Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
Кафедра Комп'ютерних інтелектуальних технологій та систем

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Інтеграція даних у хмарних обчисленнях з використанням нейромережевого підходу

(тема)

Виконав:
здобувач II курсу, групи КІТм-23-1
Софія СЕРДЮК
(власне ім'я, прізвище)

Спеціальність 123 Комп’ютерна інженерія
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерні інтелектуальні технології
(повна назва освітньої програми)

Керівник проф. каф. КІТС Олег РУДЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри Олег РУДЕНКО

(підпис) (власне ім'я, прізвище)

2024 р.

Харківський національний університет радіоелектроніки

Факультет	Комп'ютерної інженерії та управління
Кафедра	Комп'ютерних інтелектуальних технологій та систем
Рівень вищої освіти	другий (магістерський)
Спеціальність	123 Комп'ютерна інженерія
Тип програми	освітньо-професійна
Освітня програма	Комп'ютерні інтелектуальні технології

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____2024р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Сердюк Софії Сергіївні

1. Тема роботи (проекту) Інтеграція даних у хмарних обчисленнях з використанням нейромережевого підходу
затверджена наказом університету від "28" жовтня 2024 р. № 1256 Ст
2. Термін подання учнем роботи до екзаменаційної комісії 20.01.2024р.
3. Вихідні дані до роботи (проекту)
1) існуючих підходів автоматичного налаштування обчислювальних ресурсів в хмарних системах
2) методи автоскейлінгу, оркестрації контейнерів, моніторингу та логування
3) опис сервісів, використаних у проекті
4. Перелік питань, що потрібно опрацювати в роботі:
Огляд стану проблеми та постановка задачі
Аналіз літератури за напрямком дослідження
Огляд сучасних методів та засобів автоматичного налаштування обчислювальних ресурсів
Огляд сучасних методів моніторингу і аналізу метрик продуктивності
Розробка рішень оптимізації використання ресурсів у динамічних середовищах
Впровадження запропонованих рішень в реальному середовищі
Підготовка презентаційного матеріалу
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням кафедри)
16 слайдів презентаційного матеріалу
_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявністю консультантів згідно до наказу, зазначеному у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Видача та узгодження теми проєкту	28.10.2024	Виконано
2	Огляд стану проблеми та постановка задачі	30.11-29.11.2024	Виконано
3	Аналіз літератури за напрямком дослідження	30.11- 07.12.2024	Виконано
4	Аналіз існуючих підходів автоматичного налаштування обчислювальних ресурсів в хмарних системах	08.12-10.12.2024	Виконано
5	Розробка рішень оптимізації використання ресурсів у динамічних середовищах	11.12-16.12.2024	Виконано
6	Впровадження запропонованих рішень	17.12-24.12.2024	Виконано
7	Оцінка ефективності запропонованих рішень у вирішенні задач автоматизації в хмарних обчисленнях.	25.12-29.12.2024	Виконано
8	Підготовка ПЗ та презентаційного матеріалу	30.12-23.01.2025	Виконано
9	Подання до ЕК	20.01.2025	Виконано
10	Захист проєкту	22.01.2025	

Дата видачі завдання «28» жовтня 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ професор О.Г.Руденко
(підпис) (посада, ініціали,прізвище)

РЕФЕРАТ

Загальний обсяг роботи: XX с., 10 рисунків, 1 таблиця, 19 джерел.

ІНТЕГРАЦІЯ, ХМАРНІ ОБЧИСЛЕННЯ, АВТОСКЕЙЛІНГ, МОНІТОРИНГ, ЛОГУВАННЯ, МЕТРИКИ, ОРКЕСТРАЦІЯ

Використання нейромережевих підходів для автоматизації процесів в хмарних середовищах дозволяє забезпечити більш ефективне управління IT-інфраструктурою. Це включає в себе автоскейлінг, який дозволяє автоматично збільшувати або зменшувати обчислювальні ресурси в залежності від поточного рівня навантаження, що підтримує стабільну роботу додатків і оптимізує використання ресурсів. Автоматизація також включає моніторинг і логування, де автоматизовані системи відстежують стан додатків та інфраструктури в режимі реального часу, збирають і аналізують дані про продуктивність, виявляють аномалії і автоматично реагують на інциденти, забезпечуючи безперебійну роботу системи.

Таким чином, метою роботи є дослідження методів автоматизації для ефективного управління IT-інфраструктурою, зокрема, шляхом автоскейлінгу, оркестрації контейнерів, моніторингу та логування, а також роботизованої автоматизації процесів (RPA). Використання цих методів забезпечить стабільну роботу додатків, оптимізацію використання ресурсів, підвищення надійності сервісів та зменшення кількості рутинних завдань для співробітників. Це спрямовано на комплексне дослідження та впровадження методів автоматизації, що дозволить підвищити ефективність управління IT-інфраструктурою та оптимізувати бізнес-процеси.

ABSTRACT

Coursework: XX pages, 10 figures, 1 tables, 6 formulas, 19 sources.

INTEGRATION, CLOUD COMPUTING, AUTOSCALING, MONITORING, LOGGING, METRICS, ORCHESTRATION

Using neural network approaches for automating processes in cloud environments allows for more efficient IT infrastructure management. This includes autoscaling, which automatically increases or decreases computing resources depending on the current load, maintaining stable application performance and optimizing resource usage. Automation also encompasses monitoring and logging, where automated systems track the status of applications and infrastructure in real time, collecting and analyzing performance data, detecting anomalies, and automatically responding to incidents to ensure the system operates without interruption.

Thus, the aim of this work is to investigate automation methods for effective IT infrastructure management, particularly through autoscaling, container orchestration, monitoring and logging, as well as robotic process automation (RPA). The use of these methods will ensure stable application performance, resource optimization, increased service reliability, and reduced routine tasks for employees. This is directed towards comprehensive research and implementation of automation methods that will enhance IT infrastructure management efficiency and optimize business processes.

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
Кафедра Комп'ютерних інтелектуальних технологій та систем

АНОТАЦІЯ

КВАЛІФІКАЦІЙНОЇ РОБОТИ

рівень вищої освіти другий (магістерський)

Інтеграція даних у хмарних обчисленнях з використанням нейромережевого підходу

(тема)

Виконав:
здобувач II курсу, групи КІТм-23-1
Софія СЕРДЮК
(власне ім'я, прізвище)
Спеціальність 123 Комп’ютерна інженерія
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Комп'ютерні інтелектуальні технології
(повна назва освітньої програми)
Керівник проф. каф. КІТС Олег РУДЕНКО
(посада, власне ім'я, прізвище)

2024

АНОТАЦІЯ

Сердюк С. С. Інтеграція даних у хмарних обчисленнях з використанням нейромережевого підходу. – Магістерська кваліфікаційна робота.

Стрімкий розвиток технологій штучного інтелекту (ШІ) та нейронних мереж створив передумови для впровадження інноваційних рішень у сфері хмарних обчислень. Ці технології не лише оптимізують обробку даних, але й дозволяють автоматизувати численні процеси в IT-інфраструктурі, забезпечуючи зменшення витрат і підвищення надійності сервісів. Завдяки нейронним мережам хмарні системи отримують здатність аналізувати величезні обсяги інформації, знаходити приховані закономірності та швидко адаптуватися до змін у бізнес-середовищі. Це відкриває нові можливості для побудови гнучких архітектур, які ефективно реагують на зростання навантаження, збої та зміни у вимогах користувачів.

Мета роботи полягала у дослідженні та розробці методів автоматизації для ефективного управління обчислювальними ресурсами, що є ключовим завданням сучасних IT-систем. Зокрема, робота була спрямована на оптимізацію використання ресурсів у динамічних середовищах, де зміни в навантаженні можуть відбуватися раптово й непередбачувано. Ефективне управління ресурсами є критично важливим для забезпечення стабільної роботи систем, мінімізації витрат і підвищення продуктивності.

У першому розділі проведений аналіз предметної області, розглянуті основні підходи до автоматизації управління обчислювальними ресурсами, включаючи автоскейлінг, оркестрацію контейнерів, моніторинг і логування. Зроблений акцент на важливості використання нейромереж для адаптації систем до змін у навантаженнях та покращення їхньої продуктивності.

Другий розділ присвячений проблемам, пов'язаним із налаштуванням і управлінням хмарними середовищами. Детально розглянуто проблеми конфігурації, оновлень програмного забезпечення та прогнозування

навантажень. Запропоновано підхід до використання нейромереж для вирішення цих проблем, із врахуванням особливостей динамічних змін у роботі систем.

У третьому розділі наведено аналіз типів нейронних мереж, придатних для задач інтеграції даних і управління обчислювальними ресурсами. Обґрунтовано вибір архітектури для вирішення поставлених задач, визначено параметри навчання моделі та застосовані алгоритми оптимізації. Розглянуто перспективи використання отриманих результатів для створення адаптивних і масштабованих систем.

Четвертий розділ описує практичну реалізацію розробленої нейромережевої моделі. Наведено опис алгоритмів навчання, процесу обробки даних та оцінки якості моделі. Представлені результати тестування, які підтверджують ефективність моделі у вирішенні задач автоматизації в хмарних обчисленнях.

Отримані результати роботи демонструють потенціал використання нейромереж для автоматизації процесів управління обчислювальними ресурсами, підвищення надійності та продуктивності систем. Запропоновані рішення можуть бути корисними для розробки інтелектуальних систем у різних галузях, включаючи фінанси, медицину, промисловість та інші сфери, що потребують роботи з великими обсягами даних.

ІНТЕГРАЦІЯ ДАНИХ У ХМАРНИХ ОБЧИСЛЕННЯХ З ВИКОРИСТАННЯМ НЕЙРОМЕРЕЖЕВОГО ПІДХОДУ

Публікації здобувача за темою роботи:

1. Ilyunin O., Khodak M., Pyrohov V., Pasholoc O., Tryhuba M., Serdiuk S. Intelligent models for control of jet hydro-processing of rolled steel defects // Інтегровані технології та енергозбереження. Системи управління та обробки

інформації. – Харків: НТУ «ХПІ», 2023. – №2. – С. 45–56. – ISSN 2078-5364 (print), ISSN 2708-0625 (online).

2. Serdiuk S. Integration of big data processing using neural networks in Cloud computing // XVI Всеукраїнська науково-практична WEB-конференція аспірантів, студентів та молодих вчених «Комп’ютерні інтелектуальні системи та мережі» (КІСМ-2024), 26–28 березня 2024 р., Кривий Ріг, Україна. – С. 134–136.

3. Сердюк С.С. Інтеграція обробки великих даних за допомогою нейромереж у хмарні обчислення // 28-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті», т. 5, 16–18 квітня 2024 р., Харків, Україна. – С. 66–68.

ЗМІСТ

ВСТУП

У сучасному технологічному світі, де цифрові перетворення стали невід'ємною частиною роботи бізнесу, хмарні обчислення зайняли центральне місце в управлінні інфраструктурою, ресурсами та додатками. Вони надають підприємствам можливість зменшити витрати на створення й підтримку фізичної інфраструктури, забезпечують високу доступність і масштабованість програмного забезпечення, а також значно спрощують управління IT-системами. Від корпоративних додатків до особистих сервісів — хмарні технології стали фундаментом для продуктивного ведення справ, стимулюючи розвиток інновацій та адаптацію до нових умов ринку.

Водночас автоматизація процесів стала невід'ємним компонентом, який дозволяє компаніям зосереджуватись на стратегічних завданнях замість виконання рутинних операцій [1]. Завдяки таким технологіям, як роботизована автоматизація процесів (RPA), автоскейлінг, оркестрація контейнерів, моніторинг і логування, підприємства отримують змогу ефективніше використовувати свої ресурси, підвищувати надійність систем та оптимізувати робочі процеси.

Зі стрімким розвитком технологій штучного інтелекту й нейронних мереж відкриваються нові можливості для їх інтеграції у хмарні обчислення, що дозволяє вирішувати складні завдання обробки, аналізу та управління даними. Нейронні мережі, завдяки своїй здатності до навчання й адаптації, стають потужним інструментом для роботи з великими обсягами інформації. Вони дозволяють автоматизувати процеси аналізу, виявляти приховані закономірності у даних і значно підвищувати точність прогнозів. Це стає критично важливим у сучасних умовах, коли обсяги даних зростають експоненційно, а швидкість прийняття рішень може визначати конкурентоспроможність компаній.

Застосування нейромережевих підходів у хмарному середовищі відкриває широкі перспективи для створення адаптивних систем, здатних ефективно реагувати на динамічні зміни середовища [2] Це забезпечує не лише оперативність та точність обробки інформації, а й можливість автоматично оптимізувати використання ресурсів, зменшуючи витрати на інфраструктуру. Крім того, інтеграція штучного інтелекту й нейронних мереж у хмарні обчислення сприяє підвищенню доступності даних для користувачів, забезпечує їхню консистентність і захищеність навіть у складних багатокористувацьких середовищах.

У контексті бізнесу та наукових досліджень нейромережеві технології дозволяють створювати рішення, які раніше здавалися неможливими. Наприклад, інтелектуальні системи управління даними можуть в режимі реального часу аналізувати поведінку користувачів, прогнозувати попит, автоматично адаптувати сервіси до поточних потреб клієнтів. У хмарному середовищі це забезпечує гнучкість у роботі з даними, підтримку високої продуктивності сервісів і гарантію надійності їх функціонування навіть за умов значного навантаження [3].

Загалом, використання нейромереж у хмарних обчисленнях не лише покращує ефективність інтеграції й обробки даних, а й формує основу для інноваційних рішень у багатьох галузях — від фінансових технологій до охорони здоров’я, від освіти до енергетики. Це робить нейромережеві підходи невід'ємною частиною сучасного технологічного ландшафту.

Мета даного дослідження полягає у розробці методів і технологій інтеграції даних у хмарних обчисленнях із використанням нейромережевого підходу. Робота включає аналіз сучасних технологій, дослідження методів автоматизації управління IT-інфраструктурою та створення рішень для ефективної інтеграції даних у хмарному середовищі. Це сприятиме не лише оптимізації бізнес-процесів і зменшенню витрат, а й покращенню якості сервісів, стабільності роботи програм і підвищенню загальної конкурентоспроможності підприємств.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Основні поняття автоматизації IT-структури

Автоматизація — це процес використання технологій для виконання завдань з мінімальним втручанням людини. В сучасних інформаційних технологіях автоматизація є ключовим елементом, що забезпечує підвищення ефективності, зниження витрат та покращення якості послуг і продуктів. Вона проявляється в багатьох аспектах і охоплює широкий спектр процесів і систем.

Автоматизація передбачає впровадження систем і інструментів, які дозволяють виконувати рутинні та повторювані завдання без постійного втручання людини. Це досягається за допомогою програмного забезпечення, робототехніки, штучного інтелекту та інших технологій, які можуть керувати процесами, аналізувати дані і приймати рішення на основі визначених правил і алгоритмів.

Одним з основних проявів автоматизації є автоматизація IT-інфраструктури. Це включає в себе автоскейлінг, який дозволяє автоматично збільшувати або зменшувати обчислювальні ресурси (наприклад, сервери) в залежності від поточного рівня навантаження. Завдяки цьому можна підтримувати стабільну роботу додатків і оптимізувати використання ресурсів. Оркестрація контейнерів, що передбачає управління і координацію контейнеризованих додатків за допомогою таких інструментів, як Kubernetes, дозволяє автоматизувати розгортання, масштабування та управління додатками.

Автоматизація також проявляється в моніторингу і логуванні, де автоматизовані системи відстежують стан додатків та інфраструктури в режимі реального часу. Вони збирають і аналізують дані про продуктивність, виявляють аномалії і автоматично реагують на інциденти, щоб забезпечити

безперебійну роботу системи. Це дозволяє швидко виявляти та вирішувати проблеми, зменшуючи час простою і підвищуючи надійність сервісів [1, 4].

В бізнес-процесах автоматизація проявляється у вигляді роботизованої автоматизації процесів (RPA), яка використовує програмних роботів для виконання завдань, таких як обробка даних, введення інформації та управління робочими процесами. Це допомагає зменшити кількість помилок, прискорити виконання завдань і звільнити співробітників від рутинної роботи, дозволяючи їм зосередитися на більш складних і творчих завданнях.

Таким чином, автоматизація є важливим інструментом у сучасному світі, який дозволяє підвищити ефективність, знизити витрати і покращити якість послуг та продуктів. Вона забезпечує швидке і точне виконання завдань, мінімізуючи людський фактор і підвищуючи надійність і продуктивність систем.

Завдяки автоматизації рутинних задач та управління ресурсами, компанії можуть зосередитися на основних бізнес-цілях, забезпечуючи при цьому високу якість послуг і продуктів [1, 4]. Автоматизовані системи управління, автоскейлінг, оркестрація контейнерів та інші технології дозволяють швидко реагувати на зміни навантаження і оптимізувати використання ресурсів.

Незважаючи на всі переваги хмарних технологій і автоматизації, вони не позбавлені недоліків. Складність сучасних IT-систем і високий рівень інтеграції різноманітних сервісів можуть призводити до виникнення проблем і збоїв. Автоматизовані процеси, хоча й зменшують людський фактор, але також можуть виходити з ладу через помилки в конфігураціях, оновленнях програмного забезпечення або непередбачуваних навантажень.

1.2 Основні проблеми автоматизації навантажень у хмарних середовищах

Однією з основних проблем є складність конфігурації і управління хмарними середовищами. Хмарні платформи надають широкий спектр сервісів і можливостей, але кожен з них потребує правильного налаштування та

інтеграції. Неправильна конфігурація може призвести до помилок, які важко виявити та виправити. Наприклад, неправильно налаштовані правила безпеки можуть відкрити доступ до конфіденційних даних, що створює серйозні ризики для безпеки.

Оновлення програмного забезпечення є ще однією областю, де можуть виникати проблеми. Регулярні оновлення необхідні для підтримки безпеки та продуктивності системи, але вони можуть викликати збої в роботі автоматизованих процесів. Наприклад, зміни в API або інтерфейсах служб можуть призвести до несподіваних збоїв у роботі додатків, які залежать від цих служб. Це може викликати простої, втрату даних або інші негативні наслідки.

Непередбачувані навантаження є ще однією серйозною проблемою. Хмарні середовища зазвичай добре справляються з масштабуванням для обробки високих навантажень, але раптові сплески трафіку можуть перевищити можливості автоматизованих систем. Наприклад, під час великих маркетингових кампаній або новорічних розпродажів може спостерігатися різке зростання числа користувачів, що може призвести до перевантаження серверів і зниження продуктивності.

Наслідки збоїв у хмарних і автоматизованих системах можуть бути серйозними. По-перше, це може призвести до простоїв, що негативно впливає на користувачів і може спричинити фінансові втрати. Навіть короткі простої можуть коштувати компаніям значних сум, особливо в галузях, де час є критичним фактором. По-друге, збої можуть призвести до втрати даних, що може мати катастрофічні наслідки для бізнесу. Втрата важливої інформації може призвести до втрати клієнтів, репутаційних ризиків і юридичних наслідків [5].

Щоб мінімізувати ризики, пов'язані з автоматизацією і хмарними технологіями, важливо впроваджувати надійні механізми моніторингу і відновлення. Це включає постійний моніторинг стану систем, швидке виявлення і усунення проблем, а також наявність планів відновлення на випадок збоїв. Регулярні резервні копії, тестування оновлень у контрольованих

середовищах і використання стратегій поступового розгортання можуть допомогти знизити ризики і забезпечити стабільну роботу систем.

У зв'язку з цим, важливо мати надійні механізми моніторингу та відновлення для забезпечення безперебійної роботи систем. Розробка ефективних стратегій автоскейлінгу, регулярне тестування і оптимізація конфігурацій, а також швидке реагування на інциденти є критичними для підтримання стабільності і високої продуктивності додатків у хмарному середовищі.

Для забезпечення стабільної роботи додатків існують різні механізми та інструменти. Одним з основних є системи моніторингу, які дозволяють постійно відстежувати стан додатків і інфраструктури в режимі реального часу. Це включає збір даних про використання ресурсів, час відгуку, кількість запитів і інші ключові метрики. Системи моніторингу можуть автоматично виявляти аномалії і попереджати адміністраторів про можливі проблеми.

Регулярне тестування є ще одним важливим механізмом. Воно включає перевірку додатків і інфраструктури на наявність помилок і вразливостей, а також тестування на стійкість до навантажень. Це дозволяє виявляти і усувати проблеми до того, як вони вплинуть на користувачів. Тестування також допомагає переконатися, що система здатна витримувати пікові навантаження і правильно масштабуватися.

Оптимізація конфігурацій є критичною для забезпечення ефективного використання ресурсів. Це включає налаштування параметрів системи, таких як розмір і тип віртуальних машин, конфігурації мережі та балансування навантаження. Оптимізація дозволяє зменшити витрати і підвищити продуктивність, забезпечуючи при цьому високу надійність і доступність додатків [5-8].

Швидке реагування на інциденти є важливою складовою забезпечення стабільності системи. Це включає наявність планів відновлення на випадок збоїв, автоматичне переключення на резервні ресурси, перезапуск додатків і інші заходи для швидкого усунення проблем. Важливо мати інструменти для

швидкого виявлення і діагностики інцидентів, щоб мінімізувати час простою і вплив на користувачів.

Одним з найефективніших механізмів для забезпечення стабільності і продуктивності додатків є автоматичне масштабування (автоскейлінг). Більшість проблем, що виникають під час роботи додатків, пов'язані з недостатньою кількістю обчислювальних ресурсів. Автоскейлінг дозволяє автоматично збільшувати або зменшувати кількість ресурсів в залежності від поточного навантаження. Це забезпечує оптимальне використання ресурсів і стабільну роботу додатків навіть під час пікових навантажень.

Автоскейлінг базується на моніторингу ключових метрик продуктивності, таких як завантаження процесора, використання пам'яті, кількість запитів і час відгуку. Коли ці метрики перевищують встановлені порогові значення, система автоматично додає нові ресурси для обробки додаткового навантаження. Коли навантаження зменшується, надлишкові ресурси відключаються, що дозволяє знизити витрати [5-8].

Таким чином, автоматичне масштабування є найефективнішим інструментом для забезпечення стабільності і продуктивності додатків у хмарному середовищі. Воно дозволяє динамічно адаптуватися до змін навантаження, забезпечувати безперебійну роботу і оптимальне використання ресурсів. Впровадження автоскейлінгу допомагає вирішувати більшість проблем, пов'язаних з недостатньою потужністю, і забезпечувати високу якість обслуговування користувачів.

1.3 Постановка задачі

Автоскейлінг ґрунтується на моніторингу основних метрик продуктивності, таких як завантаженість процесора, використання оперативної пам'яті, кількість запитів та час відгуку. У разі перевищення встановлених порогових значень система автоматично залучає додаткові ресурси для обробки підвищеного навантаження. Зі зниженням навантаження надлишкові ресурси

автоматично вимикаються, що сприяє оптимізації витрат. Застосування алгоритмів машинного навчання та штучного інтелекту дозволяє системі самостійно аналізувати історичні дані, адаптуватися до змін у навантаженні та поведінці користувачів і покращувати ефективність своїх рішень у реальному часі.

Таким чином, метою роботи є дослідження методів автоматизації для ефективного управління IT-інфраструктурою, зокрема, шляхом автоскейлінгу, оркестрації контейнерів, моніторингу та логування, а також роботизованої автоматизації процесів. Використання цих методів забезпечить стабільну роботу додатків, оптимізацію використання ресурсів, підвищення надійності сервісів та зменшення кількості рутинних завдань для співробітників. Це спрямовано на комплексне дослідження та впровадження методів автоматизації, що дозволить підвищити ефективність управління IT-інфраструктурою та оптимізувати бізнес-процеси.

Для вирішення поставленої мети треба реалізувати наступні завдання:

- провести аналіз сучасних методів та технологій для автоматизації управління IT-інфраструктурою;

- запропонувати інноваційний підхід до інтеграції алгоритмів машинного навчання та штучного інтелекту для покращення автоматизації і адаптації до змін у навантаженні;

- провести моделювання та тестування запропонованих рішень у контрольованому середовищі для оцінки їх ефективності;

- оцінити вплив запропонованих методів на продуктивність, стабільність роботи додатків та оптимізацію витрат на IT-інфраструктуру.

2 МЕТОДИ ТА ІНСТРУМЕНТИ АВТОМАТИЧНОГО НАЛАШТУВАННЯ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ

2.1 Методи моніторингу і аналізу метрик продуктивності

Автоскейлінг — це інструмент, який автоматично налаштовує обчислювальні ресурси в залежності від поточного навантаження на систему. Цей процес є важливим для забезпечення стабільної роботи веб-приложень та інших сервісів, оскільки він дозволяє динамічно реагувати на зміни в кількості запитів чи трафіку, забезпечуючи оптимальне використання ресурсів і зменшення витрат.

Завдання автоскейлінгу полягає в тому, щоб забезпечити належний рівень продуктивності та доступності сервісу незалежно від коливань навантаження. Це досягається шляхом автоматичного додавання або видалення ресурсів (таких як віртуальні машини, контейнери, процесорні ядра тощо) відповідно до встановлених правил та політик. Наприклад, під час пікових навантажень автоскейлінг може збільшити кількість віртуальних машин, що обслуговують запити, а в періоди низького навантаження зменшити їх кількість для економії ресурсів.

Основна мета автоскейлінгу полягає в забезпеченні ефективної роботи веб-приложень при мінімізації витрат. Без автоскейлінгу адміністратори систем змушені або виділяти надмірну кількість ресурсів для обробки пікових навантажень (що призводить до значних витрат у періоди низького навантаження), або ризикувати, що під час пікових навантажень система не зможе впоратися з потоком запитів (що призводить до зниження якості обслуговування і втрати клієнтів).

Автоскейлінг працює за принципом моніторингу і аналізу метрик продуктивності.

Метрики — це кількісні показники, які використовуються для моніторингу та оцінки продуктивності систем, процесів або компонентів. Вони допомагають виявляти проблеми, контролювати продуктивність та забезпечувати досягнення поставлених цілей.

Метрики продуктивності відображають, наскільки ефективно система виконує свої функції. Вони можуть включати завантаження процесора (CPU), використання пам'яті (RAM), час відгуку (Response Time), пропускну здатність (Throughput) і кількість запитів (Request Count). Завантаження процесора показує процентне значення використання процесорних ресурсів, що дозволяє оцінити, наскільки система наближена до своїх обчислювальних меж. Використання пам'яті відображає обсяг пам'яті, який використовується додатком, що допомагає виявити потенційні проблеми з продуктивністю. Час відгуку показує, як швидко система відповідає на запити користувачів, що є критичним для забезпечення гарного користувацького досвіду. Пропускна здатність вимірює кількість запитів або операцій, які система може обробити за одиницю часу, а кількість запитів допомагає оцінити навантаження на систему.

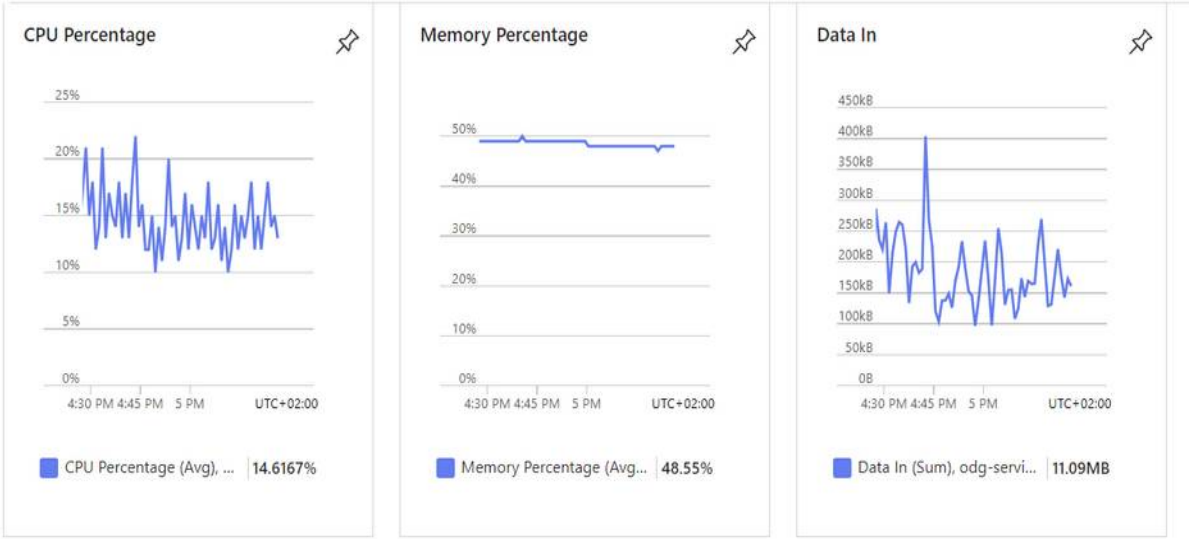

Рисунок 2.1 – Метрики продуктивності

Метрики надійності оцінюють здатність системи працювати без збоїв і включають час простою (Downtime), коефіцієнт готовності (Availability) та

частоту збоїв (Failure Rate). Час простою вказує на періоди, протягом яких система недоступна для користувачів, і менший час простою свідчить про високу надійність системи. Коефіцієнт готовності показує процентний час, протягом якого система доступна для користувачів, і високий коефіцієнт готовності вказує на надійність системи. Частота збоїв вимірює кількість збоїв або помилок, що виникають у системі за певний період, і зниження частоти збоїв є показником стабільності системи.

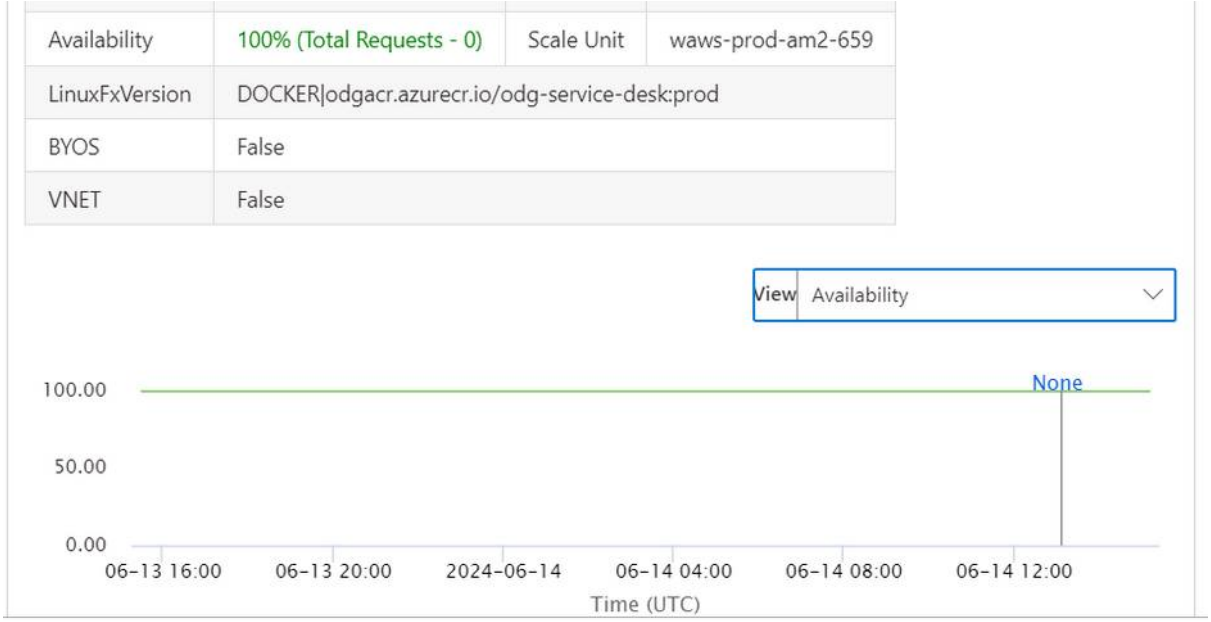

Рисунок 2.2 – Метрика надійності

Метрики безпеки відображають рівень захищеності системи і включають кількість виявлених вразливостей (Vulnerability Count) та час на усунення вразливостей (Time to Mitigate). Кількість виявлених вразливостей показує, скільки вразливих місць було знайдено в системі, і зниження цього показника свідчить про підвищення рівня безпеки. Час на усунення вразливостей вимірює, скільки часу потрібно для виправлення знайдених вразливостей, і швидше усунення вразливостей вказує на ефективність заходів безпеки.

Бізнес-метрики відображають вплив IT-систем на бізнес-процеси і включають коефіцієнт конверсії (Conversion Rate), час до виконання замовлення (Order Fulfillment Time) та задоволеність клієнтів (Customer

Satisfaction). Коефіцієнт конверсії показує процентне співвідношення кількості користувачів, які виконали бажану дію, до загальної кількості відвідувачів. Час до виконання замовлення вимірює, скільки часу потрібно для обробки та виконання замовлення, і зниження цього показника свідчить про ефективність бізнес-процесів. Задоволеність клієнтів оцінює рівень задоволеності клієнтів якістю послуг або продуктів, і високий рівень задоволеності клієнтів вказує на успішність бізнесу [5].

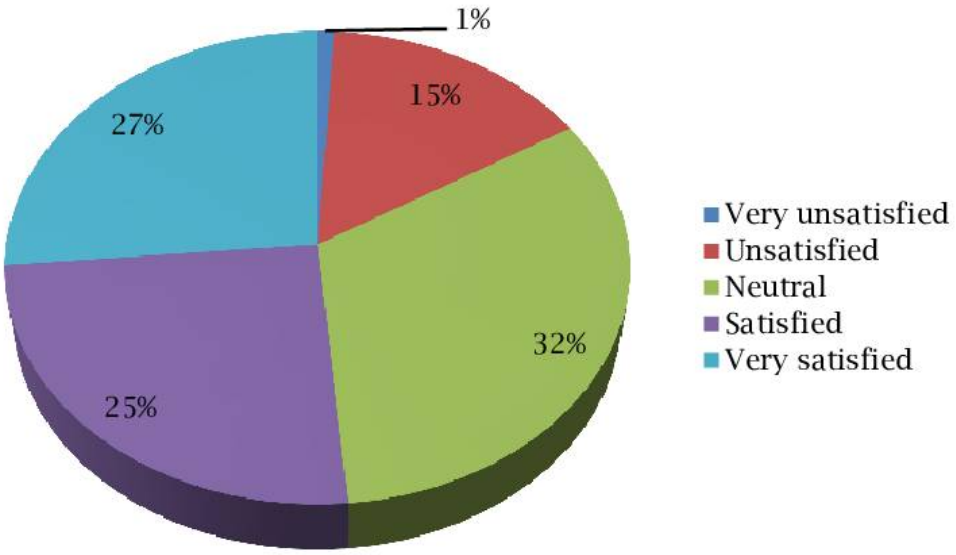

Рисунок 2.3 – Бізнес-метрика

Метрики використовуються для моніторингу і управління IT-системами, а також для прийняття рішень щодо оптимізації, масштабування та забезпечення безперервної роботи. Збір і аналіз метрик допомагає виявляти проблеми, планувати ресурси і оцінювати ефективність заходів, що вживаються. У хмарних середовищах, таких як Azure, метрики збираються і аналізуються за допомогою спеціалізованих інструментів, таких як Azure Monitor. Ці інструменти дозволяють налаштовувати політики автоскейлінгу на основі реальних даних про стан системи, що забезпечує динамічне і ефективне управління ресурсами.

Метрики є невід'ємною частиною процесу забезпечення якості IT-систем і дозволяють підтримувати високий рівень продуктивності, надійності і безпеки, що, в свою чергу, сприяє досягненню бізнес-цілей.

На основі цих метрик встановлюються тригери, які активують додавання або видалення ресурсів. Наприклад, якщо завантаженість процесора перевищує певний поріг, система може автоматично додати нові віртуальні машини для обробки запитів. Коли навантаження зменшується, надлишкові ресурси можуть бути вимкнені, що дозволяє зменшити витрати.

Автоскейлінг залежить від кількох факторів, включаючи налаштування політик масштабування, природу навантаження на систему, доступність ресурсів у хмарному середовищі та конкретні вимоги до продуктивності додатка. Правильне налаштування автоскейлінгу вимагає розуміння робочих навантажень додатка та можливих сценаріїв використання, що дозволяє створити ефективні правила і політики масштабування.

Крім того, автоскейлінг забезпечує гнучкість і адаптивність, що є важливим у сучасних динамічних середовищах, де навантаження можуть змінюватися дуже швидко. Це особливо корисно для веб-додатків, які піддаються різким коливанням трафіку, таких як інтернет-магазини під час розпродажів, новинні сайти під час важливих подій або будь-які сервіси, що зазнають пікових навантажень у певні періоди [8].

2.2 Налаштування автоскейлінгу

Загалом, автоскейлінг є ключовим елементом сучасного управління обчислювальними ресурсами, що дозволяє забезпечувати високу продуктивність і доступність сервісів при оптимальних витратах.

Автоскейлінг використовується для автоматичного регулювання обчислювальних ресурсів у відповідь на зміну навантаження на систему. Це забезпечує підтримку продуктивності додатків і економію ресурсів. Використання автоскейлінгу можна розділити на кілька основних кроків, які

включають налаштування політик масштабування, моніторинг метрик і автоматичне виконання дій.

Першим кроком є визначення метрик, які будуть використовуватися для моніторингу стану системи. Зазвичай це метрики продуктивності, такі як завантаження процесора (CPU), використання пам'яті, кількість оброблених запитів, час відгуку та інші показники. Ці метрики можуть бути зібрані за допомогою вбудованих інструментів моніторингу, наданих хмарними платформами, такими як Azure Monitor.

Наступним кроком є налаштування правил і політик автоскейлінгу. Це включає визначення порогових значень для кожної метрики, які ініціюватимуть масштабування. Наприклад, можна налаштувати правило, що додає нові віртуальні машини, коли завантаження процесора перевищує 70%, і зменшує їх кількість, коли завантаження падає нижче 30%. Крім того, можна визначити мінімальну і максимальну кількість обчислювальних одиниць, щоб уникнути надмірного масштабування або недостатнього забезпечення ресурсами.

Політики та правила автоскейлінгу залежать від багатьох факторів, які можуть варіюватися в залежності від специфіки додатка, бізнес-вимог, типу навантаження та доступних ресурсів. Нижче розглянуто основні аспекти, від яких залежать політики та правила автоскейлінгу, а також детальний опис кожного з них.

Одним з головних факторів, що визначають політики автоскейлінгу, є вимоги до продуктивності додатка. Це може включати такі показники, як максимальний час відгуку, доступність сервісу, здатність обробляти певну кількість запитів за секунду тощо. Вимоги до продуктивності визначають порогові значення метрик, при перевищенні яких необхідно виконати масштабування.

Тип навантаження на додаток грає важливу роль у визначенні політик автоскейлінгу. Навантаження може бути постійним, змінюваним або піковим. Для постійного навантаження можуть бути застосовані простіші політики з меншими порогами для масштабування, тоді як для пікового навантаження

необхідно враховувати швидкість збільшення і зменшення ресурсів для забезпечення стабільності роботи додатка.

Політики автоскейлінгу базуються на метриках продуктивності, які визначають стан системи.

Політики масштабування повинні враховувати швидкість реакції системи на зміни навантаження. Це включає час, необхідний для додавання або видалення ресурсів. Наприклад, якщо додавання нових віртуальних машин займає кілька хвилин, політики повинні враховувати це і починати масштабування заздалегідь, щоб уникнути простоїв.

Доступність ресурсів також визначає політики автоскейлінгу. У хмарних середовищах, таких як Azure, можуть бути встановлені обмеження на максимальну кількість обчислювальних одиниць, які можна використовувати. Політики масштабування повинні враховувати ці обмеження, щоб уникнути перевищення доступних ресурсів.

Вартість обчислювальних ресурсів є важливим фактором при розробці політик автоскейлінгу. Підвищення кількості ресурсів призводить до збільшення витрат, тому важливо знайти баланс між продуктивністю і витратами. Політики повинні забезпечувати ефективне використання ресурсів для мінімізації витрат, не знижуючи продуктивність додатка.

Конкретні сценарії використання додатка також впливають на політики масштабування. Наприклад, для інтернет-магазинів може бути важливо забезпечити високу продуктивність під час розпродажів, тоді як для внутрішніх корпоративних систем важливішим може бути стабільна робота протягом робочого дня.

Аналіз історичних даних про навантаження на систему може допомогти в налаштуванні політик автоскейлінгу. Вивчення попередніх патернів навантаження дозволяє краще розуміти, коли і які ресурси будуть необхідні. Це допомагає налаштувати правила так, щоб вони ефективно реагували на очікувані зміни навантаження [12].

Політики автоскейлінгу не є статичними і потребують регулярного тестування та оптимізації. Це включає перевірку, як система реагує на різні сценарії навантаження, коригування порогових значень і параметрів масштабування для забезпечення оптимальної продуктивності та ефективного використання ресурсів.

Коли налаштовані політики автоскейлінгу, система починає автоматичний моніторинг метрик у реальному часі. Якщо значення метрик перевищують встановлені пороги, тригери масштабування активують відповідні дії. Наприклад, якщо кількість запитів різко зростає, автоскейлінг може автоматично додати нові інстанси веб-сервера, щоб забезпечити обробку додаткового трафіку. Після зменшення навантаження зайві ресурси автоматично відключаються, що дозволяє зменшити витрати.

Використання автоскейлінгу включає інтеграцію з хмарними сервісами, такими як Azure Virtual Machine Scale Sets або Azure App Service. Ці сервіси надають інструменти для налаштування і управління автоскейлінгом. Наприклад, у Azure можна налаштувати автоскейлінг через портал Azure, використовуючи інтуїтивно зрозумілий інтерфейс для створення і редагування правил масштабування.

Крім того, можна використовувати API і скрипти для автоматизації процесу налаштування автоскейлінгу. Це дозволяє інтегрувати автоскейлінг з іншими системами управління та оркестрації, такими як Kubernetes для контейнеризованих додатків. За допомогою Kubernetes Horizontal Pod Autoscaler можна автоматично масштабувати кількість подів на основі метрик використання ресурсів, таких як CPU або пам'ять.

Важливим аспектом використання автоскейлінгу є тестування і оптимізація налаштувань. Це включає перевірку, як система реагує на різні сценарії навантаження, і коригування правил масштабування для забезпечення оптимальної продуктивності та економії ресурсів. Регулярний моніторинг і аналіз роботи автоскейлінгу дозволяють виявляти і усувати можливі проблеми, а також адаптувати налаштування до змін у навантаженні та вимогах додатка.

Автоскейлінг є потужним інструментом для забезпечення ефективного управління обчислювальними ресурсами, що дозволяє автоматично адаптувати систему до змін у навантаженні, забезпечуючи стабільну продуктивність і оптимальні витрати.

2.3 Вибір моделі нейронної мережі

Нейронні мережі є потужним інструментом для розв'язання широкого спектра завдань у галузі штучного інтелекту, машинного навчання та обробки даних. Вони надихаються біологічними нейронними мережами, такими як мозок людини, і здатні навчатися з прикладів, узагальнювати.

Нейронна мережа — це математична модель, яка складається з великої кількості взаємозв'язаних обчислювальних елементів, званих нейронами. Кожен нейрон приймає один або кілька вхідних сигналів, обробляє їх і передає вихідний сигнал іншим нейронам. Основною метою нейронної мережі є навчання функції, яка може зв'язувати вхідні дані з бажаними вихідними результатами.

2.3.1 Одношаровий перцептрон

Одношаровий перцептрон є найпростішою моделлю нейронної мережі. Він складається з одного шару нейронів, які безпосередньо зв'язують вхідні дані з виходом. Основний принцип роботи одношарового перцептрона полягає у визначенні зваженої суми вхідних сигналів, до якої додається значення зсуву. Ця сума пропускається через порогову функцію активації, результатом якої є +1 чи -1.

Відповідно, вихідний сигнал визначає, до якого класу належить вхідний сигнал (рис. 3.1).

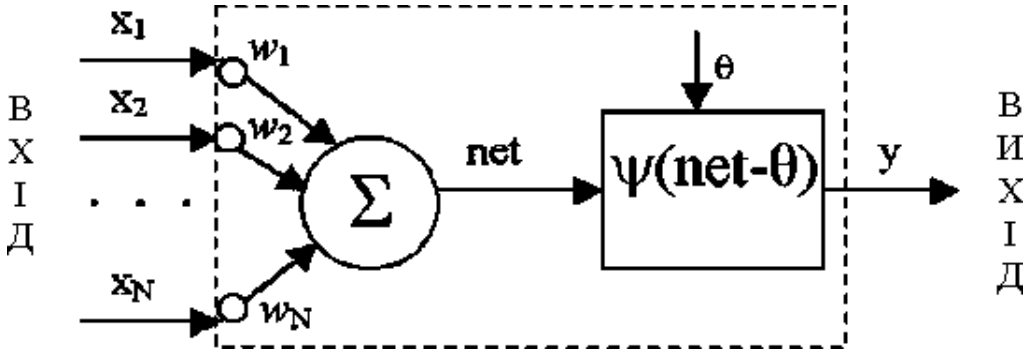

Рисунок 2.4 – Схема одношарового персептрону

2.3.2 Багатошаровий перцептрон (MLP)

Багатошаровий перцептрон (MLP) є більш складною моделлю, яка складається з одного або більше прихованих шарів нейронів між вхідним і вихідним шарами. Кожен шар додає рівень абстракції, що дозволяє мережі розпізнавати складніші закономірності. MLP використовує нелінійні функції активації, такі як сигмоїдна функція або ReLU (Rectified Linear Unit), для забезпечення здатності до навчання складних функцій. Багатошарові перцептрони використовуються для різних задач, включаючи класифікацію, регресію та розпізнавання образів. Вони є основою для багатьох сучасних нейронних мереж і є дуже ефективними для обробки різноманітних типів даних. Багатошаровий перцептрон також має здатність до генералізації, тобто до створення правильних висновків на основі нових, раніше не бачених даних. Це досягається завдяки навчанню моделі на великій кількості прикладів, що дозволяє їй виявляти ключові закономірності й залежності в даних. Однією з важливих характеристик MLP є його універсальність: мережа може бути адаптована до різних типів задач, змінюючи кількість нейронів, шарів і параметри навчання. Завдяки цим властивостям багатошарові перцептрони є фундаментальними компонентами для створення складніших архітектур, таких як згорткові та рекурентні нейронні мережі.

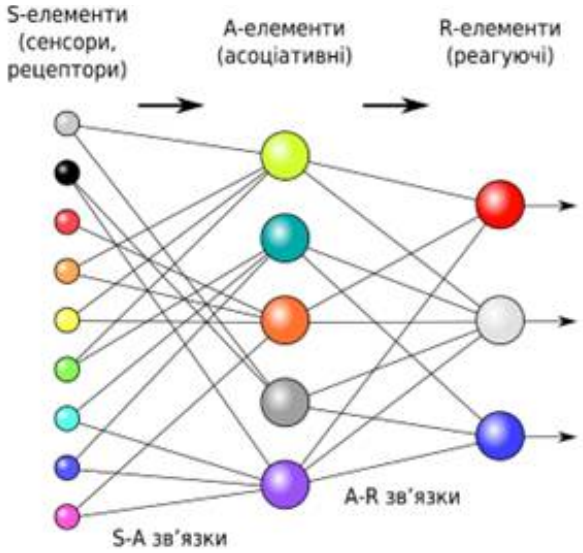

Рисунок 2.5 – Схема багатошарового персептрона

2.3.3. Рекурентні нейронні мережі (RNN)

Рекурентні нейронні мережі (RNN) відрізняються від звичайних нейронних мереж тим, що вони мають зворотні зв'язки, які дозволяють зберігати інформацію про попередні входи. Це робить їх ідеальними для обробки послідовностей даних, таких як текст, мова або часові ряди. Завдяки своїй здатності враховувати контекст, RNN широко використовуються для завдань обробки природної мови, машинного перекладу та прогнозування часових рядів. Популярними варіантами RNN є LSTM (довготривала короткочасна пам'ять) і GRU (гейтована рекурентна одиниця), які розв'язують проблему зникнення градієнта і дозволяють ефективно працювати з довгими послідовностями даних. Рекурентні нейронні мережі також знаходять застосування у задачах генерації тексту, створення музики та аналізу поведінки користувачів, де врахування послідовності подій має критично важливе значення.

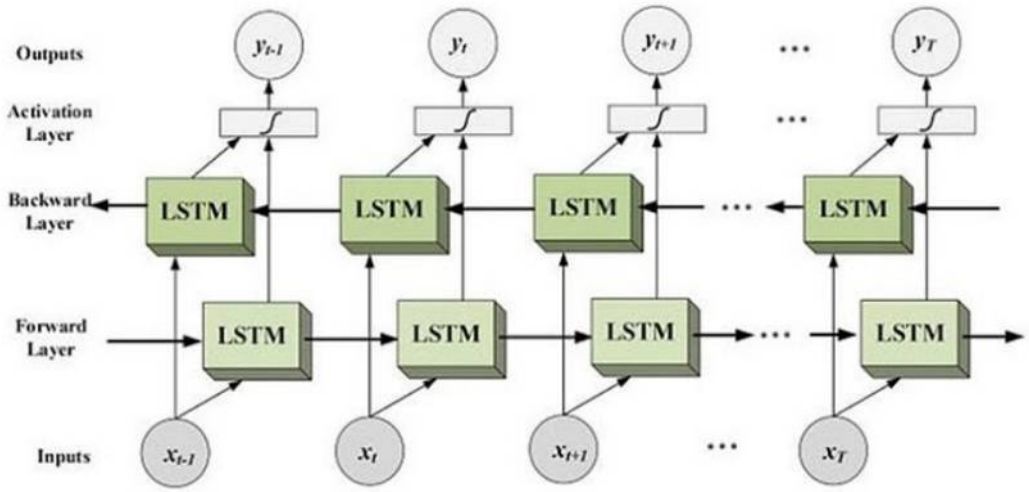

Рисунок 2.6 – Схема рекурентної нейронної мережі

2.3.4 Конволюційні нейронні мережі (CNN)

Конволюційні нейронні мережі (CNN) спеціально розроблені для обробки даних, які мають просторову структуру, таких як зображення і відео. Вони використовують операції згортки для виділення особливостей з вхідних даних, що дозволяє їм бути дуже ефективними для задач розпізнавання образів. CNN складаються з шарів згортки, підвибірки (pooling) і повнозв'язаних шарів. Завдяки своїй архітектурі CNN здатні автоматично виділяти важливі особливості на різних рівнях абстракції, що робить їх дуже ефективними для розпізнавання об'єктів на зображеннях, класифікації зображень та інших задач обробки візуальної інформації.

Окрім розпізнавання образів, конволюційні нейронні мережі успішно застосовуються у відеоаналізі, медичній діагностиці, розпізнаванні рукописного тексту та обробці тривимірних даних. Їхня здатність виявляти просторові та локальні залежності у вхідних даних робить CNN незамінними в задачах комп'ютерного зору, таких як сегментація зображень, виявлення об'єктів і аналіз динамічних сцен. Крім того, завдяки ефективному використанню параметрів і можливості перенавчання на різних наборах даних, CNN широко

використовуються в різних галузях, включаючи автономні транспортні засоби, безпеку та розваги.

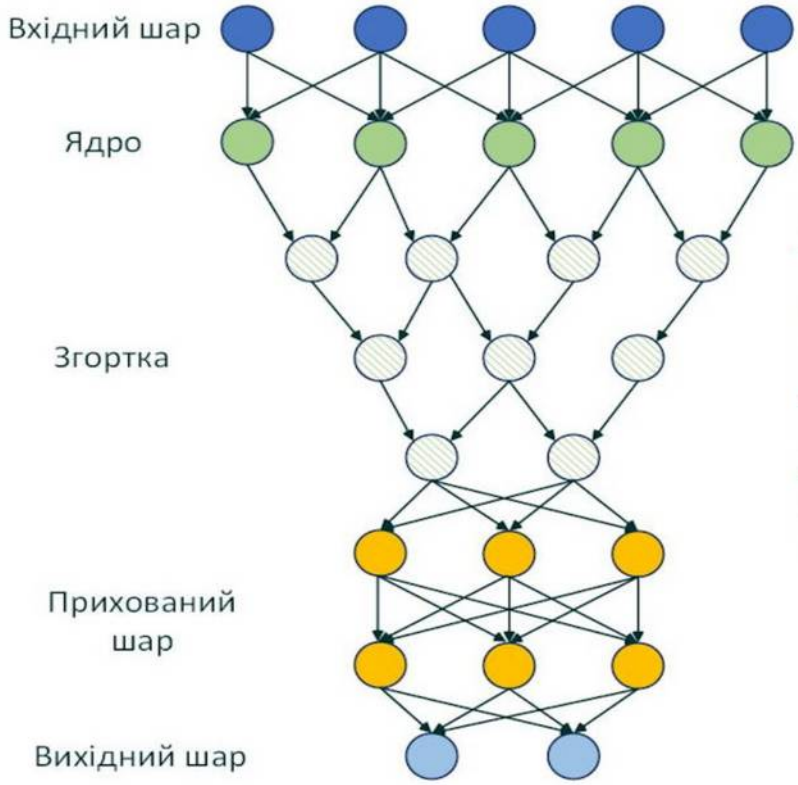

Рисунок 2.7 – Схема конволюційної нейронної мережі

2.3.5 Мережі Кохонена

Мережі Кохонена, також відомі як самоорганізуючі карти (SOM), використовуються для кластеризації і візуалізації даних. Вони навчаються знаходити кластери в даних без потреби в контрольованому навчанні. Мережі Кохонена складаються з вхідного шару і вихідного шару, де вихідні нейрони розташовуються на двовимірній решітці. Під час навчання мережа Кохонена змінює ваги зв'язків таким чином, що топологічно близькі вузли реагують на схожі вхідні сигнали. Це дозволяє виявляти схожість між різними класами і використовувати їх для кластерного аналізу, розпізнавання образів та інших задач, де потрібно виявляти структури в даних.

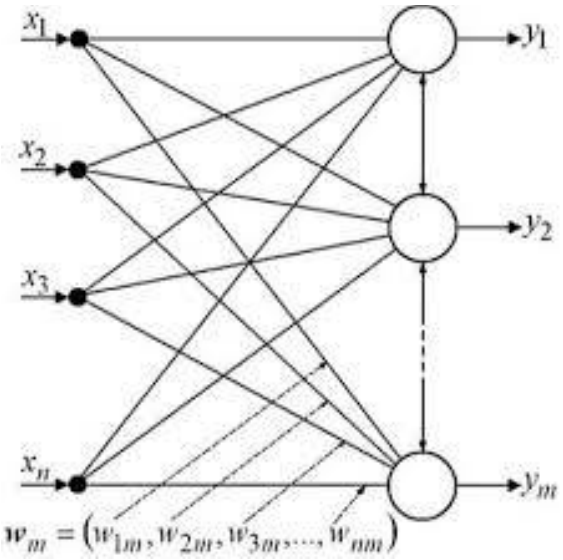

Рисунок 2.8 – Схема мережі Кохонена

2.3.6 Процес навчання нейронних мереж

Процес навчання нейронних мереж є ключовим етапом у створенні ефективних моделей штучного інтелекту. Він включає кілька важливих кроків, що дозволяють мережі оптимально налаштовувати свої параметри для виконання конкретних завдань.

На початковому етапі відбувається ініціалізація ваг. Початкові ваги зв'язків між нейронами встановлюються випадковими значеннями, щоб мережа могла почати процес навчання. Ці випадкові значення зазвичай малі, щоб уникнути переваги одного сигналу над іншими на початковому етапі.

Після ініціалізації починається прямий прохід (forward pass). Вхідні дані передаються через мережу, і на кожному шарі обчислюються вихідні значення нейронів. Цей процес включає обчислення зважених сум входів для кожного нейрона та застосування функції активації для отримання вихідного сигналу. Функція активації може бути лінійною або нелінійною, залежно від типу нейронної мережі та задачі.[14, 15]

Обчислення помилки є наступним критичним кроком. Після отримання вихідного сигналу мережі він порівнюється з бажаним вихідним сигналом (міткою або ціллю). Для цього використовується функція втрат, яка обчислює різницю між фактичним виходом і бажаним виходом. Найпоширенішими

функціями втрат є середньоквадратична помилка для регресії та крос-ентропія для класифікації.

Зворотний прохід (backpropagation) є основним методом для корекції ваг на основі обчисленої помилки. Цей процес включає поширення помилки назад через мережу, починаючи з вихідного шару до вхідного. Під час цього кроку обчислюється градієнт функції втрат щодо кожної ваги, використовуючи метод зворотного поширення похибки. Градієнт показує, наскільки зміна кожної ваги вплине на загальну помилку.

Корекція ваг відбувається шляхом оновлення вагових коефіцієнтів з урахуванням обчислених градієнтів. Це робиться за допомогою алгоритму градієнтного спуску або його варіацій. Швидкість навчання, або темп навчання, визначає, наскільки сильно ваги коригуються на кожній ітерації. Занадто висока швидкість навчання може призвести до нестабільності, тоді як занадто низька — до дуже повільного навчання.

Процес навчання повторюється для багатьох ітерацій, поки помилка не стане прийнятно малою або не буде досягнуто задану кількість епох. Кожна ітерація включає прямий прохід, обчислення помилки, зворотний прохід і корекцію ваг. Під час навчання можуть використовуватися різні стратегії для покращення продуктивності, такі як адаптивні методи навчання (Adam, RMSprop), регуляризація для запобігання перенавчанню (L2-регуляризація, Dropout) та інші техніки.

Після завершення навчання нейронна мережа здатна узагальнювати знання, отримані з навчальних даних, і застосовувати їх до нових, невідомих даних. Це дозволяє мережі виконувати передбачення, класифікацію, розпізнавання образів та інші завдання з високою точністю.

Таким чином, процес навчання нейронної мережі є складним і багатоетапним, але він є фундаментальним для створення потужних моделей, здатних вирішувати широкий спектр завдань у різних областях. Вибір правильної моделі нейронної мережі залежить від конкретної задачі, вимог до точності, обчислювальних ресурсів і наявних даних. Розуміння основних

принципів і типів нейронних мереж допомагає приймати обґрунтовані рішення під час проектування і розробки моделей.

3 ВИБІР НЕЙРОННОЇ МЕРЕЖІ ДЛЯ КОРЕГУВАННЯ АВТОМАТИЧНОГО НАЛАШТУВАННЯ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ

У наш час є багато проблем пов’язаних із процесами перевантаження серверів та невчасного додавання потужностей до ресурсів, що існують, тому нейромережі стають корисним та перспективним напрямком розвитку та інтегрування цієї технології до систем, що вже створені. Таким чином, якщо є виклик про покращення процесу автоматичного та вчасного масштабування, час звернутися до найпростішого – перцептронів.

Перцептрон — це модель, яка складається з трьох основних типів елементів: входів, вагових коефіцієнтів і активаційної функції. Входи (сигнали) отримують інформацію від сенсорів, потім ці сигнали зважуються вагами. Зважені сигнали сумуються, і результат передається через активаційну функцію, яка вирішує, чи активується нейрон (тобто, чи буде вихід 1 або 0) [16].

Компоненти перцептрона:

- входи (Inputs): приймають сигнали від сенсорів;
- ваги (Weights): множать кожен вхід на ваговий коефіцієнт;
- суматор (Summation): обчислює зважену суму всіх входів;
- активаційна функція (Activation Function): вирішує, чи активується нейрон на основі зваженої суми.

Основна математична задача, яку розв'язує перцептрон — це лінійне розділення даних, де він створює лінійну межу для класифікації вхідних даних на дві категорії.

Перцептрон приймає на вхід сигнали, що надходять від сенсорів. Ці сигнали зважуються за допомогою вагових коефіцієнтів, потім сумуються. Якщо зважена сума перевищує певний поріг, нейрон активується, і на виході

з'являється сигнал. Така проста схема дозволяє перцептрону розв'язувати задачі лінійного розділення.

У контексті предикатів перцептрон моделюється як система логічних висловлювань. Входи інтерпретуються як предикати, а вагові коефіцієнти визначають їхню важливість. Активація відбувається, коли лінійна комбінація предикатів відповідає встановленому критерію.

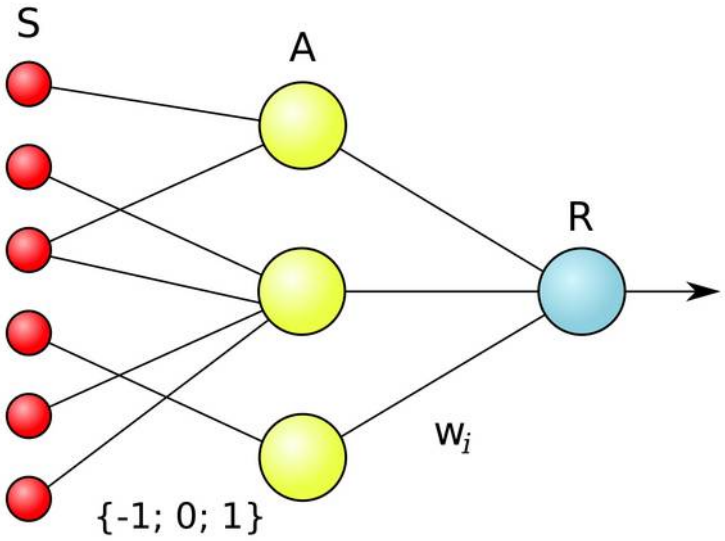

Рис. 3.1- Логічна схема елементарного перцептрону

3.1 Алгоритм навчання одношарового перцептрона

Елементарний перцептрон складається з трьох типів елементів: S-елементів, A-елементів та одного R-елемента. S-елементи утворюють шар сенсорів або рецепторів. У фізичному втіленні вони можуть бути, наприклад, світлочутливими клітинами сітківки ока або фоторезисторами матриці камери. Кожен рецептор може бути в одному з двох станів — спокою або збудження, і лише в стані збудження він передає одиничний сигнал до наступного шару, асоціативним елементам.

A-елементи називаються асоціативними, оскільки кожному такому елементу зазвичай відповідає цілий набір (асоціація) S-елементів. A-елемент активується, коли кількість сигналів від S-елементів на його вході перевищує певний поріг θ.

Сигнали від збуджених А-елементів передаються до суматора R, причому сигнал від і-го асоціативного елемента передається з коефіцієнтом ω_i. Цей коефіцієнт називається вагою зв'язку між А і R.

Подібно до А-елементів, R-елемент підраховує суму значень вхідних сигналів, помножених на ваги (лінійну форму). R-елемент, а разом з ним і елементарний перцептрон, видає «1», якщо лінійна форма перевищує поріг θ, інакше на виході буде «-1».

Після навчання перцептрон готовий працювати в режимі розпізнавання або узагальнення. У цьому режимі перцептрону пред'являються нові, раніше невідомі об'єкти, і він повинен визначити, до якого класу вони належать. Робота перцептрона відбувається так: при пред'явленні об'єкта збуджені А-елементи передають сигнал R-елементу, що дорівнює сумі відповідних коефіцієнтів ω_i. Якщо ця сума позитивна, приймається рішення, що даний об'єкт належить до першого класу, а якщо негативна — до другого.

Одношаровий перцептрон здатний розпізнавати найпростіші образи. Окремий нейрон обчислює зважену суму сигналів вхідних елементів, віднімає значення зсуву і пропускає результат через жорстку порогову функцію, вихід якої дорівнює +1 чи -1. В залежності від значення вихідного сигналу приймається рішення:

1) +1 — вхідний сигнал належить до класу А;
2) -1 — вхідний сигнал належить до класу В.

На рисунку 3.2 показано схему одношарового перцептрона, графік передатної функції і схему вирішальних областей, створених у багатовимірному просторі вхідних сигналів. Вирішальні області визначають, які вхідні образи будуть віднесені до класу А, а які — до класу В. Перцептрон, що складається з одного нейрона, формує дві вирішальні області, які розділено гіперплощиною.

Алгоритм навчання одношарового перцептрона є основним методом, який використовується для налаштування ваг мережі так, щоб вона могла правильно класифікувати вхідні сигнали. Цей алгоритм є відносно простим, але

він демонструє основні принципи, що лежать в основі навчання нейронних мереж.

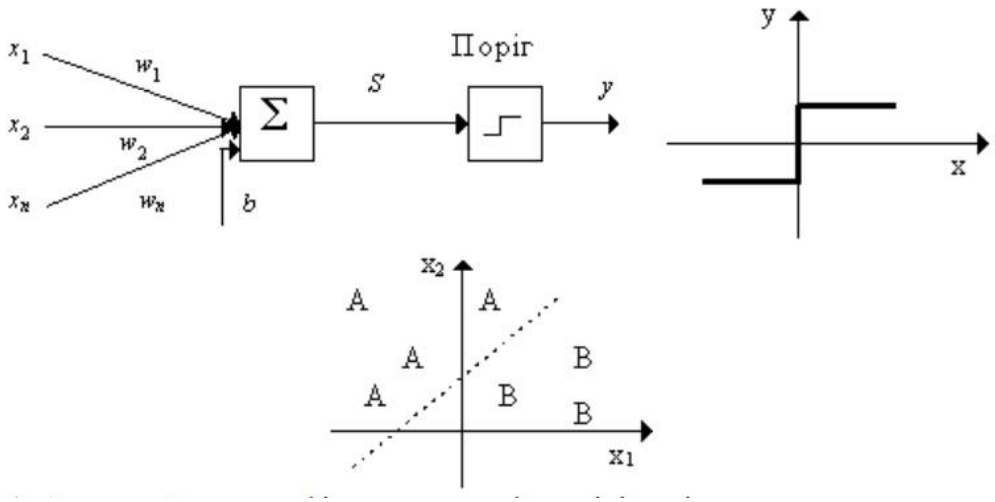

Рисунок 3.2 - Схема нейрона, графік передатної функції і поділяюча поверхня

Процес навчання починається з ініціалізації ваг і значення зсуву. Синаптичні ваги, які пов'язують вхідні нейрони з вихідними, встановлюються на випадкові малі значення. Це потрібно для того, щоб уникнути домінування одного нейрону над іншими на початкових етапах навчання. Значення зсуву, яке також відоме як bias, відіграє роль у зміщенні активації нейрону і також встановлюється на початкове значення.

Після ініціалізації ваг відбувається процес пред'явлення мережі нового вхідного і бажаного вихідного сигналів. Вхідні дані, що складаються з вектора $x=(x_1, x_2, \ldots, x_n)$, подаються на вхід нейрону разом з бажаним вихідним сигналом *d*. Цей етап важливий, оскільки мережа повинна навчитися зв'язувати вхідні дані з відповідними вихідними значеннями.

На наступному етапі відбувається обчислення вихідного сигналу нейрона. Вихідний сигнал *y* обчислюється як сума зважених вхідних сигналів, до якої додається значення зсуву:

$$y=\sum_{i=1}^{n} \omega_i x_i + b, \qquad (3.1)$$

де ω_i— це вага зв'язку від і-го вхідного нейрона, x_i— це вхідний сигнал від і-го нейрона, b — значення зсуву.

Результат цієї суми передається через порогову функцію активації, яка визначає вихідний сигнал нейрону. Для одношарового перцептрона найчастіше використовується жорстка порогова функція, яка видає +1, якщо зважена сума більше або дорівнює порогу, і -1, якщо менше:

$$y=\begin{cases} +1, \wedge \textit{якщо} \sum_{i=1}^{n} \omega_i x_i + b \geq \theta \\ -1, \wedge \textit{якщо} \sum_{i=1}^{n} \omega_i x_i + b < \theta \end{cases} \quad (3.2)$$

де θ — це порогове значення.

Після обчислення вихідного сигналу відбувається налаштування значень ваг. Якщо вихідний сигнал не збігається з бажаним, ваги коригуються за допомогою правила навчання перцептрона:

$$\omega_i(t+1)=\omega_i(t)+r\cdot\left(d(t)-y(t)\right)\cdot x_i(t), \quad (3.3)$$

де ω_i — вага зв'язку від і-го вхідного нейрона до нейрона в момент час t, r — швидкість навчання, d(t) — фактичний вихідний сигнал. Швидкість навчання є позитивною константою, зазвичай меншою за 1, яка визначає, наскільки сильно коригуються ваги на кожній ітерації.

Цей процес повторюється для кожного вхідного сигналу з навчального набору даних. Якщо мережа приймає правильне рішення (тобто фактичний вихідний сигнал збігається з бажаним), ваги не модифікуються. Процес продовжується до тих пір, поки всі вхідні сигнали не будуть правильно класифіковані, або не буде досягнуто задану кількість епох.

Одношаровий перцептрон є основною моделлю, яка демонструє принципи навчання нейронних мереж. Хоча він має обмеження і може

вирішувати лише лінійно роздільні задачі, його алгоритм навчання є важливим кроком у розумінні більш складних нейронних мереж і їх навчальних процесів.

Таким чином, можемо зробити висновок, що одношаровий перцептрон є найпростішою моделлю нейронної мережі, що легко впроваджується та налаштовується. Він потребує мінімальних обчислювальних ресурсів, що особливо важливо у хмарному середовищі, де економія ресурсів є критичною.

Завдання прийняття рішення про збільшення чи зменшення потужностей веб-приложення є лінійно роздільним. Це означає, що можна встановити поріг (лінію розділу), який визначає, коли необхідно збільшувати потужності, а коли — зменшувати. Одношаровий перцептрон чудово справляється з такими задачами, де вихід залежить від простого порівняння вхідних даних із пороговим значенням.

Одношаровий перцептрон має швидкий процес навчання, що дозволяє швидко адаптувати модель до змін у паттернах трафіку чи навантаження. Це дозволяє постійно підтримувати ефективну роботу веб-приложення без потреби в складних і довгих тренувальних процесах [16].

3.2 Код для налаштування прецептрону за допомогою мови програмування Python

Автоматичне масштабування є одним з ключових механізмів для забезпечення стабільної і високопродуктивної роботи додатків у хмарному середовищі. Більшість проблем, що виникають під час роботи додатків, пов'язані з недостатньою кількістю обчислювальних ресурсів. У цьому контексті буде дуже корисно налаштувати одношаровий перцептрон для автоматичного прийняття рішень щодо масштабування на основі кількості повідомлень у сервісах.

Одношаровий перцептрон є простою моделлю нейронної мережі, яка може ефективно класифікувати вхідні дані та приймати рішення на основі порогових значень. У випадку з вашою задачею, перцептрон може

використовуватися для визначення необхідності збільшення або зменшення потужностей веб-додатка залежно від кількості повідомлень. Це дозволить автоматично адаптувати ресурси під поточне навантаження, забезпечуючи стабільну роботу системи і оптимальне використання ресурсів.

Цей приклад коду демонструє навчання одношарового перцептрона і його використання для прийняття рішень про масштабування веб-додатка на основі кількості повідомлень. Вихідне рішення визначається на основі порогового значення, яке встановлюється під час навчання перцептрона. Якщо вихідний сигнал дорівнює 1, то це означає необхідність збільшення ресурсів, якщо -1 — зменшення. Нижче наведено приклад коду на Python, який реалізує одношаровий перцептрон для автоматичного масштабування:

Лістинг 3.1 – Приклад коду на Python, який реалізує одношаровий перцептрон для автоматичного масштабування

```
import numpy as np

# Функція активації (порогова функція)
def activation_function(x):
    return 1 if x >= 0 else -1

# Навчання перцептрона
def train_perceptron(training_data, learning_rate, epochs):
    weights = np.random.rand(len(training_data[0]) - 1)
    bias = np.random.rand(1)

    for _ in range(epochs):
        for sample in training_data:
            x = np.array(sample[:-1])
            expected = sample[-1]
            result = activation_function(np.dot(weights, x) +
bias)
            error = expected - result
```

```
            weights += learning_rate * error * x
            bias += learning_rate * error
    return weights, bias

# Використання перцептрона для прийняття рішень
def predict(weights, bias, x):
    return activation_function(np.dot(weights, x) + bias)

# Приклад навчальних даних (кількість повідомлень, вихідне рішен-
ня)
# 1 означає необхідність збільшення ресурсів, -1 - зменшення
training_data = [
    [5, 1],
    [10, 1],
    [15, 1],
    [20, -1],
    [25, -1]
]

# Параметри навчання
learning_rate = 0.01
epochs = 1000

# Навчання перцептрона
weights, bias = train_perceptron(training_data, learning_rate,
epochs)

# Використання перцептрона для прийняття рішень
current_message_count = 18
decision = predict(weights, bias,
np.array([current_message_count]))
print("Рішення про масштабування:", "Збільшити" якщо decision == 1
else "Зменшити")
```

Функція активації визначає, чи повинен нейрон активуватися (відправити сигнал 1) або залишитися неактивним (відправити сигнал -1) залежно від

значення входу. Вона є пороговою функцією, яка повертає 1, якщо вхідне значення більше або дорівнює нулю, і -1, якщо менше нуля.

Таблиця 3.1 - Переваги використання одношарового перцептрона для задач автоматичного масштабування

Фактор	Переваги одношарового перцептрона
Простота реалізації	Одношаровий перцептрон є найпростішою моделлю нейронної мережі, яку легко реалізувати і налаштувати.
Мінімальні обчислювальні ресурси	Потребує мінімальних обчислювальних ресурсів, що дозволяє ефективно використовувати його в хмарних середовищах.
Швидкість навчання	Навчання одношарового перцептрона відбувається швидко завдяки простій структурі і обмеженій кількості параметрів.
Ефективність для лінійно роздільних задач	Одношаровий перцептрон добре підходить для задач, де дані можуть бути лінійно розділені.
Легкість інтерпретації	Результати роботи перцептрона легко інтерпретувати, оскільки він використовує просту порогову функцію активації.
Можливість розширення	Модель можна розширити, додаючи нові входи або змінюючи порогові значення для покращення продуктивності.

Код навчає перцептрон на основі навчальних даних, а потім використовує навчену модель для прийняття рішення про масштабування на основі поточної кількості повідомлень. У цьому прикладі, якщо кількість повідомлень дорівнює 18, то вихідне рішення буде вказувати на необхідність збільшення або

зменшення ресурсів. Переваги використання одношарового перцептрона для задач автоматичного масштабування наведені у табл. 4.1.

Впровадження одношарового перцептрона для автоматичного масштабування дозволить забезпечити адаптивне і ефективне управління обчислювальними ресурсами, підтримуючи стабільну продуктивність вашого веб-додатка.

4 ІНТЕГРУВАННЯ НЕЙРОННОЇ МЕРЕЖІ ДО АРХІТЕКТУРИ ПРОЕКТУ

Сучасні підходи до автоматичного масштабування часто базуються на статичних правилах або евристичних алгоритмах, що можуть бути недостатньо гнучкими для адаптації до змінюваних умов. Використання нейронних мереж, таких як одногаровий персептрон, дозволяє створити більш адаптивні та інтелектуальні системи, які враховують історичні дані та поточні показники навантаження. У контексті хмарних обчислень це означає більш ефективне використання ресурсів, зниження витрат та забезпечення високого рівня продуктивності.

Одношаровий персептрон є ідеальним вибором для початкової реалізації завдяки своїй простоті, зрозумілості та низьким обчислювальним вимогам. Він здатний приймати рішення на основі аналізу вхідних даних, таких як кількість запитів або навантаження на CPU, що робить його ефективним інструментом для інтеграції в архітектуру хмарних обчислень.

4.1 Архітектура проекту з використанням автоскейлу та нейроної мережі

Архітектура — це не просто набір технічних рішень, а стратегічний інструмент, що дозволяє впорядкувати процес розробки, зробити його прозорим і передбачуваним. Завдяки чіткій архітектурі кожен учасник команди розуміє свою роль і завдання, а всі етапи реалізації проєкту стають зрозумілими й логічно пов’язаними. Це зменшує ризик помилок, забезпечує ефективну комунікацію між різними відділами та сприяє оперативному вирішенню проблем, які можуть виникати під час роботи.

Окрім цього, архітектура дозволяє заздалегідь передбачити, як система буде поводитися в реальних умовах експлуатації. Вона враховує не лише поточні потреби, але й можливі зміни, які можуть виникнути в майбутньому:

зростання навантаження, інтеграція нових технологій чи змінення бізнес-моделі. Завдяки продуманій архітектурі проєкт залишається гнучким, а його адаптація до нових викликів стає менш витратною і складною.

Ще один важливий аспект архітектури — це її вплив на якість і безпеку системи. Добре спроєктована архітектура забезпечує стабільність роботи навіть за умов високого навантаження, знижує ризик збоїв і полегшує процес відновлення у випадку інцидентів. Крім того, архітектура дозволяє інтегрувати необхідні заходи безпеки ще на етапі планування, що забезпечує захист даних і системи від потенційних загроз. Без чітко визначеної архітектури багато аспектів, таких як резервування, масштабування чи захист від атак, можуть бути пропущені або реалізовані неефективно.

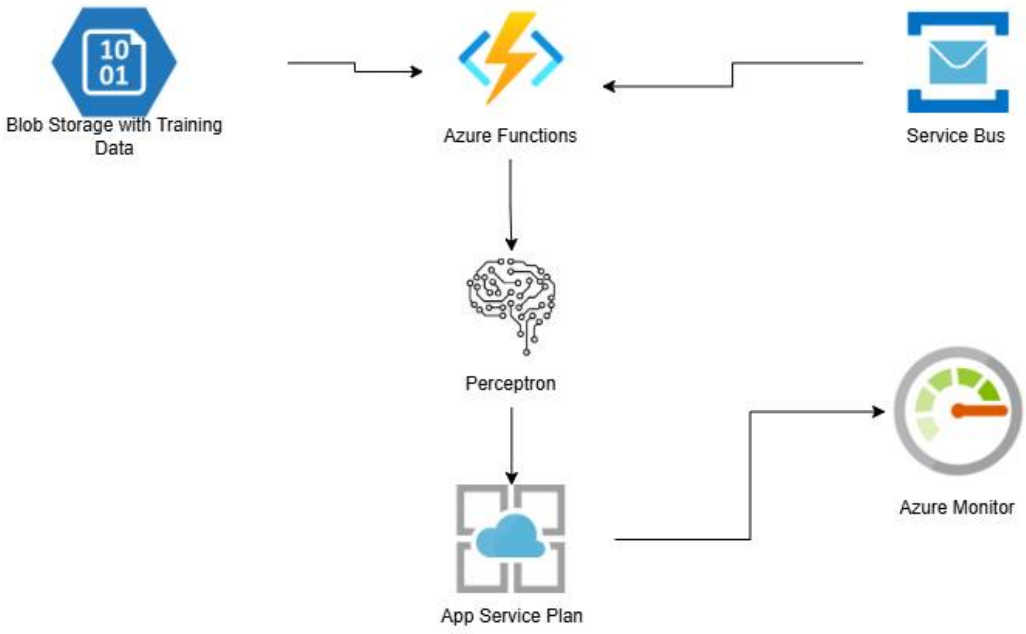

Рисунок 4.1 – Схема архітектури із залученням нейроної мережі у автоскейлі

На схемі показана взаємодія компонентів архітектури:

- blob storage: зберігає історичні дані, які використовуються для навчання персептрону;

- azure functions: виконує обробку даних із Blob Storage, Service Bus і тренує/використовує персептрон;

- service bus: виступає джерелом поточних даних про кількість повідомлень у черзі;

- training data: дані зберігаються у Blob Storage для навчання персептрону;

- perceptron: аналізує вхідні дані й приймає рішення про масштабування;

- app service plan: реалізує масштабування на основі рішення персептрону;

- azure monitor: відстежує метрики масштабування та продуктивності.

4.2 Опис сервісів, використанних у проекті та їх функцій

Blob Storage – це хмарне сховище, призначене для збереження великих обсягів неструктурованих даних. У цьому випадку воно використовується для зберігання історичних даних, необхідних для навчання персептрону. Ці дані можуть включати інформацію про навантаження, кількість запитів або інші метрики, які впливають на прийняття рішень про масштабування. Blob Storage дозволяє безпечно та економно зберігати дані, забезпечуючи легкий доступ через API або SDK

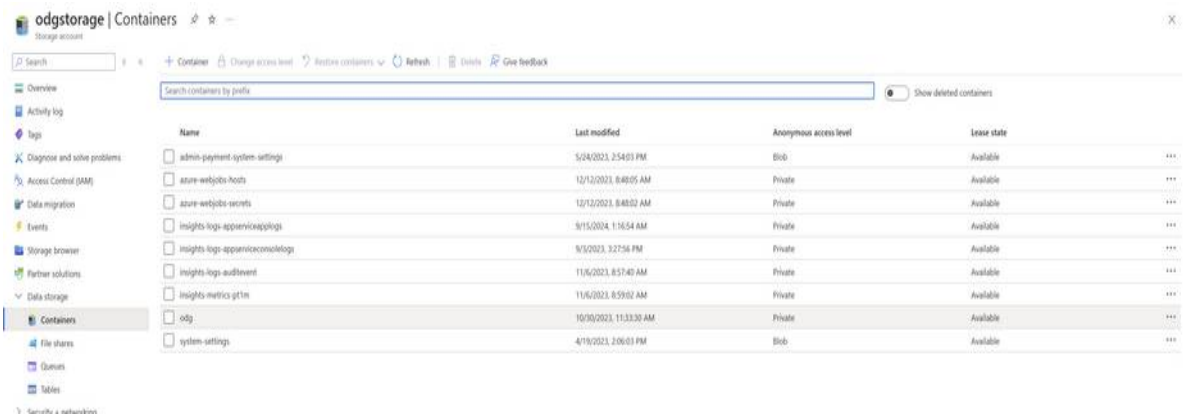

Рисунок 4.2 – Інтерфейс Blob Storage

Azure Functions – це безсерверне рішення, яке виконує код у відповідь на певні події. У цьому випадку функції обробляють дані з Blob Storage,

отримують інформацію з Service Bus і тренують або застосовують персептрон для аналізу даних. Наприклад, при надходженні нових даних функція може оновити модель персептрону, а при обробці реальних даних використати вже натреновану модель для прийняття рішень.

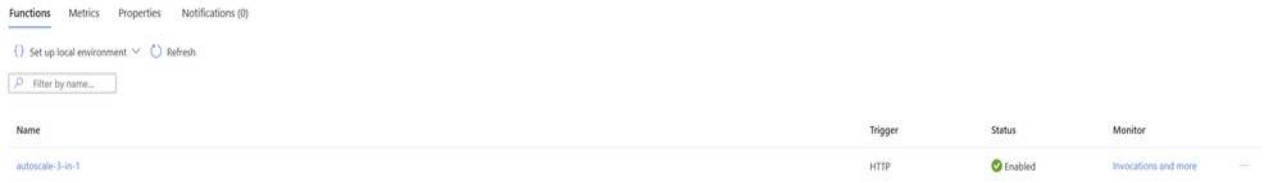

Рисунок 4.3 – Інтерфейс Azure Functions

Service Bus – це хмарний сервіс обміну повідомленнями, який використовується для передачі даних між компонентами системи. У цій системі він виступає джерелом поточних даних, таких як кількість повідомлень у черзі. Ці дані аналізуються, щоб визначити, коли навантаження потребує масштабування.

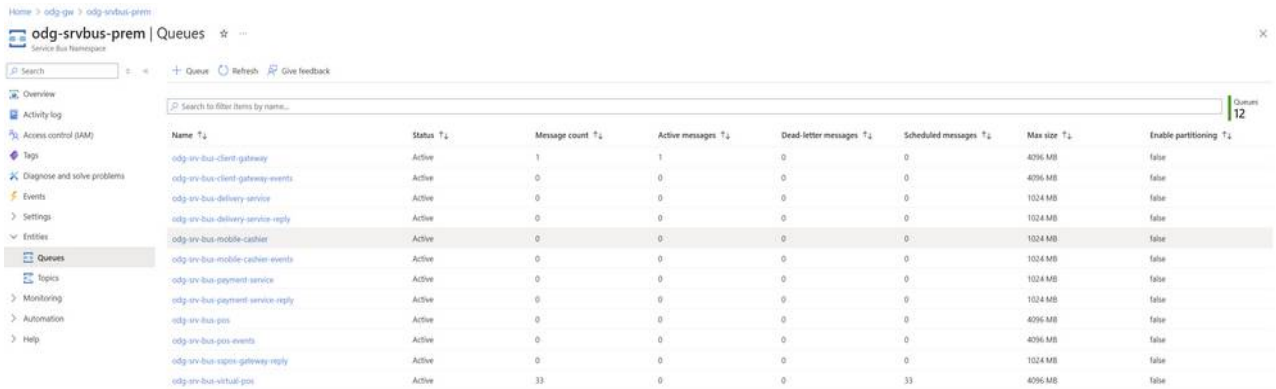

Рисунок 4.4 – Інтерфейс Service Bus

Training Data – це набір даних, що зберігається в Blob Storage і використовується для навчання персептрону. Ці дані включають ключові

показники системи, які допомагають моделі зрозуміти, як змінюється навантаження і які дії потрібно виконувати для її оптимізації.

Perceptron – це простий тип нейронної мережі, яка аналізує вхідні дані, такі як повідомлення із Service Bus або історичні дані з Blob Storage, і приймає рішення про необхідність масштабування ресурсів. На основі своїх розрахунків персептрон видає рекомендацію, чи потрібно збільшити або зменшити кількість ресурсів.

App Service Plan відповідає за реальне виконання масштабування. На основі рішення персептрону він збільшує або зменшує кількість доступних обчислювальних потужностей. Це дозволяє системі автоматично адаптуватися до змін навантаження, ефективно використовуючи ресурси.

Рисунок 4.5 – Інтерфейс App Service Plan

Azure Monitor – це інструмент для відстеження метрик продуктивності та моніторингу системи. У цій системі він використовується для спостереження за процесом масштабування і забезпечення прозорості в роботі всіх компонентів. Azure Monitor надає дані, які можна використовувати для аналізу та оптимізації продуктивності.

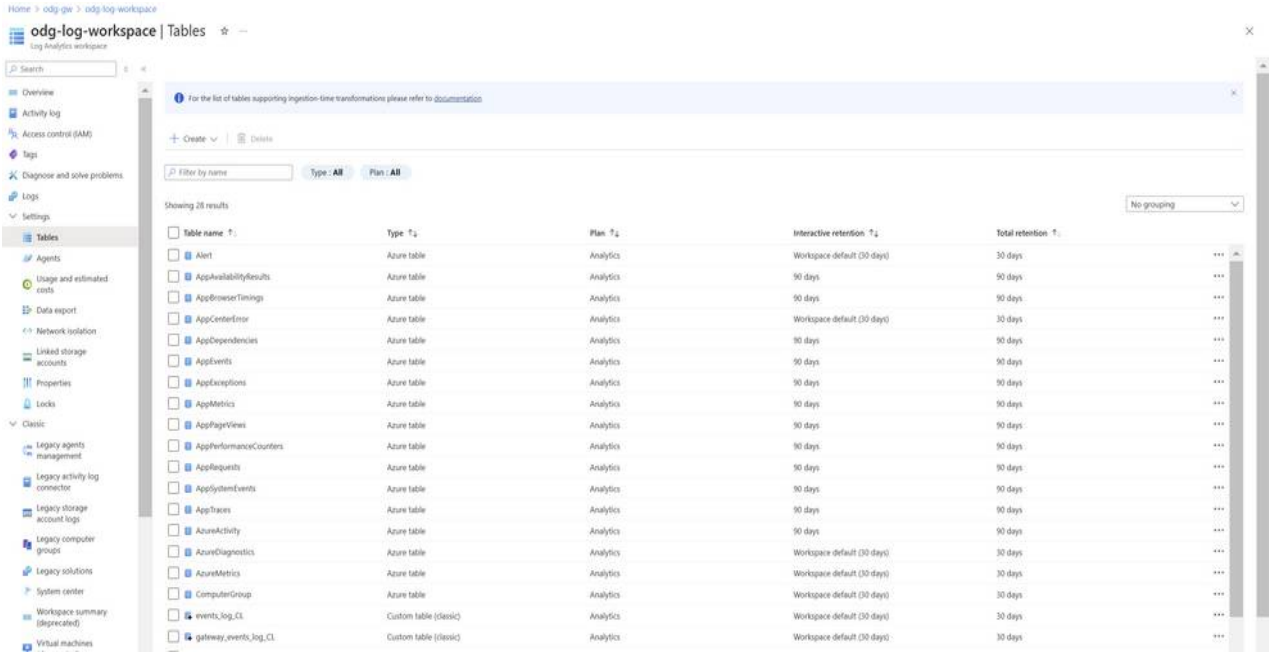

Рисунок 4.6 – Інтерфейс Azure Monitor

4.3 Взаємодія сервісів у розробленій архітектурі

У цій архітектурі взаємодія починається з Blob Storage, який зберігає історичні дані. Azure Functions отримують ці дані з Blob Storage через відповідний API або SDK, витягуючи метрики, такі як навантаження на систему, тенденції використання або історію роботи черг. Ці дані передаються до персептрону для навчання, щоб модель могла краще прогнозувати ситуації.

Service Bus надсилає поточні дані про стан системи, зокрема кількість повідомлень у черзі або навантаження в реальному часі. Ці дані також обробляються Azure Functions, які перетворюють їх у зручний формат для подальшого аналізу персептроном.

Дані з Training Data, що знаходяться в Blob Storage, комбінуються з даними реального часу із Service Bus. Azure Functions передає цю інформацію до персептрону, де модель використовує її для прийняття рішень. Наприклад, якщо в Service Bus кількість повідомлень у черзі перевищує допустимий поріг, персептрон вирішує збільшити ресурси.

Результати роботи персептрону у вигляді рекомендацій (наприклад, "збільшити кількість інстансів") надходять до App Service Plan, який відповідає за виконання масштабування. Azure Functions виступає посередником, передаючи рішення персептрону в App Service Plan через API.

На фінальному етапі Azure Monitor збирає метрики зі всіх компонентів системи, таких як Blob Storage, Service Bus, App Service Plan і навіть з самого персептрону. Дані про масштабування, час обробки запитів і продуктивність фіксуються в Azure Monitor, що дозволяє аналізувати загальний стан системи.

Таким чином, процес виглядає так: дані з Blob Storage і Service Bus обробляються Azure Functions, передаються до персептрону для аналізу, потім персептрон передає своє рішення до App Service Plan, а результати дій відстежуються за допомогою Azure Monitor. Усі ці компоненти працюють у зв'язці, забезпечуючи динамічне масштабування та оптимізацію ресурсів.

4.4 Реалізація інтегрування нейроної мережі до коду функції автоскейлу

Таким чином, враховуючи всі перераховані компоненти та їх взаємодію, ми прийшли до такого варіанту реалізації: система будується на інтеграції нейронної мережі, яка дозволяє обробляти дані з різних джерел, приймати рішення на основі отриманих метрик та забезпечувати динамічне масштабування. Використання даних із Blob Storage для навчання моделі, актуальних показників із Service Bus та подальшої автоматизації прийняття рішень допомагає забезпечити стабільність і гнучкість у роботі системи.

Лістинг 4.2 – Код функції із інтегрованим персептроном та іншими сервісами у архітектурі проекту

```
import logging
import os
import subprocess
import numpy as np
```

```
    from azure.functions import HttpRequest, HttpResponse
    from azure.servicebus.management import
ServiceBusAdministrationClient
    from azure.storage.blob import BlobServiceClient

    # Функція активації
    def activation_function(x):
        return 1 if x >= 0 else -1

    # Навчання персептрону
    def train_perceptron(training_data, learning_rate, epochs):
        weights = np.random.rand(len(training_data[0]) - 1)
        bias = np.random.rand(1)

        for _ in range(epochs):
            for sample in training_data:
                x = np.array(sample[:-1])
                expected = sample[-1]
                result = activation_function(np.dot(weights, x) +
bias)
                error = expected - result
                weights += learning_rate * error * x
                bias += learning_rate * error

        return weights, bias

    def predict(weights, bias, x):
        return activation_function(np.dot(weights, x) + bias)

    def main(req: HttpRequest) -> HttpResponse:
        logging.info('Azure Function processing a request.')

        try:
            # Загружаем переменные окружения
            blob_connection_str =
os.getenv('BLOB_CONNECTION_STRING')
```

```
            blob_container_name =
os.getenv('BLOB_CONTAINER_NAME')
            blob_file_name = os.getenv('BLOB_FILE_NAME')

            service_bus_connection_str =
os.getenv('SERVICE_BUS_CONNECTION_STRING')
            topic_name = os.getenv('SERVICE_BUS_TOPIC_NAME')
            subscription_name =
os.getenv('SERVICE_BUS_SUBSCRIPTION_NAME')

            resource_group_name = 'odg-gw'
            app_service_plan_name = 'odg-service-plan-vp'

            min_instance_count =
int(os.getenv('MIN_INSTANCE_COUNT'))
            max_instance_count =
int(os.getenv('MAX_INSTANCE_COUNT'))

            # Підключення до сховища даних
            blob_service_client =
BlobServiceClient.from_connection_string(blob_connection_str)
            blob_client =
blob_service_client.get_blob_client(container=blob_container_name,
blob=blob_file_name)

            training_data = []
            blob_data =
blob_client.download_blob().readall().decode('utf-8')
            for line in blob_data.splitlines():
                values = list(map(int, line.split(',')))
                training_data.append(values)

            logging.info(f'Training data loaded:
{training_data}')

            # Навчання персептрона
```

```
            learning_rate = 0.01
            epochs = 1000
            weights, bias = train_perceptron(training_data,
learning_rate, epochs)



            servicebus_admin_client =
ServiceBusAdministrationClient.from_connection_string(service_bus_
connection_str)
            subscription_properties =
servicebus_admin_client.get_subscription_runtime_properties(topic_
name, subscription_name)
            message_count =
subscription_properties.total_message_count

            logging.info(f'Message count in subscription:
{message_count}')

            get_instances_command = f'az appservice plan show --
name {app_service_plan_name} --resource-group
{resource_group_name} --query sku.capacity --output tsv'
            current_instance_count_result = subprocess.run(
                get_instances_command, shell=True, check=True,
text=True, capture_output=True
            )
            current_instance_count =
int(current_instance_count_result.stdout.strip())

            decision = predict(weights, bias,
np.array([message_count]))

            if decision == 1:
                new_instance_count = min(current_instance_count +
1, max_instance_count)
                action = "Scale out"
```

```
            else:
                new_instance_count = max(current_instance_count -
1, min_instance_count)
                action = "Scale in"

            if new_instance_count != current_instance_count:
                update_instances_command = f'az appservice plan
update --name {app_service_plan_name} --resource-group
{resource_group_name} --number-of-workers {new_instance_count}'
                subprocess.run(update_instances_command,
shell=True, check=True)
                logging.info(f'{action}. Updated instance count
to {new_instance_count}.')
            else:
                logging.info('Instance count unchanged. No
scaling performed.')



            return HttpResponse(f'{action}. Instance count:
{new_instance_count}. Message count: {message_count}.',
status_code=200)



        except Exception as e:
            logging.error(f'Error: {e}')
            return HttpResponse(f'Error: {e}', status_code=500)
```

Результати порівняльного аналізу традиційних функцій та функцій із інтегрованим персептроном наведені у табл. 4.2. Основними критеріями порівняння були обрані гнучкість, масштабованість, прогнозування, складність обробки та простота інтеграції.

Таблиця 4.2 – Порівняння традиційних функцій та функцій із інтегрованим персептроном

Критерії	Звичайна функція	Функція із інтегрованим персептроном
Гнучкість	Неможливість адаптації до нових даних без ручного втручання	Адаптація до змін даних у реальному часі
Масштабованість	Вимоги до точного налаштування формул і правил	Можливість навчання на великих наборах даних
Прогнозування	Складність обробки великої кількості змінних	Покращене прогнозування та прийняття рішень
Складність обробки	Лінійність у прийнятті рішень, що обмежує точність	Автоматичне врахування складних залежностей між даними
Простота інтеграції	Висока вірогідність помилок через людський фактор	Зменшення потреби у жорсткому кодуванні правил

Таким чином, у цьому розділі було продемонстровано, як за допомогою інтеграції різних сервісів Azure та використання нейронної мережі можна досягти більш ефективного управління ресурсами. Використання Blob Storage для зберігання історичних даних забезпечує стабільну основу для навчання персептрону, що дозволяє адаптуватися до змін у поведінці системи та прогнозувати необхідність масштабування. Інтеграція із Service Bus додає можливість отримувати актуальні метрики в реальному часі, що робить систему більш чутливою до динамічних змін.

Завдяки реалізації нейронної мережі вдалося значно зменшити залежність від статичних правил прийняття рішень, адже система здатна автоматично навчатися та враховувати нові фактори. Це, у свою чергу, забезпечує більш гнучкий та адаптивний підхід до управління навантаженням. Описаний підхід дозволяє не лише оптимізувати використання ресурсів, але й мінімізувати ризики людських помилок, оскільки прийняття рішень передано на рівень автоматизації. Додатково, використання Azure Monitor забезпечує постійний моніторинг продуктивності, що дає змогу виявляти та вирішувати потенційні проблеми до того, як вони вплинуть на систему. Впровадження нейронної мережі у поєднанні з сервісами Azure стало ключовим рішенням для побудови системи, яка не лише відповідає поточним потребам, але й готова до масштабування в майбутньому.

ВИСНОВКИ

В результаті роботи було досліджено методи автоматизації для ефективного управління IT-інфраструктурою, зокрема, шляхом автоскейлінгу, оркестрації контейнерів, моніторингу та логування, а також роботизованої автоматизації процесів (RPA). Використання цих методів забезпечило б стабільну роботу додатків, оптимізацію використання ресурсів, підвищення надійності сервісів та зменшило б кількості рутинних завдань для співробітників. Це спрямовано на комплексне дослідження та впровадження методів автоматизації, що дозволить підвищити ефективність управління IT-інфраструктурою та оптимізувати бізнес-процеси.

У майбутньому запропоновану модель одношарового перцептрона для автоматичного масштабування можна буде ускладнити за рахунок аналізу більшої кількості метрик і логів, що дозволить приймати більш точні та обґрунтовані рішення. Наприклад, можна враховувати не лише кількість повідомлень, а й інші показники, такі як завантаження процесора, використання пам'яті, кількість активних користувачів і час відгуку системи. Інтеграція даних логів може допомогти виявляти патерни і аномалії, що додатково підвищить точність рішень щодо масштабування.

Використання алгоритмів машинного навчання та штучного інтелекту дозволить системі самостійно навчатися на основі історичних даних і постійно покращувати свої рішення, адаптуючись до змін у навантаженні та поведінці користувачів.

Таким чином, початкове впровадження одношарового перцептрона є відмінним стартом для автоматизації процесів масштабування. Згодом, завдяки аналізу метрик і логів та використанню більш складних моделей, можна досягти ще вищого рівня автоматизації та ефективності управління ресурсами, що забезпечить стабільну роботу додатків навіть в умовах змінного навантаження.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ilyunin O., Khodak M., Pyrohov V., Pasholoc O., Tryhuba M., Serdiuk S. Intelligent models for control of jet hydro-processing of rolled steel defects // Інтегровані технології та енергозбереження. Системи управління та обробки інформації. – Харків: НТУ «ХПІ», 2023. – №2. – С. 45–56. – ISSN 2078-5364 (print), ISSN 2708-0625 (online).

2. Serdiuk S. Integration of big data processing using neural networks in Cloud computing // XVI Всеукраїнська науково-практична WEB-конференція аспірантів, студентів та молодих вчених «Комп’ютерні інтелектуальні системи та мережі» (KICM-2024), 26–28 березня 2024 р., Кривий Ріг, Україна. – С. 134–136.

3. Сердюк С.С. Інтеграція обробки великих даних за допомогою нейромереж у хмарні обчислення // 28-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті», т. 5, 16–18 квітня 2024 р., Харків, Україна. – С. 66–68.

4. Taherizadeh S., Stankovski V. Dynamic multi-level autoscaling rules for containerized applications // Computer Journal. – 2018. – Vol. 62. – P. 174–197.

5. Zhang F., Tang X., Li X., Khan S.U., Li Z. Quantifying cloud elasticity with container-based autoscaling // Future Generation Computer Systems. – 2019. – Vol. 98. – P. 672–681.

6. Kan C. Docloud: an elastic cloud platform for web applications based on Docker // Proceedings of the 2016 18th International Conference on Advanced Communication Technology (ICACT). – 2016. – P. 1.

7. Li Y., Xia Y. Auto-scaling web applications in hybrid cloud based on Docker // Proceedings of the 2016 5th International Conference on Computer Science and Network Technology (ICCSNT). – 2016. – P. 75–79.

8. Klinaku F., Frank M., Becker S. Caus: an elasticity controller for a containerized microservice // Companion of the 2018 ACM/SPEC international

conference on performance engineering, ICPE ’18, New York: ACM, 2018. – P. 93–98.

9. Amazon Web Services AWS Auto Scaling Documentation [Електронний ресурс]. – Режим доступу: https://aws.amazon.com (дата звернення: 05.01.2025).

10. Microsoft Azure Documentation Multitenancy and Data Isolation Guidelines for Cloud Computing [Електронний ресурс]. – Режим доступу: https://azure.microsoft.com (дата звернення: 05.01.2025).

11. Docker Best Practices for Containerized Multitenancy [Електронний ресурс]. – Режим доступу: https://www.docker.com (дата звернення: 05.01.2025).

12. Red Hat Introduction to Kubernetes for Multitenant Applications [Електронний ресурс]. – Режим доступу: https://www.redhat.com (дата звернення: 05.01.2025).

13. Google Cloud Best Practices for Multitenant Architectures [Електронний ресурс]. – Режим доступу: https://cloud.google.com (дата звернення: 05.01.2025).

14. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. – 2015. – Vol. 521, No. 7553. – P. 436–444.

15. Goodfellow I., Bengio Y., Courville A. Deep Learning. – Cambridge, MA: MIT Press, 2016. – 775 p.

16. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2016. – P. 770–778.

17. GCP Documentation Autoscaling Best Practices [Електронний ресурс]. – Режим доступу: https://cloud.google.com/docs (дата звернення: 05.01.2025).

18. Kubernetes Documentation Horizontal Pod Autoscaler [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale (дата звернення: 05.01.2025).

19. IBM Cloud Multitenancy in Cloud Computing: Solutions and Approaches [Електронний ресурс]. – Режим доступу: https://www.ibm.com/cloud (дата звернення: 05.01.2025).