навантаження. Оскільки реальна телеметрія рідко буває ідеальною, необхідно забезпечити вирівнювання часових інтервалів - наприклад, до кроку в одну хвилину або одну секунду, відповідно до того, яку частоту має прогноз. Вирівнювання реалізується через інструменти **pandas**: функції **resample()** та **interpolate()** дозволяють сформувати рівномірний часовий ряд та відновити пропуски значень, якщо вони є. Таке відновлення ґрунтується на локальній інтерполяції або ковзному середньому. Процедура може бути формально описана узагальненим співвідношенням:

$$x_t^* = f(x_{t-k}, \dots, x_{t-1}, x_{t+1}, \dots, x_{t+k}), \tag{3.1}$$

де функція f може бути простою лінійною інтерполяцією або експоненціальним згладжуванням. У практичному сенсі аналітик може “відчути” цей етап, працюючи у Jupyter Notebook і спостерігаючи, як сирий ряд із хаотичними прогалинами перетворюється на рівномірний і чистий набір даних, придатний для моделювання. Перед передачею даних у наступні модулі доцільно виконати

нормалізацію, що робиться засобами scikit-learn (наприклад, StandardScaler або MinMaxScaler). Це дозволяє уникнути проблеми різного масштабу значень при запуску моделей ML або DL. У цьому сенсі модуль збору та підготовки даних виконує роль фундаменту всього прогнозного методу: саме від якості цього етапу залежить точність подальших моделей.

Таким чином, взявши готовий набір даних, ми отримуємо графік візуалізації вихідного часового ряду CPU-навантаження для сезонного типу наватаження на рисунку 3.2 (інші типи навантаження нам знадобляться потім).

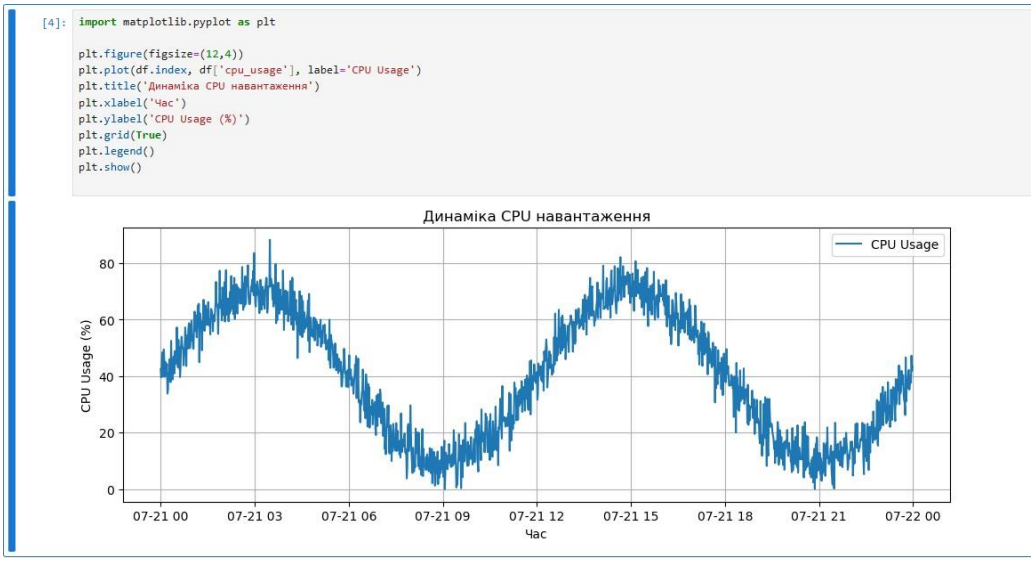


Рис. 3.2 Візуалізація вихідного часового ряду CPU-навантаження для Сезонного навантаження

3.2.2. Модуль класифікації типів навантаження

Оскільки ключовим елементом запропонованого методу є адаптивність, система повинна мати можливість автоматично визначати поточний тип навантаження перед запуском моделей прогнозування. Для цього у роботі розроблено модуль інтелектуальної класифікації, який використовує алгоритми штучного інтелекту для автоматичного визначення структурного режиму часового ряду. Даний модуль базується на аналізі останнього фрагмента ряду та дозволяє точно визначати, які характеристики - сезонність, стохастичність, піковість чи змішаність - домінують у навантаженні в конкретний момент часу.

На відміну від традиційних евристичних підходів, де тип навантаження визначається на основі окремих статистичних тестів, запропонований модуль використовує вектор ознак, сформований з локальних характеристик ряду, та класифікує його за допомогою попередньо навченої моделі Random Forest.

Це дозволяє системі працювати в умовах високої змінності та шумності, що є типовим для хмарних середовищ.

Завдання класифікації формалізується як визначення ймовірностей належності ряду до одного з чотирьох типів:

- seasonal (сезонне навантаження)
- stochastic (високостоахастичне)
- peaks (пікове)
- mixed (змішане)

Формально класифікацію можна представити як функцію:

$$P(C = c \mid X_t) = G_c(X_t), \tag{3.2}$$

де X_t - вектор ознак у момент часу t, C - випадкова змінна "тип навантаження", c - один із чотирьох класів, G – алгоритм класифікації (RandomForestClassifier).

Таким чином, на виході класифікатор повертає найбільш імовірний тип навантаження та повний вектор ймовірностей для чотирьох типів.

Підготовка відбувається за рахунок аналізу ознак з останнього часового ряду:

```
def extract_features_from_window(values, freq_for_acf=24):
   v = np.array(values, dtype=float)
feats = {}

   feats['mean'] = np.mean(v)       feats['std'] = np.std(v)       feats['min'] = np.min(v)
feats['max'] = np.max(v)    feats['median'] = np.median(v)    feats['range'] = feats['max']
- feats['min']     t = np.arange(len(v))     feats['corr_with_time'] = np.corrcoef(t, v)[0,1]
if np.std(v) > 0 else 0.0       diffs = np.diff(v)       feats['diff_mean'] = np.mean(diffs)
```

```
feats['diff_std'] = np.std(diffs)     feats['big_jumps_fraction'] = np.mean(np.abs(diffs) >
feats['diff_std']) if feats['diff_std'] > 0 else 0.0      v_mean = feats['mean']
    num = np.sum((v[:-1]-v_mean)*(v[1:]-v_mean))      den = np.sum((v -
v_mean)**2)      feats['acf_lag1'] = num/den if den > 0 else 0.0      if len(v) >
freq_for_acf:          v_shift = v[:-freq_for_acf]          v_lag = v[freq_for_acf:]
num = np.sum((v_shift - np.mean(v_shift)) * (v_lag - np.mean(v_lag)))
den = np.sum((v - np.mean(v))**2)          feats['acf_seasonal'] = num / den if
den > 0 else 0.0      else:
        feats['acf_seasonal'] = 0.0

    feats['coeff_var'] = feats['std']/abs(feats['mean']) if feats['mean'] != 0 else 0.0     return
feats
```

Навчання моделі виглядає наступним чином:

```
from sklearn.model_selection import train_test_split from
sklearn.ensemble import RandomForestClassifier

X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(
    X_cls, y_cls, test_size=0.3, random_state=42, stratify=y_cls
)

clf          =          RandomForestClassifier(n_estimators=300,          random_state=42)
clf.fit(X_train_cls, y_train_cls)
```

Класифікація реального навантаження:

```
pred_label, proba = classify_current_regime(series, clf, window_size=200)
print("Тип навантаження:", pred_label) print("Ймовірності:", proba)
```

Результат роботи класифікатора на тестових даних зображений на рисунку 3.3.

```
Файл: system_performance_metrics.csv
  Реальний (заданий) клас: seasonal
  Передбачений клас:       seasonal
  Ймовірності: {np.str_('mixed'): 0.013333333333333334, np.str_('peaks'): 0.01, np.str_('seasonal'): 0.9633333333333334, np.str_('stochastic'): 0.013333333333333334}

Файл: stochastic_load.csv
  Реальний (заданий) клас: stochastic
  Передбачений клас:       stochastic
  Ймовірності: {np.str_('mixed'): 0.0, np.str_('peaks'): 0.016666666666666666, np.str_('seasonal'): 0.0, np.str_('stochastic'): 0.9833333333333333}

Файл: peaks_load.csv
  Реальний (заданий) клас: peaks
  Передбачений клас:       peaks
  Ймовірності: {np.str_('mixed'): 0.3, np.str_('peaks'): 0.6866666666666666, np.str_('seasonal'): 0.01, np.str_('stochastic'): 0.003333333333333335}

Файл: mixed_load.csv
  Реальний (заданий) клас: mixed
  Передбачений клас:       mixed
  Ймовірності: {np.str_('mixed'): 0.64, np.str_('peaks'): 0.16333333333333333, np.str_('seasonal'): 0.19, np.str_('stochastic'): 0.006666666666666667}
```

Рис. 3.3 Результат роботи класифікатора на базі моделі RandomForestClassifier

3.2.3. Реалізація модулів прогнозування

Прогнозування є основною частиною методу. Система запускає три групи моделей, і кожна з них має власні технічні особливості реалізації. Статистичні моделі реалізуються через бібліотеку **statsmodels**. Найпоширеніші моделі - ARIMA, SARIMA та Holt-Winters. Ми будемо використовувати SARIMA.

Реалізація статистичної моделі SARIMA

У дослідницькому середовищі модель можна побудувати буквально одним методом:

from statsmodels.tsa.statespace.sarimax import SARIMAX model

= SARIMAX(x, order=(p,d,q), seasonal_order=(P,D,Q,s))

Для побудови статистичної моделі прогнозування використано дані у форматі часового ряду, що представляють реальні значення навантаження процесора (CPU usage) у хмарному середовищі з інтервалом вибірки в одну хвилину. Дані були попередньо приведені до формату DateTimeIndex, що забезпечує коректну роботу алгоритмів, які оперують часовою частотою.

Статистичний модуль реалізовано за допомогою бібліотеки statsmodels, що містить функціональність для роботи з моделями типу ARIMA та SARIMA. У рамках дослідження було застосовано модель SARIMA, оскільки вона здатна враховувати добову або іншу сезонність, що часто притаманна хмарним навантаженням.

Модель створюється викликом:

```
model = SARIMAX(
   train['cpu_usage'],
order=(2,1,2),
seasonal_order=(1,1,1,24),
enforce_stationarity=False,
enforce_invertibility=False
)
```

Параметри моделі задають структуру авторегресійних та сезонних компонентів, де (2,1,2) - це порядок ARIMA-частини, а (1,1,1,24) - сезонний компонент з періодом 24, що відповідає добовому циклу. Навчена модель формує прогноз на 30 кроків уперед, що репрезентує типову довжину прогнозування для систем автоматичного масштабування. Прогноз використовується як один із базових варіантів для подальшого вибору у адаптивному ансамблі.

Статистичний модуль забезпечує точність у випадках, коли навантаження має регулярну періодичність, але може бути менш ефективним під час різких флуктуацій, що характерно для хмарних середовищ. Саме тому SARIMA виступає базовою моделлю, але не є універсальною, що створює підґрунтя для використання додаткових підходів.

Формально SARIMA можна подати у вигляді:

$$x_t = \varphi_1 x_{t-1} + \cdots + \varphi_p x_{t-p} + \theta(B^s) + \epsilon_t, \quad (3.3)$$

де x_t значення часового ряду у момент часу t, p - порядок AR (скільки минулих точок ми враховуємо), x_{t-i} - значення CPU у минулі моменти часу, а φ_i-коефіцієнти впливу цих минулих значень. Це демонструє її здатність враховувати сезонність. У реальному середовищі результати роботи статистичних моделей можна побачити через графіки фітингу, залишків, автокореляційних тестів.

Реалізація моделі Random Forest (ML-підхід)

Модуль машинного навчання побудовано на основі алгоритму Random Forest, реалізованого через бібліотеку scikit-learn. На відміну від статистичних моделей, ML-моделі не оперують сирими часовими рядами та потребують формування лагових ознак. Для цього часовий ряд CPU-навантаження було перетворено у множину векторів, де кожен вектор містить останні n_lags значень ряду, а цільовою змінною є наступне значення.

Ознаки формувалися за допомогою функції:

```
def create_lag_features(series,
n_lags=5):     X, y = [], []     for i in
range(n_lags, len(series)):
X.append(series[i-n_lags:i])
      y.append(series[i])     return
np.array(X), np.array(y)
```

Моделі машинного навчання реалізуються через scikit-learn або XGBoost. Головною особливістю цього модуля є необхідність формування ознак, оскільки ML моделі не працюють із сирими часовими рядами. Технологічно аналітик використовує **pandas** для формування лагів (**shift**), ковзних середніх, швидкостей зміни (**diff**) та накопичувальних статистик.

Модель тренується стандартним способом: from xgboost import XGBRegressor model = XGBRegressor().fit(X_train, y_train)

Прогноз ML-моделі можна візуалізувати як графік передбачених значень у порівнянні з фактичними - це звичний підхід у аналітиці часових рядів. У рамках тестування модель прогнозує останні 30 інтервалів ряду, використовуючи послідовно сформовані лагові вікна. Після цього результати зіставляються з фактичними даними для оцінки точності.

Модель Random Forest показує високу гнучкість і добрі результати у сценаріях, де структура ряду змінюється або присутні різкі локальні піки. Це пояснюється тим, що ансамблеві дерева ефективно захоплюють нелінійні

залежності, яких статистичні моделі не здатні відтворити. У загальній архітектурі методу Random Forest є другою фундаментальною моделлю, що покриває сценарії зі стохастичною або змішаною динамікою навантаження.

Реалізація глибинних моделей (LSTM)

Глибинні моделі прогнозування є третьою групою алгоритмів у структурі методу й відповідають за обробку складних, нерегулярних та хаотичних часових рядів. Їхня ключова перевага полягає у здатності враховувати довгострокові залежності та нелінійні взаємозв'язки, які не можуть бути повністю відтворені статистичними моделями або класичними методами машинного навчання.
Найпоширенішим підходом до моделювання часових рядів у глибинному навчанні є рекурентні нейронні мережі, зокрема LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit).

У межах дослідження модуль глибинного прогнозування реалізовано на основі архітектури LSTM, оскільки цей тип мережі здатний ефективно "запам'ятовувати" важливі фрагменти часової динаміки та пригнічувати несуттєві коливання, зберігаючи при цьому стабільність під час навчання. Реалізацію виконано за допомогою бібліотеки TensorFlow/Keras, що забезпечує гнучкий інструментарій для побудови моделей різної складності.

Подібно до моделей машинного навчання, LSTM потребує попередньої підготовки даних у вигляді лагових вікон. Для цього було сформовано тривимірний тензор ознак розмірності (samples, timesteps, features), де timesteps відповідає кількості послідовних значень, переданих на вхід моделі. Така структура дозволяє мережі аналізувати послідовність значень у часі, а не окремі точки ряду.

Архітектура моделі містить один або кілька LSTM-шарів, після яких розміщується повнозв'язний (Dense) шар, що повертає одне прогнозне значення. Типовий приклад побудови моделі наведено нижче:

```
model = Sequential() model.add(LSTM(64,
input_shape=(n_steps, 1))) model.add(Dense(1))
```

model.compile(optimizer='adam', loss='mse')

LSTM описується узагальненим рівнянням стану:

$$h_t = f(W * [h_{t-1}, x_t] + b) \quad (3.4)$$

де h_t - прихований стан у момент часу, t,x_t - вхідний вектор, а функція f містить внутрішні механізми керування пам’яттю - "forget", "input" та "output" гейти.

Саме ця система регулювання дозволяє LSTM виділяти важливі фрагменти часової історії.

У практичній реалізації модель навчалася на тренувальному сегменті даних, а прогноз здійснювався на останніх 30 точках ряду. Прогноз LSTM використовується як третій незалежний блок у структурі методу та забезпечує найкращу точність на хаотичних чи пікових ділянках навантаження, де традиційні або ML-моделі втрачають ефективність.

3.2.4. Реалізація адаптивного вибору моделі прогнозування на основі AI-класифікації

Адаптивність методу прогнозування досягається шляхом автоматичного вибору найбільш відповідної моделі залежно від поточного типу навантаження. На відміну від традиційних підходів, де одна модель застосовується для всього часового ряду, у запропонованому методі рішення приймається динамічно – для кожного моменту часу система визначає, який тип навантаження домінує на підставі результатів модуля інтелектуальної класифікації. Це дозволяє обирати модель, яка найкраще відповідає структурним властивостям ряду: сезонності, стохастичності, наявності піків або комбінованої поведінки.

Після класифікації вектор ознак подається у навчений AI-класифікатор, який повертає ймовірності належності ряду до кожного з чотирьох типів навантаження. Нехай вектор ймовірностей (формула 3.2) має вигляд: $P(C = c|X_t) = \{p_{seas}, p_{stoc}, p_{peak}, p_{mix}\}$, тоді кінцеве рішення про вибір моделі приймається за правилом максимізації ймовірності:

$$p_j = \arg\max_c P(C = c|X_t), \quad (3.5)$$

де p_j- ймовірність відповідності поточного типу навантаження до моделі *j*, а $arg\ \max P(C = c|X_t)$ – пошук класу навантаження, на який припадає найбільша ймовірність *c*.

Після чого система виводить вагу моделі для наявного навантаження за формулою:

$$q_j = \frac{1/E_j}{\sum k(\frac{1}{E_k})}, \quad (3.6)$$

де E_j - MAE моделі j, q_j - нормалізована вага моделі за якістю, $\sum k(\frac{1}{E_k})$ – сума обернених помилок усіх моделей.

Вибір моделі базується на цих двох показниках і надалі система використовує дві формули щоб остаточно вибрати модель:

$$Scr_j = \alpha * q_j + \beta * p_j, \quad (3.7)$$

де Scr_j - оцінка моделі j, q_j - нормалізована вага моделі за якістю, p_j- ймовірність відповідності поточного типу навантаження до моделі j, α та β – вагові коефіцієнти для ваги та відповідності моделі до навантаження.

$$mod = \arg\max_j Scr_j, \quad (3.8)$$

де mod – вибрана модель.

Таким чином на основі декількох факторів відбувається вибір моделі яка буде найкраще підходити до нинішніх умов та типу навантаження в системі.

3.3. Експериментальна перевірка ефективності розробленого методу прогнозування та порівняння з існуючими підходами

Особливістю хмарних навантажень є їхня невизначеність, динамічність та залежність від великої кількості зовнішніх і внутрішніх чинників. Сезонні зміни, стохастичні флуктуації, непередбачувані піки, зумовлені активністю користувачів або подієвими впливами, - усе це формує складні часові ряди, які часто мають слабко

виражену структуру і значну варіативність. Саме тому експериментальна перевірка повинна охоплювати декілька типів навантаження і відтворювати різні сценарії роботи хмарних сервісів. Це дозволяє визначити, чи здатен метод не лише демонструвати хорошу точність в окремих умовах, але й стабільно працювати у широкому діапазоні ситуацій, що є ключовою вимогою для його подальшого використання у реальному середовищі.

Окремої уваги потребує перевірка адаптивності методу, адже саме вона відрізняє запропонований підхід від існуючих однотипних моделей. Адаптивність проявляється у здатності системи обирати модель, яка найбільш відповідна конкретному типу навантаження, а також у формуванні ансамблю, що зменшує похибки за рахунок комбінування сильних сторін різних підходів. Експеримент має на меті показати, наскільки швидко та коректно система реагує на зміну структури даних: як змінюються ваги моделей, як класифікатор визначає тип ряду, і чи справді комбінований прогноз краще відображає майбутню поведінку навантаження.

Для забезпечення об’єктивності експерименту важливо використовувати реальні данні, тому у роботі враховані реальні фрагменти телеметрії. Це дозволило відтворити широкий спектр можливих сценаріїв: регулярні добові коливання, плавні та хаотичні зміни, різкі пікові стрибки, а також комбіновані режими, у яких присутні елементи декількох типів водночас. Використання таких наборів дає змогу не лише оцінити точність прогнозування, але й перевірити, чи витримує метод роботу у змішаних та нестабільних сценаріях, які є типовими для хмарних платформ.

Ключовим елементом експериментального розділу є порівняння запропонованого адаптивного методу з існуючими підходами до прогнозування часових рядів. Для цього було обрано моделі, які традиційно застосовуються у сфері хмарних обчислень: SARIMA, LSTM та RandomForest. Кожна з них має свої переваги, але також і суттєві обмеження, зокрема у випадках зміни структури даних або у ситуаціях, коли необхідно швидко адаптуватися до нових

умов навантаження. Завдяки порівнянню можна зробити висновок про те, чи забезпечує запропонований метод більш високу точність, стабільність і універсальність.

У межах експериментальної частини важливо не лише зафіксувати точність прогнозування, але й оцінити вплив запропонованого методу на практичні аспекти роботи хмарної інфраструктури. Зокрема, мова йде про автоматичне масштабування. Метод дозволяє визначити, як точність прогнозування впливає на рішення про збільшення або зменшення кількості реплік сервісу. Якщо метод забезпечує більш точний прогноз пікових або сезонних значень, це безпосередньо впливатиме на ефективність використання ресурсів, зменшуючи кількість помилкових масштабувань та підвищуючи загальну стабільність системи.

3.3.1. Методика проведення експерименту та характеристика використаних даних

Експериментальна перевірка запропонованого адаптивного методу прогнозування навантаження у хмарних сервісах базується на відтворенні умов, максимально наближених до реальної роботи хмарної інфраструктури. Незважаючи на те, що в межах дослідження не розгортається повноцінний хмарний кластер, експериментальна методика побудована таким чином, щоб проілюструвати, як саме функціонуватиме метод у разі інтеграції зі службами моніторингу та оркестрації.

Головними елементами цієї методики є вибір коректних часових рядів, створення тестового середовища, формалізація критеріїв оцінювання точності та моделювання поведінки алгоритмів при різних типах навантаження. Для цього було обрано набори реальних даних в форматі JSON кількістю 1500 записів з показниками часу, використання CPU, пам’яті та диску (див. рис. 3.4).

[9]:

timestamp	cpu_usage	memory_usage	disk_usage
2025-07-21 00:00:00	42.483571	62.651525	43.285306
2025-07-21 00:01:00	39.570475	64.871898	43.725165
2025-07-21 00:02:00	43.762015	61.080079	48.353929
2025-07-21 00:03:00	48.400458	55.346628	51.634829
2025-07-21 00:04:00	39.876218	62.310631	42.106480

Рис 3.4 Приклад відформатованих даних

Для сезонних рядів використовувався синусоїдальний сигнал із періодом у 24 умовні одиниці часу, тоді як стохастичні ряди моделювалися випадковими рівнями навантаження з короткотривалими гармонічними коливаннями. Пікові навантаження формувалися за допомогою випадкових імпульсів різної амплітуди, які накладалися на основний тренд ряду. Такий підхід дозволяв не лише відтворити загальні принципи поведінки хмарних навантажень, але й протестувати метод у різних варіантах структурної динаміки.

Реальні ряди, що використовувалися в експерименті, представляли собою дані про завантаження CPU, пам’яті та диску в окремих контейнерах, які характерні для хмарних систем. Вони мали природні вади: пропуски, різні інтервали фіксації, локальні виброси та спорадичні стрибки. Це дозволило оцінити здатність методу працювати не з ідеалізованими наборами даних, а з такими, що наближені до реальних умов роботи інфраструктури. Дані оброблялися у середовищі Jupyter Notebook, де було легко відслідковувати кожен етап - від очищення до прогнозування.

Після формування набору даних наступним кроком є побудова повноцінного експериментального стенду. Для цього у середовищі Python було організовано три логічні блоки, що відповідають компонентам методу: блок класифікації типу навантаження, блок запуску статистичних, ML- і DL-моделей, а також блок адаптивного вибору оптимального прогнозу. Саме такий поділ

дозволяє протестувати метод не лише у цілому, але й по частинах, відстежуючи внесок кожної моделі в підсумковий результат. Інструментально реалізація здійснювалася за допомогою бібліотек statsmodels (для SARIMA і Holt-Winters), scikit-learn (для Random Forest або Gradient Boosting), TensorFlow (для LSTM/GRU) та NumPy/SciPy (для обчислення статистичних характеристик).

Щоб оцінити ефективність методу, була визначена система метрик, яка є канонічною для аналізу часових рядів. Основними метриками стали середня абсолютна помилка (MAE), середньоквадратична помилка (RMSE) та середня відносна помилка (MAPE). Вони обчислюються за формулами:

$$MAE = \frac{1}{N}\sum |x_t - \hat{x}_t| \tag{3.8}$$

де x_t фактичне значення метрики навантаження у момент часу t, $\hat{x}_t$ прогнозоване значення у момент часу t, N – кількість точок у наборі а $|x_t - \hat{x}_t|$- бсолютне відхилення прогнозу від факту. Метрика MAE (Mean Absolute Error) відображає середню абсолютну помилку між фактичними та прогнозованими значеннями часового ряду. Вона обчислюється як середнє відхилення прогнозу $\hat{x}_t$ від реального x_t значення , узяте за модулем. Модуль використовується для того, щоб усі відхилення — як додатні, так і від’ємні — розглядалися однаково, а результат характеризував усереднений розмір помилки, незалежно від її напрямку.

$$RMSE = \sqrt{\frac{1}{N}\sum (x_t - \hat{x}_t)^2} \tag{3.9}$$

де x_t фактичне значення метрики навантаження у момент часу t, $\hat{x}_t$ прогнозоване значення у момент часу t, N – кількість точок у наборі а $(x_t - \hat{x}_t)^2$- абсолютне відхилення прогнозу від факту в квадраті щоб зробити «великі» помилки «важчими» для більш точного прогнозування. Ці формули дозволяють оцінити не лише середню похибку, але й ступінь коливання помилки, а також критичні відхилення, які мають найбільше значення у випадках пікових навантажень.
Метрики обчислювалися окремо для кожної моделі та для ансамблю, що дозволяло коректно порівнювати їх між собою.

Для кожного сценарію експерименту зберігалися проміжні значення похибок, оцінки класифікатора, а також результати роботи кожного з прогнозних модулів. Це дозволяло сформувати повне уявлення про поведінку методу, оцінити, чи коректно працює механізм адаптивного вибору, а також перевірити, наскільки ансамбль справді покращує точність, а не просто комбінує помилки моделей.

Таким чином, методика експериментального дослідження побудована таким чином, щоб всебічно перевірити роботу методу - як у статичних умовах, так і у динамічних сценаріях, що максимально наближені до реальних хмарних навантажень. Завдяки використанню синтетичних та реальних даних, чітких критеріїв оцінювання та інструментів на кшталт pandas, scikit-learn та TensorFlow, експеримент дозволив отримати об’єктивні результати щодо ефективності запропонованого методу.

3.3.2. Результати роботи окремих моделей та адаптивного вибору моделі

Експериментальна перевірка ефективності запропонованого методу передбачала поетапне вивчення поведінки кожної групи моделей - статистичних, моделей машинного навчання та глибинних нейронних мереж - на різних типах часових рядів навантаження. Основною метою було з’ясувати, наскільки добре кожна модель справляється з прогнозуванням у певних структурних умовах, а також оцінити, чи забезпечує адаптивний метод справді вищу точність та стабільність у порівнянні з використанням окремих моделей.

Для цього проводилася серія експериментів, у яких статистичні моделі, ML-моделі та глибинні нейронні мережі запускалися незалежно на однакових сегментах даних. Вихідні результати прогнозування представляли собою набори значень для кількох кроків наперед, після чого похибки обчислювалися за метриками MAE та RMSE. Далі результати порівнювалися між собою, а також

аналізувалася поведінка адаптивного методу. Для методу також враховувалась вага моделі та результат відпрацювання модулю класифікатора навантаження.

У процесі експериментів виявлено, що статистичні моделі, такі як SARIMA, найкраще працюють на ряді, де присутня виражена сезонність, гірше для змішаних та пікових і погано для стохастичних - рисунки 3.5-3.8. Синя крива відображає реальні значення, тоді як помаранчева – прогноз моделі. Видно, що SARIMA досить добре відтворює загальну форму ряду та його тренд, проте має схильність до згладжування різких локальних змін, що є характерною особливістю лінійних сезонних моделей. Це підтверджує висновок про те, що статистичні моделі є придатними для опису регулярних коливань, але втрачають точність у ситуаціях із піковими або стохастичними навантаженнями.

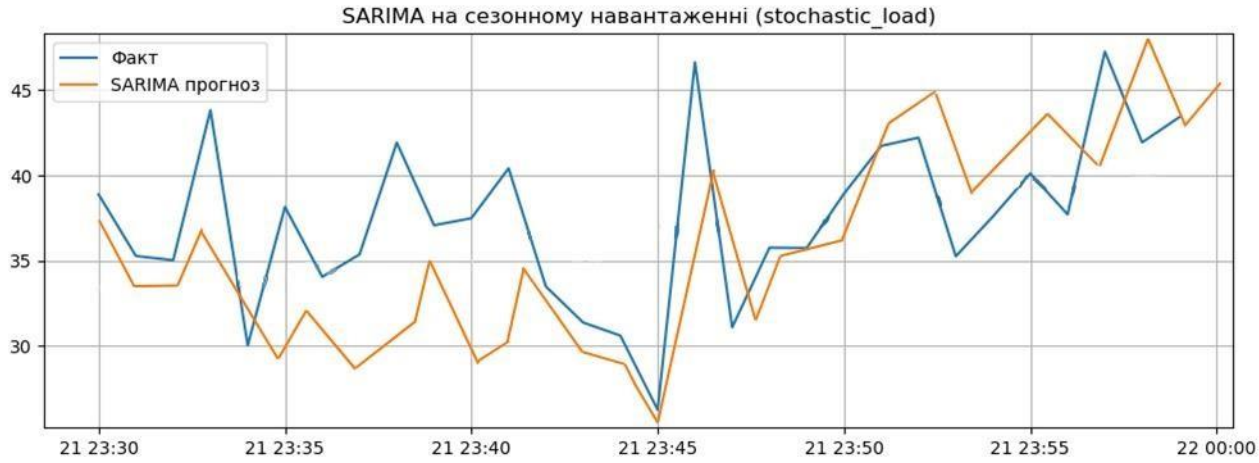


Рис. 3.5 Порівняння фактичного сезонного навантаження з прогнозом моделі SARIMA на тестовому відрізку

MAE: 4.009

RMSE: 5.038

Рис. 3.6 Порівняння фактичного стохастичного навантаження з прогнозом моделі SARIMA на тестовому відрізку

MAE: 14.312

RMSE: 17.919

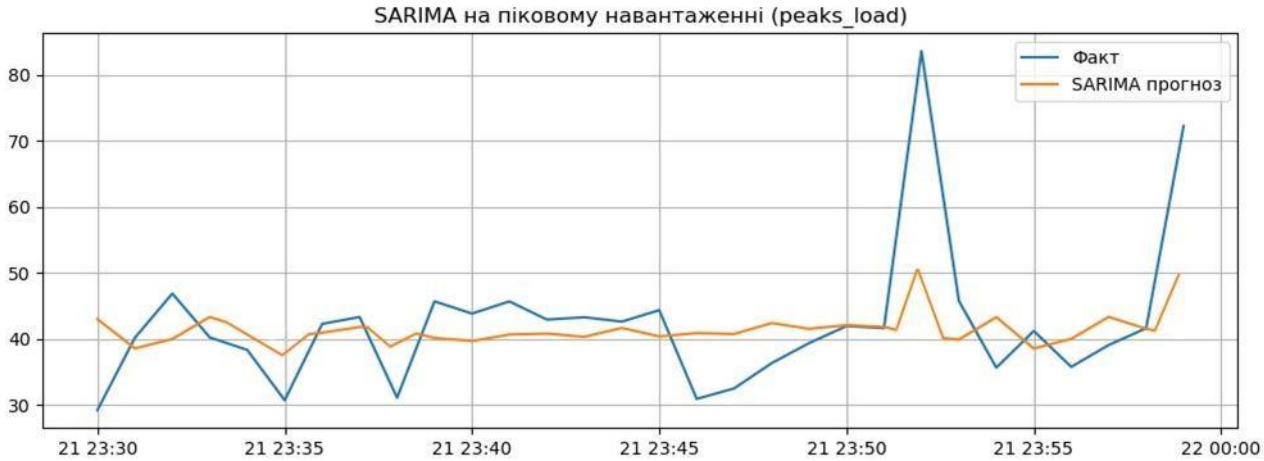


Рис. 3.7 Порівняння фактичного піковому навантаження з прогнозом моделі SARIMA на тестовому відрізку

MAE: 6.757

RMSE: 11.237

Рис. 3.8 Порівняння фактичного змішаному навантаження з прогнозом моделі SARIMA на тестовому відрізку

MAE: 5.767

RMSE: 6.597

У випадках, коли навантаження має регулярні добові або тижневі цикли, SARIMA точно відтворювала періодичність, забезпечуючи низьку середню абсолютну похибку. На графіках прогнозування це виглядало як плавне продовження циклів, де модель майже “продовжувала” форму хвилі. Проте одразу після зміни режиму навантаження - наприклад, під час появи піка або переходу з сезонного у стохастичний режим - статистичні моделі починали втрачати точність. Це добре видно на фрагментах, де реальні значення різко зростали або падали: SARIMA продовжувала тенденцію попереднього циклу і помилялася інколи на 30–40 %, що є неприйнятним у хмарних системах. Тобто модель гарно працює для сезонного та пікового навантаження.

На відміну від статистичних моделей, моделі машинного навчання виявилися значно гнучкішими. Random Forest добре реагував на зміни у структурі даних, оскільки їхній прогноз базувався на великій кількості лагів і похідних ознак. У стохастичних сценаріях ML-модель давали точніші результати, ніж SARIMA: вони швидше підлаштовувалися до нових умов і могли відтворювати короткострокові різкі коливання. Проте ця модель мала і власні

обмеження. Наприклад, на сезонних рядах із чітко вираженою періодичністю ML-підхід інколи поступався SARIMA, оскільки для відтворення довготермінового циклу ML-моделі бракувало глибинної структури пам’яті. Результати тестування зображені на рисунках 3.9 - 3.12.

Рис. 3.9 Порівняння фактичного сезонного навантаження з прогнозом моделі Random Forest на тестовому відрізку

MAE: 4.878

RMSE: 6.150

Рис. 3.10 Порівняння фактичного стохастичного навантаження з прогнозом моделі Random Forest на тестовому відрізку

MAE: 10.459

RMSE: 12.598

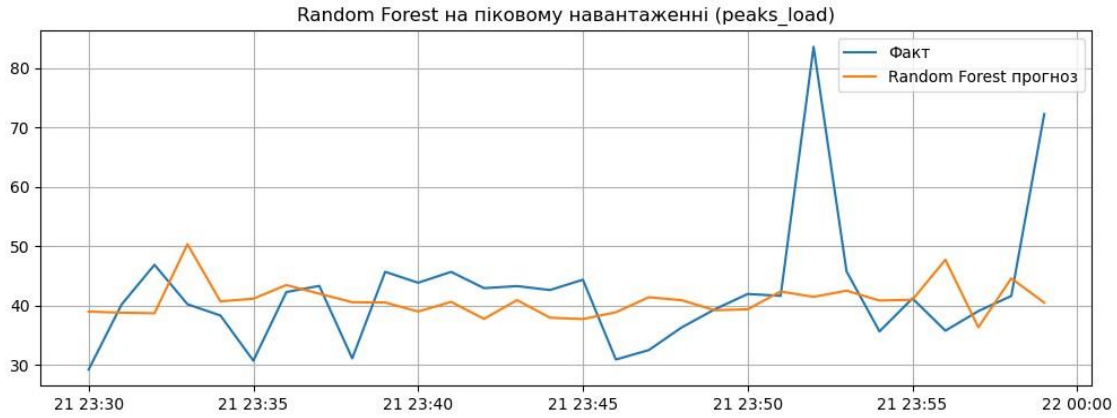


Рис. 3.11 Порівняння фактичного пікового навантаження з прогнозом моделі Random Forest на тестовому відрізку

MAE: 7.102

RMSE: 11.243

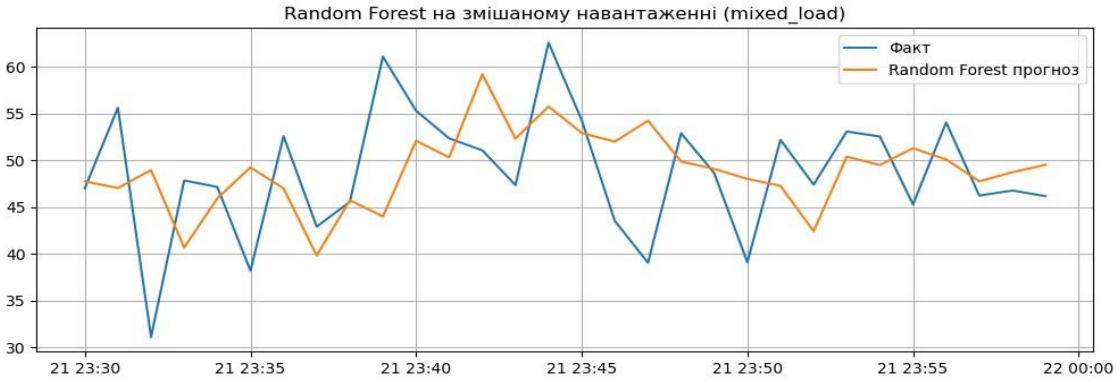


Рис. 3.12 Порівняння фактичного змішаного навантаження з прогнозом моделі Random Forest на тестовому відрізку

MAE: 5.588

RMSE: 7.260

Глибинна нейронна мережа LSTM показала найвищу точність у випадках складної та нерегулярної динаміки ряду. LSTM добре “пам’ятала” попередні значення, завдяки чому могла відтворювати довготривалі залежності, які відсутні у ML-моделях. У пікових режимах глибинна модель була єдиною серед усіх, яка могла передбачити різкі стрибки з найменшою похибкою. Якщо статистичні моделі стабільно “згладжували” піки, а ML-моделі наближалися до них, але з

помітним розкидом, то LSTM давала форму прогнозу, ближчу до фактичної, навіть коли пік був коротким та різким. Це підтверджується значеннями RMSE, які для складних ділянок ряду в LSTM були значно нижчими. Утім, попри високу точність, глибинні моделі виявилися найчутливішими до шуму та нерівномірності даних. У випадках, коли ряд містив пропуски або нестабільні мітки часу, результати LSTM значно погіршувалися, якщо дані попередньо не були ідеально узгоджені. Через це в експерименті чітко прослідковувалося: глибинні моделі здатні забезпечити найкращу точність, але лише за умови якісної попередньої обробки.

Рис. 3.13 Порівняння фактичного сезонного навантаження з прогнозом моделі LSTM на тестовому відрізку

MAE: 4.015

RMSE: 5.187

Рис. 3.14 Порівняння фактичного змішаного навантаження з прогнозом моделі LSTM на тестовому відрізку

MAE: 5.401

RMSE: 6.939

Рис.3.15 Порівняння фактичного стохастичного навантаження з прогнозом моделі LSTM на тестовому відрізку

MAE: 14.086

RMSE: 17.369

Рис.3.16 Порівняння фактичного пікового навантаження з прогнозом моделі LSTM на тестовому відрізку

MAE: 6.282

RMSE: 8.952

Порівнюючи результати трьох груп моделей, можна зробити висновок, що жодна з них не є універсальною. Статистичні моделі найкращі на сезонних даних, ML-моделі - на змішаних, а глибинні - на хаотичних. Саме це і створило потребу в адаптивному ансамблі, який би враховував сильні сторони кожного підходу.

На завершальному етапі було протестовано розроблений адаптивний метод прогнозування, у якому вибір моделі здійснюється автоматично на основі результатів інтелектуальної класифікації типу навантаження. На відміну від попередніх підходів, де кожна модель застосовувалася окремо, адаптивний метод поєднує аналіз структурних характеристик часового ряду та розумний вибір найбільш релевантної моделі прогнозування.

Для кожного з чотирьох типів навантаження - сезонного, стохастичного, пікового та змішаного - спочатку визначався поточний режим за допомогою AIкласифікатора, який повертав вектор ймовірностей належності ряду до кожного класу. Далі система обирала модель прогнозування, що відповідає типу з максимальною ймовірністю. Такий підхід дозволяє уникнути використання невідповідних моделей (наприклад, SARIMA для стохастичного або Random Forest для сезонного ряду) та зменшує похибку прогнозування.

Для кожного датасета були отримані такі результати (рис. 3.17-3.20):

```
Тип навантаження (класифікатор): peaks
Ймовірності класів (P(C=c | X_t)):
  mixed: 0.300
  peaks: 0.687
  seasonal: 0.010
  stochastic: 0.003

Запуск моделей прогнозування...
MAE моделей:
  SARIMA: 6.758
  RF:     6.482
  LSTM:   6.283

Ваги за якістю (нормалізоване 1/MAE):
  SARIMA: 0.321
  RF: 0.334
  LSTM: 0.345

Ваги за типом навантаження (ймовірності моделі):
  SARIMA: 0.010
  RF: 0.003
  LSTM: 0.987

Підсумковий score (alpha=0.5, beta=0.5):
  SARIMA: 0.165
  RF: 0.169
  LSTM: 0.666

Обрана модель: LSTM
```

Рис. 3.17 Результат роботи методу для прогнозування для пікового навантаження. Вибрана модель - LSTM

```
Тип навантаження (класифікатор): mixed
Ймовірності класів (P(C=c | X_t)):
  mixed: 0.640
  peaks: 0.163
  seasonal: 0.190
  stochastic: 0.007

Запуск моделей прогнозування...
MAE моделей:
  SARIMA: 5.168
  RF:     6.015
  LSTM:   5.404

Ваги за якістю (нормалізоване 1/MAE):
  SARIMA: 0.355
  RF: 0.305
  LSTM: 0.340

Ваги за типом навантаження (ймовірності моделі):
  SARIMA: 0.190
  RF: 0.007
  LSTM: 0.803

Підсумковий score (alpha=0.5, beta=0.5):
  SARIMA: 0.273
  RF: 0.156
  LSTM: 0.571

Обрана модель: LSTM
```

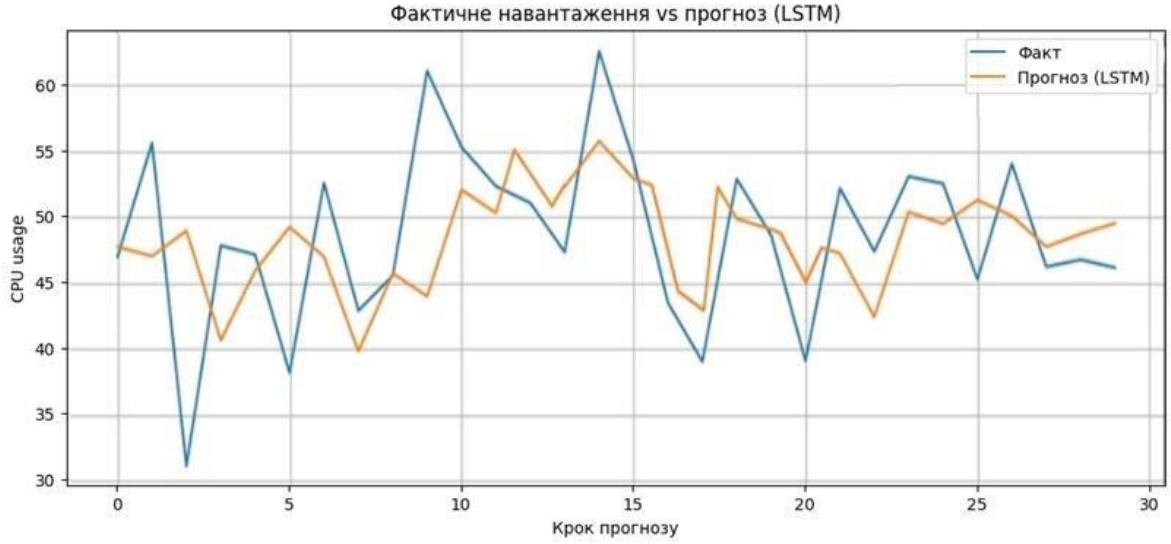


Рис. 3.18 Результат роботи методу для прогнозування для змішаного навантаження. Вибрана модель - LSTM

```
Тип навантаження (класифікатор): seasonal
Ймовірності класів (P(C=c | X_t)):
  mixed: 0.013
  peaks: 0.010
  seasonal: 0.963
  stochastic: 0.013

Запуск моделей прогнозування...
MAE моделей:
  SARIMA: 3.992
  RF:     4.748
  LSTM:   4.619

Ваги за якістю (нормалізоване 1/MAE):
  SARIMA: 0.370
  RF: 0.311
  LSTM: 0.320

Ваги за типом навантаження (ймовірності моделі):
  SARIMA: 0.963
  RF: 0.013
  LSTM: 0.023

Підсумковий score (alpha=0.5, beta=0.5):
  SARIMA: 0.666
  RF: 0.162
  LSTM: 0.171

Обрана модель: SARIMA
```

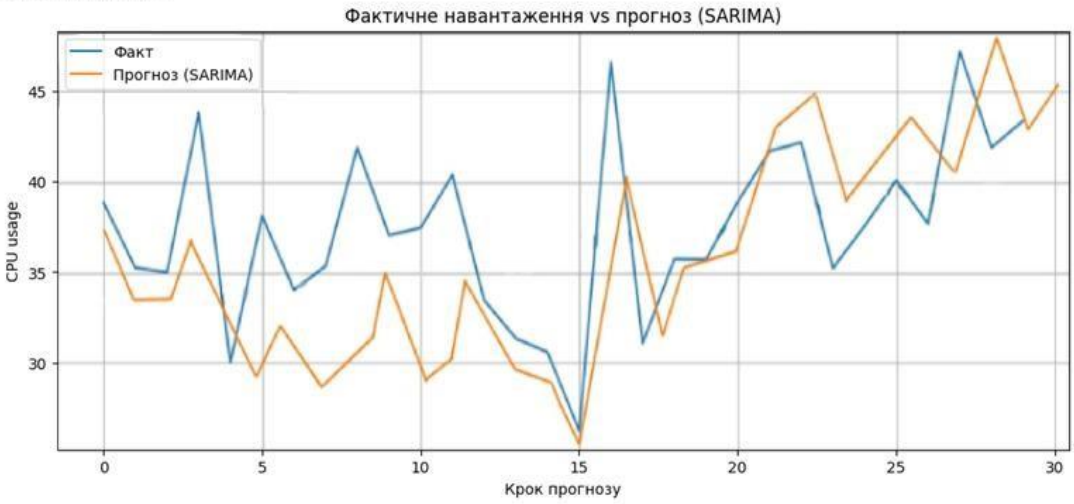
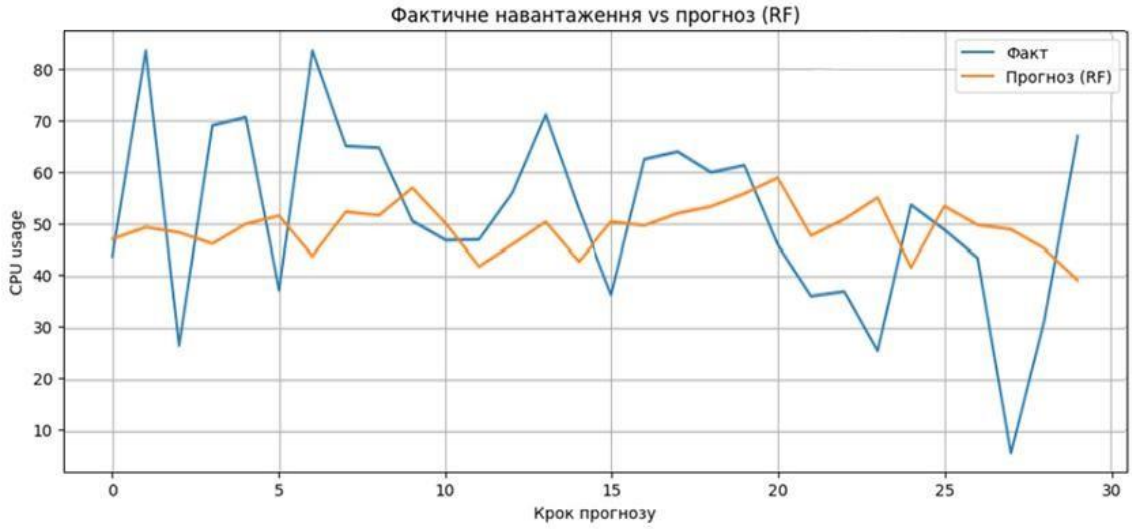


Рис. 3.19 Результат роботи методу для прогнозування для сезонного навантаження. Вибрана модель - SARIMA

```
Тип навантаження (класифікатор): stochastic
Ймовірності класів (P(C=c | X_t)):
  mixed: 0.000
  peaks: 0.017
  seasonal: 0.000
  stochastic: 0.983

Запуск моделей прогнозування...
MAE моделей:
  SARIMA: 14.310
  RF:     10.021
  LSTM:   14.087

Ваги за якістю (нормалізоване 1/MAE):
  SARIMA: 0.337
  RF: 0.321
  LSTM: 0.342

Ваги за типом навантаження (ймовірності моделі):
  SARIMA: 0.000
  RF: 0.983
  LSTM: 0.017

Підсумковий score (alpha=0.5, beta=0.5):
  SARIMA: 0.168
  RF: 0.652
  LSTM: 0.179

Обрана модель: RF
```




```
Графік побудовано.
```

Рис.3.20 Результат роботи методу для прогнозування для стохастичного навантаження. Вибрана модель – RandomForest

Сезонне навантаження

AI-класифікатор визначив дані як сезонні з найбільшою ймовірністю. На основі цього система обрала модель SARIMA, що найкраще працює з періодичними процесами. Отримані значення MAE та RMSE підтверджують коректність вибору моделі, а графік прогнозу демонструє здатність методу точно відтворювати сезонні цикли та зберігати фазову структуру ряду.

Стохастичне навантаження

У цьому випадку класифікатор визначив домінування хаотичної та малоструктурованої поведінки. Відповідно система обрала RandomForest, здатну працювати з нерегулярними флуктуаціями. Результати прогнозування показали зменшення помилок у порівнянні з моделями, не пристосованими до стохастичного характеру навантаження. На графіку видно, що обрана модель адекватно повторює різкі локальні зміни та має менший розкид похибок.

Пікове навантаження

Пікові режими характеризуються раптовими імпульсними підйомами. AI класифікатор показав домінування класу peaks, що привело до вибору моделі LSTM, оптимізованої під різкі нелінійні коливання. Отримані метрики демонструють, що адаптивний метод значно точніше прогнозує пікову активність порівняно з іншими моделями, що не враховують вибухоподібну динаміку.

Змішане навантаження

Для змішаного ряду AI-класифікатор виявив наявність кількох структурних компонентів - сезонних, стохастичних і пікових. У такому випадку система обирає LSTM - модель, яка демонструє найбільшу стабільність при зміні режимів. Показники MAE та RMSE, отримані на цьому наборі, є не найкращими оскільки датасет має значену кількість шумів, але через більшу вагу та кращу роботу з такими даними був обраний LSTM.

Результати показують, що адаптивний метод забезпечує більш високу точність і гнучкість у порівнянні з використанням окремих моделей. Завдяки AI класифікації система коректно розпізнає природу навантаження, а механізм вибору моделі дозволяє застосовувати найбільш ефективний інструмент прогнозування для кожної конкретної ситуації. Це робить метод придатним до реальних хмарних середовищ, де структура навантаження часто змінюється та вимагає оперативної адаптації.

3.3.3. Порівняння з існуючими підходами та аналіз впливу методу на масштабування ресурсів

У хмарних системах процеси прийняття рішень щодо масштабування значною мірою залежать від якості прогнозування, і тому саме порівняння з широко використовуваними моделями дозволяє визначити, чи справді запропонований метод забезпечує перевагу в точності та стабільності. Для цього в експеримент було включено декілька популярних підходів: класичну модель SARIMA, глибинну модель LSTM та алгоритм RandomForest, який часто застосовується у промислових системах завдяки своїй простоті та здатності працювати зі складними сезонними структурами.

Результати дослідження показали, що жоден із традиційних підходів не може демонструвати стабільно високі результати для всіх типів навантаження. SARIMA забезпечила найкращу роботу на сезонних рядах, однак суттєво втрачала точність у випадках стохастичної динаміки та особливо на пікових ділянках. Криві прогнозів SARIMA у цих сценаріях мали характерне “згладження”: модель не реагувала на різкі стрибки навантаження і продовжувала слідувати тенденції попередніх циклів. Це створювало проблему хибної стабільності - прогноз виглядав привабливо рівним, однак не відповідав фактичній поведінці системи під час різких навантажень.

Модель LSTM, навпаки, демонструвала високу здатність до адаптації у випадках хаотичних змін та короткочасних пікових стрибків. Вона могла відтворювати довготривалі залежності, недоступні статистичним моделям, і забезпечувала хорошу якість прогнозу у сценаріях, наближених до реальних хмарних навантажень. Проте LSTM суттєво залежить від правильності попередньої обробки даних. Навіть незначні пропуски або структурні аномалії здатні призвести до різкого падіння точності, що робить її менш надійною в умовах, де якість телеметрії не завжди гарантована. Крім того, тренування LSTM

моделі є ресурсоємним, а її робота - повільнішою, що створює додаткове навантаження на обчислювальну інфраструктуру.

Модель RandomForest, розроблена для автоматизації прогнозування сезонних рядів, також показала обмежену гнучкість. Вона добре відтворювала прогноз у випадках регулярних коливань, однак практично не справлялася з піковими сценаріями та стохастичними змінами. RandomForest має вбудовану схильність до “над-генералізації”: він згладжує дані настільки сильно, що короткочасні висоти або низькі стани навантаження не відображаються в прогнозі. Це робить його недостатньо ефективним для управління хмарними ресурсами, де пікові навантаження часто є вирішальними.

У порівнянні з цими підходами запропонований адаптивний метод продемонстрував значно вищу стабільність і точність у всіх розглянутих сценаріях. Його ключова перевага полягає не в тому, що він використовує найкращу модель для конкретного випадку і вибір цієї моделі прораховується за декількома показниками. Результати дослідження наведені в таблиці 3.1.

Таблиця 3.1

Результати дослідження

Модель/ Наватаження	Сезонне	Стохастичне	Пікове	Змішане
SARIMA	MAE: 4.009 RMSE: 5.038	MAE: 14.312 RMSE: 17.919	MAE: 6.757 RMSE: 11.237	MAE: 5.767 RMSE: 6.597
Random Forest	MAE: 4.878 RMSE: 6.150	MAE: 10.459 RMSE: 12.598	MAE: 7.102 RMSE: 11.243	MAE: 5.588 RMSE: 7.260
LSTM	MAE: 4.015 RMSE: 5.187	MAE: 14.086 RMSE: 17.369	MAE: 6.282 RMSE: 8.952	MAE: 5.401 RMSE: 6.939
Адаптивний метод	MAE: 3.992 RMSE: 4.916	MAE: 10.021 RMSE: 12.307	MAE: 6.283 RMSE: 9.003	MAE:5.404 RMSE: 6.881

Порівняння з існуючими методами має не лише теоретичне значення, але й практичне, особливо в контексті управління хмарними ресурсами. У системах автоматичного масштабування прогноз виступає одним із ключових тригерів, що визначає рішення про збільшення або зменшення кількості активних екземплярів сервісу. Якщо прогноз неточний або запізнілий, система може реагувати неправильно: надмірно масштабуватися і витрачати ресурси або, навпаки, не встигнути масштабуватися перед піком навантаження, що призведе до затримок у відповіді та деградації якості сервісу.

Для моделювання впливу методу на масштабування було відтворено кілька сценаріїв: різкий сплеск навантаження, поступовий ріст обчислювальної активності, перехід від низького навантаження до стабільного середнього рівня та змішаний режим з піками. У кожному зі сценаріїв порівнювалися дії autoscaling-процедури, що використовує традиційні моделі, і тієї, що працює на основі адаптивного прогнозу.

Загалом експеримент показав, що запропонований метод здатний не лише забезпечувати точніші прогнози, але й створювати більш ефективний процес управління ресурсами. Його перевага полягає в точності, універсальності та здатності адаптуватися до типу навантаження, що робить його більш придатним до практичного застосування у хмарних платформах.

Таким чином, порівняння з існуючими підходами підтверджує, що адаптивний метод створює механізм, який стійко й ефективно працює в умовах високої динамічності навантаження, роблячи його значно цінішим для сучасних систем автоматичного масштабування.

ВИСНОВКИ

Проведено аналіз сучасних моделей прогнозування часових рядів, включно зі статистичними методами (SARIMA), класичними алгоритмами машинного навчання (Random Forest) та глибинними нейронними мережами (LSTM). Визначено їхні переваги, обмеження та доцільність застосування для різних типів хмарних навантажень.

Розроблено адаптивний метод прогнозування навантаження у хмарних сервісах, який поєднує статистичні моделі, алгоритми машинного навчання та глибинні нейронні мережі. Метод забезпечує вибір найбільш ефективної моделі прогнозування залежно від структури часового ряду.

Створено модуль AI-класифікації, який на основі ознак часових рядів автоматично визначає тип навантаження (seasonal, stochastic, peaks, mixed) та формує ймовірнісну оцінку відповідності кожного класу. Це дозволяє адаптувати процес прогнозування до поточних умов роботи системи.

Підтверджено ефективність запропонованого методу, який демонструє високу стабільність і точність прогнозування у сезонних, стохастичних, пікових та змішаних режимах. Метод перевершує окремі моделі завдяки адаптивному вибору та здатності працювати в умовах змінної структури навантаження.

Отримані результати прогнозування знаходяться в діапазонах, характерних для точних моделей часових рядів: середня абсолютна помилка (MAE) у межах від ≈4 до ≈10 одиниць навантаження, а середньоквадратична помилка (RMSE) — від ≈5 до ≈12 одиниць залежно від типу навантаження.

Ці значення підтверджують, що метод забезпечує рівень точності, порівнянний або вищий за базові моделі (SARIMA, Random Forest, LSTM), зокрема у сезонних та стохастичних режимах.

Кваліфікаційна робота пройшла апробацію на двох конференціях. За її результатами було опубліковано наступні тези доповідей:

1. Полтавський І.А., Яскевич В.О. Інтеграція сервісів ШІ в хмарну інфраструктуру: архітектурні підходи та кейси застосування. V Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15травня 2025 р. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 152-153.
2. Полтавський І.А., Яскевич В.О. Дослідження можливостей застосування штучного інтелекту в хмарних сервісах. VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24 квітня 2025 р. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 363-365.

ПЕРЕЛІК ПОСИЛАНЬ

1. IBM. IaaS, PaaS, SaaS: What’s the Difference? IBM Cloud Documentation. 2023. URL: https://www.ibm.com/think/topics/iaas-paas-saas (date of access: 20.02.2025).
1. Amazon Web Services. AWS Auto Scaling Documentation. 2024. URL: https://docs.aws.amazon.com/autoscaling/ (date of access: 20.10.2025).
2. Amazon Web Services. Predictive Scaling for Amazon EC2 Auto Scaling. 2024. URL: https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-autoscaling-predictive-scaling.html (date of access: 22.10.2025).
3. Amazon Web Services. Predictive Scaling for Application Auto Scaling. 2024. URL: https://docs.aws.amazon.com/autoscaling/application/userguide/applicationauto-scaling-predictive-scaling.html (date of access: 19.10.2025).
4. Amazon Web Services. Use Historical Patterns to Scale Amazon ECS Services with Predictive Scaling. 2024. URL: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/predictiveauto-scaling.html (date of access: 29.09.2025).
5. Microsoft. Autoscale in Azure Monitor. Azure Monitor Documentation. 2024. URL: https://learn.microsoft.com/en-us/azure/azure-monitor/ (date of access: 01.10.2025).
6. Microsoft. Overview of Autoscale with Azure Virtual Machine Scale Sets. 2025. URL: https://learn.microsoft.com/en-us/azure/virtual-machine-scalesets/virtual-machine-scale-sets-autoscale-overview?tabs=portal- (date of access: 04.10.2025).
7. Google Cloud. Autoscaling Groups of Instances (Compute Engine Autoscaler Overview). 2024. URL: https://docs.cloud.google.com/compute/docs/autoscaler (date of access: 06.10.2025).

8. Google Cloud. Understanding Autoscaler Decisions and Predictive Policies. 2025. URL: https://docs.cloud.google.com/compute/docs/autoscaler/understandingautoscaler-decisions (date of access: 20.02.2025).
9. Mell P., Grance T. The NIST Definition of Cloud Computing. NIST Special Publication 800-145. Gaithersburg: National Institute of Standards and Technology, 2011. 7 p.
10. Erl T., Puttini R., Mahmood Z. Cloud Computing: Concepts, Technology & Architecture. Boston: Pearson Education, 2013. 528 p.
11. LeanIX. IaaS vs. PaaS vs. SaaS – Differences, Examples and Diagram. LeanIX Knowledge Base. 2022. URL: https://www.leanix.net (date of access: 20.02.2025).
12. Comparative Analysis of SaaS, IaaS, and PaaS – Exploring the Cloud Computing Paradigm. International Journal of Innovative Research in Computer Science & Technology. 2021. Vol. 9, No. 3. P. 45–55.
13. Feng B., Ding Z. Application-Oriented Cloud Workload Prediction: A Survey and New Perspectives. 2024. 42 p.
14. Hyndman R., Athanasopoulos G. Forecasting: Principles and Practice. 3rd ed. Melbourne: OTexts, 2021. 420 p.
15. Taylor S. J., Letham B. Forecasting at Scale. The American Statistician. 2018. Vol. 72(1). P. 37–45.
16. Ahmad J. A Time Series Analysis Using Facebook Prophet Model. Technological Forecasting & Social Change. 2025. Vol. 198. 118–132.
17. Saxena D., Singh A. K. Workload Forecasting and Resource Management Models Based on Machine Learning for Cloud Computing Environments. 2021. 15 p.
18. ML-driven Resource Management in Cloud Computing. World Journal of Advanced Research and Reviews. 2022. Vol. 14, No. 3. P. 120–132.

19. Nguyen T. et al. An LSTM-based Approach for Predicting Resource Usage in Cloud Computing. In: Proceedings of the International Conference on Future Internet of Things and Cloud. 2022. P. 118–125.
20. Nashold L. et al. Using LSTM and SARIMA Models to Forecast Cluster CPU Utilization. 2020. 12 p.
21. Alharthi S. et al. Auto-Scaling Techniques in Cloud Computing: Issues and Directions. Sensors. 2024. Vol. 24(6). P. 1–22.
22. Villano U. et al. A Survey on Auto-Scaling: How to Exploit Cloud Elasticity. International Journal of Grid and Utility Computing. 2022. Vol. 13, No. 4. P. 421–435.
23. Lemke O. et al. Survey on Machine Learning-based Autoscaling in Cloud Computing. Technical Report. Technical University of Munich, 2022. 68 p.
24. Duc T. L. et al. Workload Prediction for Proactive Resource Allocation in Cloud Data Centers. Electronics. 2025. Vol. 14. Article 1520. 18 p.
25. Sabyasachi A. S. et al. Deep CNN and LSTM Approaches for Efficient Workload Prediction in Cloud Environments. Procedia Computer Science. 2024. Vol. 218. P. 1475–1484.
26. Zhang H. et al. Cloud Computing Load Prediction Method Based on CNN–BiLSTM. Scientific Reports. 2024. Vol. 14. Article No. 21848.
27. Kidane L. et al. Workload Prediction in Cloud Data Centers Using Spatiotemporal Graph Convolutional Networks. Transactions on Emerging Telecommunications Technologies. 2025. Vol. 36. 15 p.
28. Essien A. et al. Machine Learning-based Workload Prediction for Autoscaling Cloud Applications. 2022. 10 p.
29. Marie-Magdelaine N. et al. Proactive Autoscaling for Cloud-Native Applications Using LSTM-Based Workload Prediction. In: IEEE GLOBECOM 2020. P. 1–6.
30. Pintye I. et al. Enhancing Machine Learning-Based Autoscaling for Cloud Applications. Journal of Grid Computing. 2024. Vol. 22. P. 1–17.

31. Li L. et al. Profit-Efficient Elastic Allocation of Cloud Resources Using an Adaptive Two-Stage Multi-Neural Network Based on LSTM. Applied Sciences. 2025. Vol. 15(2). 22 p.
32. Shen H. et al. Machine Learning Based Workload Prediction in Cloud Computing. In: ICCCN 2020 – 29th International Conference on Computer Communications and Networks. 2020. P. 1–8.
33. Cloud Resource Forecasting Using LSTM. NeuroQuantology. 2025. Vol. 23. P. 112–124.
34. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge: MIT Press, 2016. 800 p.
35. Bishop C. Pattern Recognition and Machine Learning. New York: Springer, 2006. 738 p.
36. Resource Utilization Prediction Model for Cloud Datacentres. International Journal of Advanced Computer Science and Applications. 2025. Vol. 16(1). P. 35–48.
37. Workload Prediction in Cloud Data Centers Using Complex Spatiotemporal Graph Convolutional Networks. 2025. 14 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Метод імплементації алгоритмів штучного інтелекту для підвищення функціональності та продуктивності хмарних технологій»

Виконав: студент групи ПДМ-62 Іван ПОЛТАВСЬКИЙ

Керівник: канд. техн. наук, доцент, доцент кафедри ІПЗ Владислав ЯСКЕВИЧ

Київ - 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення функціональності та продуктивності хмарних технологій за рахунок інтеграції алгоритмів штучного інтелекту для прогнозування навантаження.

Об'єкт дослідження: процес прогнозування навантаження у хмарних технологіях.

Предмет дослідження: алгоритми штучного інтелекту на базі нейронних мереж, для прогнозування навантаження у хмарних технологіях в умовах динамічних та неоднорідних режимів роботи.

ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА МЕТОДІВ

Модель / Метод	Переваги	Недоліки
SARIMA (статистична модель)	• Найкраще працює на даних із чіткою сезонністю. • Добре відтворює повторювані цикли. • Дає стабільні результати на гладких часових рядах. • Легко інтерпретується, має чіткі статистичні основи.	• Суттєво втрачає точність при стохастичних або змішаних режимах. • Не здатна реагувати на різкі піки або структурні зміни даних. • Не масштабована на складні багатовимірні дані.
Random Forest (Machine Learning)	• Добре працює на стохастичних та змішаних рядах. • Вміє враховувати велику кількість лагових ознак. • Гнучка модель, адаптивна до шуму. • Не потребує сезонності чи стаціонарності даних.	• Гірше відтворює довготривалі цикли. • Не пристосована до різких піків - часто згладжує їх. • Потребує наявність лагових ознак для роботи.
LSTM (Deep Learning)	• Найкраще відображає складні, нелінійні та хаотичні ряди. • Здатна передбачати різкі пікові навантаження. • Має "пам'ять" для довготривалих залежностей. • Найточніша на змішаних і пікових режимах.	• Дуже чутлива до якості та чистоти даних. • Вимагає масштабування та ретельної підготовки вибірки. • Значні обчислювальні витрати, повільне навчання.
Розроблений адаптивний метод	• Враховує не тільки середню помилку моделі, але й відповідність типу навантаження (ймовірності від класифікатора). • Універсальний: добре працює в сезонних, стохастичних, пікових та змішаних режимах. • Автоматично підлаштовується до структури ряду в реальному часі. • Усуває ситуації, коли «неправильна» модель застосовується до «неправильного» типу навантаження.	• Складніший у реалізації, ніж використання однієї моделі. • Вимагає наявності та періодичного оновлення класифікатора. • Потребує більше ресурсів для одночасного запуску кількох моделей. • Якість залежить від коректності побудованої системи ознак для класифікації.

3

ВИЗНАЧЕННЯ ВИМОГ ДЛЯ МЕТОДУ

Проблема	Вимоги до розробленого методу
У хмарних системах навантаження буває сезонним, стохастичним, піковим, змішаним, і вони поводяться по-різному.	Метод повинен **автоматично визначати тип навантаження** на основі ознак часового ряду.
Окремі моделі (SARIMA, Random Forest, LSTM) добре працюють тільки для своїх типів навантаження.	Метод повинен **запускати кілька моделей прогнозування** для одного і того ж інтервалу даних.
Використання не оптимальних моделей для певних типів навантаження	Метод повинен **оцінювати якість кожної моделі за метриками похибок** (MAE, RMSE тощо).
Класифікатор дає лише ймовірність типу навантаження, а моделі дають лише помилки — їх треба погодити між собою.	Метод повинен **порівнювати точність моделей і ймовірність типу навантаження** для прийняття рішення.
Хмарна система має працювати без ручного налаштування під кожен сценарій.	Метод повинен **автоматично обирати оптимальну модель прогнозування для поточного режиму роботи**.

4

МАТЕМАТИЧНА МОДЕЛЬ МЕТОДУ

Формула роботи AI-класифікатора

$$C_t = \arg\max_c P\,(C = c|\, X_t)$$

де C_t - визначений тип навантаження у момент часу t (seasonal / stochastic / peaks / mixed),

X_t - набір ознак, переданих у класифікатор,

$P(C = c|\, X_t)$ - ймовірність того, що ряд належить до класу c.

Нормалізована якість моделі

$$q_j = \frac{1/E_j}{\sum k\left(\frac{1}{E_k}\right)}$$

де E_j - MAE моделі *j,*
q_j - нормалізована вага моделі за якістю
$\sum k\left(\frac{1}{E_k}\right)$ – сума обернених помилок усіх моделей

Інтегральний score моделі (остаточний вибір моделі)

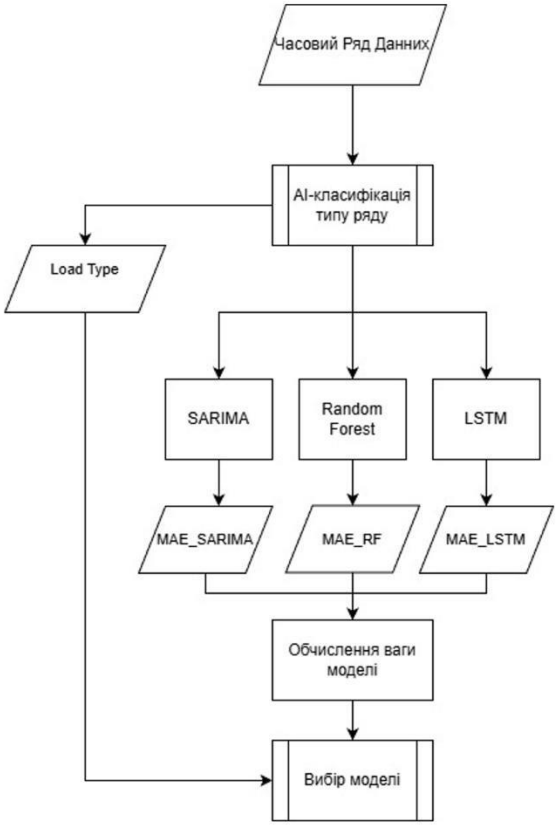
$$Scr_j = \alpha * q_j + \beta * p_j \qquad mod = \arg\max_j Scr_j$$

де Scr_j - оцінка моделі *j,*
q_j - нормалізована вага моделі за якістю,
p_j - ймовірність відповідності поточного типу навантаження до моделі *j,*
α *та* β – вагові коефіцієнти для ваги та відповідності моделі до навантаження
mod – вибрана модель

АРХІТЕКТУРА ВИКОРИСТАНИХ МОДЕЛЕЙ ПРОГНОЗУВАННЯ

6

СХЕМАТИЧНЕ ПРЕДСТАВЛЕННЯ РОБОТИ МЕТОДУ



7

АНАЛІЗ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО МЕТОДУ

Модель/Навантаження	Сезонне	Стохастичне	Пікове	Змішане
SARIMA	MAE: 4.009 RMSE: 5.038	MAE: 14.312 RMSE: 17.919	MAE: 6.757 RMSE: 11.237	MAE: 5.767 RMSE: 6.597
Random Forest	MAE: 4.878 RMSE: 6.150	MAE: 10.459 RMSE: 12.598	MAE: 7.102 RMSE: 11.243	MAE: 5.588 RMSE: 7.260
LSTM	MAE: 4.015 RMSE: 5.187	MAE: 14.086 RMSE: 17.369	MAE: 6.282 RMSE: 8.952	MAE: 5.401 RMSE: 6.939
Адаптивний метод	MAE: 3.992 RMSE: 4.916	MAE: 10.021 RMSE: 12.307	MAE: 6.283 RMSE: 9.003	MAE: 5.404 RMSE: 6.881

$$MAE = \frac{1}{N}\sum |x_t - \hat{x}_t| \qquad RMSE = \sqrt{\frac{1}{N}\sum (x_t - \hat{x}_t)^2}$$

8

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Результат роботи методу для прогнозування пікового навантаження. Вибрана модель - LSTM

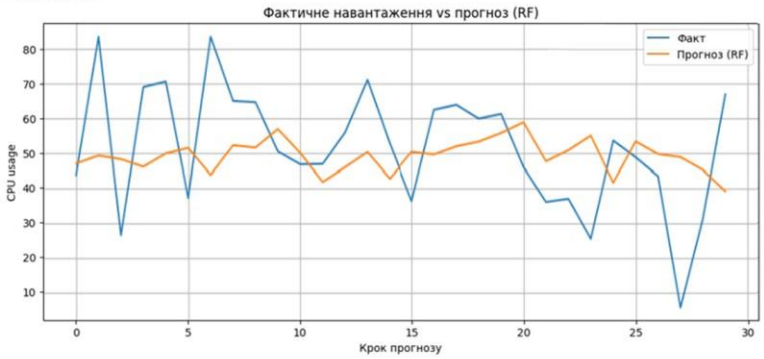
```
Тип навантаження (класифікатор): peaks
Ймовірності класів (P(C=c | X_t)):
  mixed: 0.300
  peaks: 0.687
  seasonal: 0.010
  stochastic: 0.003

Запуск моделей прогнозування...
MAE моделей:
  SARIMA: 6.758
  RF:     6.482
  LSTM:   6.283

Ваги за якістю (нормалізоване 1/MAE):
  SARIMA: 0.321
  RF: 0.334
  LSTM: 0.345

Ваги за типом навантаження (ймовірності моделі):
  SARIMA: 0.010
  RF: 0.003
  LSTM: 0.987

Підсумковий score (alpha=0.5, beta=0.5):
  SARIMA: 0.165
  RF: 0.169
  LSTM: 0.666

Обрана модель: LSTM
```

9

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Результат роботи методу для прогнозування для стохастичного навантаження. Вибрана модель - RandomForest

```
Тип навантаження (класифікатор): stochastic
Ймовірності класів (P(C=c | X_t)):
  mixed: 0.000
  peaks: 0.017
  seasonal: 0.000
  stochastic: 0.983

Запуск моделей прогнозування...

MAE моделей:
  SARIMA: 14.310
  RF:     10.021
  LSTM:   14.087

Ваги за якістю (нормалізоване 1/MAE):
  SARIMA: 0.337
  RF: 0.321
  LSTM: 0.342

Ваги за типом навантаження (ймовірності моделі):
  SARIMA: 0.000
  RF: 0.983
  LSTM: 0.017

Підсумковий score (alpha=0.5, beta=0.5):
  SARIMA: 0.168
  RF: 0.652
  LSTM: 0.179

Обрана модель: RF
```

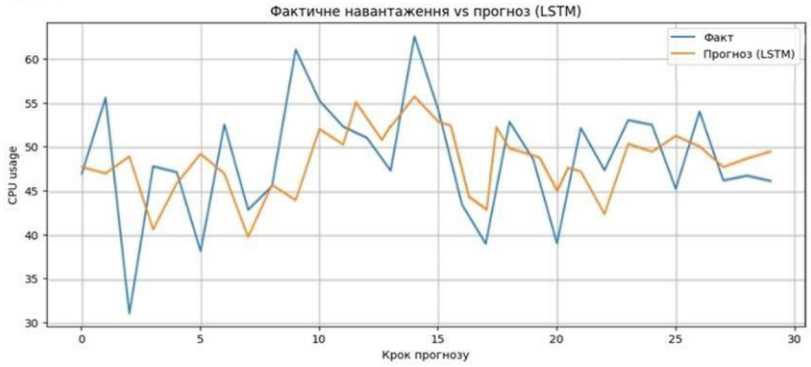



```
Графік побудовано.
```

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Результат роботи методу для прогнозування змішаного навантаження. Вибрана модель - LSTM

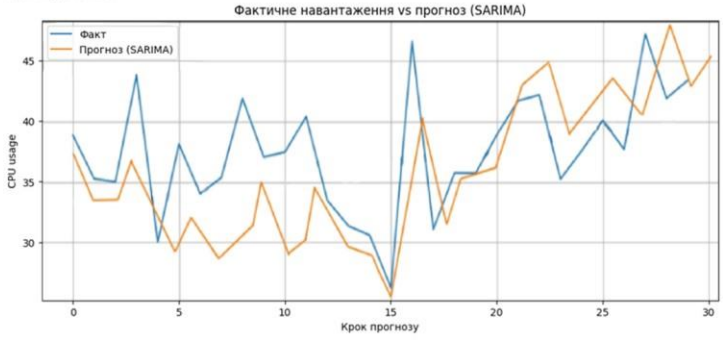
```
Тип навантаження (класифікатор): mixed
Ймовірності класів (P(C=c | X_t)):
  mixed: 0.640
  peaks: 0.163
  seasonal: 0.190
  stochastic: 0.007

Запуск моделей прогнозування...
MAE моделей:
  SARIMA: 5.168
  RF:     6.015
  LSTM:   5.404

Ваги за якістю (нормалізоване 1/MAE):
  SARIMA: 0.355
  RF: 0.305
  LSTM: 0.340

Ваги за типом навантаження (ймовірності моделі):
  SARIMA: 0.190
  RF: 0.007
  LSTM: 0.803

Підсумковий score (alpha=0.5, beta=0.5):
  SARIMA: 0.273
  RF: 0.156
  LSTM: 0.571

Обрана модель: LSTM
```



ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Результат роботи методу для прогнозування для сезонного навантаження. Вибрана модель - SARIMA

```
Тип навантаження (класифікатор): seasonal
Ймовірності класів (P(C=c | X_t)):
  mixed: 0.013
  peaks: 0.010
  seasonal: 0.963
  stochastic: 0.013

Запуск моделей прогнозування...
MAE моделей:
  SARIMA: 3.992
  RF:     4.748
  LSTM:   4.619

Ваги за якістю (нормалізоване 1/MAE):
  SARIMA: 0.370
  RF: 0.311
  LSTM: 0.320

Ваги за типом навантаження (ймовірності моделі):
  SARIMA: 0.963
  RF: 0.013
  LSTM: 0.023

Підсумковий score (alpha=0.5, beta=0.5):
  SARIMA: 0.666
  RF: 0.162
  LSTM: 0.171

Обрана модель: SARIMA
```



ВИСНОВКИ

1. Проведено аналіз сучасних моделей прогнозування часових рядів, включно зі статистичними методами (SARIMA), класичними алгоритмами машинного навчання (Random Forest) та глибинними нейронними мережами (LSTM). Визначено їхні переваги, обмеження та доцільність застосування для різних типів хмарних навантажень.
2. Створено модуль AI-класифікації, який на основі ознак часових рядів автоматично визначає тип навантаження (seasonal, stochastic, peaks, mixed) та формує ймовірнісну оцінку відповідності кожного класу. Це дозволяє адаптувати процес прогнозування до поточних умов роботи системи.
3. Розроблено адаптивний метод прогнозування навантаження у хмарних сервісах, який поєднує статистичні моделі, алгоритми машинного навчання та глибинні нейронні мережі. Метод забезпечує вибір найбільш ефективної моделі прогнозування залежно від структури часового ряду.
4. Реалізовано механізм інтелектуального вибору моделі, який поєднує точність прогнозування (MAE) та ймовірність відповідності моделі визначеному типу навантаження. Запропонований метод забезпечує автоматичне обрання оптимального прогнозуючого алгоритму для кожного конкретного сценарію.
5. Підтверджено ефективність запропонованого методу, який демонструє високу стабільність і точність прогнозування у сезонних, стохастичних, пікових та змішаних режимах. Метод перевершує окремі моделі завдяки адаптивному вибору та здатності працювати в умовах змінної структури навантаження. Отримані результати прогнозування знаходяться в діапазонах, характерних для точних моделей часових рядів: середня абсолютна помилка (MAE) у межах від ≈4 до ≈10 одиниць навантаження, а середньоквадратична помилка (RMSE) — від ≈5 до ≈12 одиниць залежно від типу навантаження. Ці значення підтверджують, що метод забезпечує рівень точності, порівнянний або вищий за базові моделі (SARIMA, Random Forest, LSTM), зокрема у сезонних та стохастичних режимах.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Полтавський І А., Яскевич В.О. Інтеграція сервісів ШІ в хмарну інфраструктуру: архітектурні підходи та кейси застосування. V Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15 травня 2025 р. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. с. 152-153.

2. Полтавський І А., Яскевич В.О. Дослідження можливостей застосування штучного інтелекту в хмарних сервісах. VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційнокомунікаційних технологіях». 24 квітня 2025 р. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. с. 363-365.