

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО"
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО"
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

Кваліфікаційна наукова
праця на правах рукопису

КАСЕРЕС АНТОН

УДК 004.78

ДИСЕРТАЦІЯ


**КОМПЛЕКСНИЙ ПІДХІД ПРОЄКТУВАННЯ ОБЧИСЛЮВАЛЬНОЇ
ІНФРАСТРУКТУРИ У ГЕТЕРОГЕННОМУ МУЛЬТИХМАРНОМУ
СЕРЕДОВИЩІ**

Спеціальність 172 – Телекомунікації та радіотехніка

Галузь знань 17 – Електроніка та телекомунікації

Подається на здобуття наукового ступеня доктора філософії

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

 Касерес А.

Науковий керівник: Глоба Л.С. доктор технічних наук, професор

Київ – 2025

АНОТАЦІЯ

Касерес А. Комплексний підхід проєктування обчислювальної інфраструктури у гетерогенному мультимарному середовищі. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 172 – Телекомунікації та радіотехніка. – Навчально-науковий інститут телекомунікаційних систем КПІ ім. Ігоря Сікорського, Київ, 2025.

Інтенсивний розвиток хмарних обчислень трансформував процеси зберігання та обробки даних, пропонуючи масштабованість, економічну ефективність та доступність. Потреба у високоефективних засобах розподілу обчислень у гетерогенних хмарних інфраструктурах є надзвичайно актуальною. Традиційні підходи до розподілу обчислювальних завдань у хмарних середовищах часто не здатні адекватно вирішити виклики динамічної природи хмарних середовищ, що призводить до неефективного використання ресурсів, підвищення витрат та потенційних вразливостей. Дана дисертаційна робота представляє комплексне дослідження сучасних мультимарних архітектур, методів багатокритеріальної оптимізації та інтелектуальних систем прийняття рішень, що завершується розробкою інноваційного комплексного методу формування обчислювальної інфраструктури у гетерогенному мультимарному середовищі.

Мета дослідження полягає у підвищенні рівня автоматизації, достовірності та об'єктивності оцінювання варіантів розгортання мультимарної архітектури для обробки обчислювальних завдань у багатомарному середовищі, яка забезпечує адаптацію до динамічно змінюваного набору критеріїв функціонування системи.

Особливу увагу в роботі приділено ролі хмарної обчислювальної інфраструктури у сучасних інформаційно-комунікаційних мережах (ІКМ). Запропонований підхід демонструє, що мультимарні архітектури є ключовими для

забезпечення масштабованості, гнучкості та надійності обчислень, необхідних для функціонування телекомунікаційних систем нового покоління, зокрема у середовищі 5G. Сформульовано та реалізовано метод динамічного розподілу обчислювальних задач у мультихмарному середовищі з урахуванням специфіки телекомунікаційних навантажень, що дозволяє підвищити ефективність управління ресурсами в ІКМ.

Мультихмарні обчислення стали домінуючим підходом для розгортання та надання послуг через глобальні мережі. Їх переваги сприяють широкому впровадженню в різних галузях. Однак еволюція хмарних обчислень, що характеризується різноманітністю сервісів, несумісністю інтерфейсів та варіативністю умов розгортання, створює значні виклики в управлінні обчислювальними процесами. Динамічна природа хмарних середовищ, поєднана з множиною взаємозалежних факторів, що впливають на продуктивність та економічну ефективність, потребує інноваційних підходів до оптимізації розподілу обчислювальних завдань та вибору хмарних провайдерів.

Організації все частіше впроваджують стратегії використання кількох хмарних провайдерів для отримання конкурентних переваг та стикаються з викликами забезпечення продуктивності, оптимізації витрат, безпеки даних та ефективного управління ресурсами. Складність виникає з необхідності балансувати між кількома цілями, такими як мінімізація затримок, максимізація пропускної здатності, забезпечення надійності системи та дотримання регуляторних вимог, при цьому адаптуючись до динамічних змін у параметрах хмарних сервісів.

Завдання дослідження: провести аналіз сучасних концепцій і методів вибору хмарних сервісів у багатохмарних середовищах; визначити набір критеріїв для проєктування віртуальної мультихмарної платформи; дослідити доцільність застосування LLM-моделей для підтримки прийняття рішень; визначити інтегрований показник ефективності сформованої віртуальної мультихмарної

архітектури; розробити прототип системи для інтегрованої оцінки ефективності виконання обчислювальних завдань; провести комплексну оцінку ефективності розробленого підходу.

Наукова новизна полягає у тому, що вперше запропоновано комплексний метод формування обчислювальної інфраструктури у гетерогенному мультитимарному середовищі, який використовує інтегрований показник ефективності виконання обчислювальних завдань і поєднує переваги методів навчання з підкріпленням та багатокритеріальних еволюційних алгоритмів; запропоновано математичну модель мультитимарної взаємодії обчислювальних сервісів у динамічно змінюваному середовищі; розроблено метод визначення інтегрованого показника ефективності з використанням LLM моделі для аналізу стану обчислювального середовища; запропоновано метод визначення складових інтегрованого показника ефективності, який відрізняється використанням LLM для багатокритеріального аналізу.

В результаті виконання запропоновано наступні рішення: комплексний метод щодо формування обчислювальної інфраструктури у гетерогенному мультитимарному середовищі з використанням інтегрованого показника ефективності; математичну модель інтегрованого показника ефективності виконання обчислювальних завдань; мультитимарну систему підтримки прийняття рішень на основі LLM; алгоритм оптимізації архітектури мультитимарної інфраструктури; модель динамічної адаптації до зміни параметрів середовища; програмний прототип системи автоматизованого проектування мультитимарної платформи.

Методологія дослідження охоплює всебічний аналіз існуючих підходів до мультитимарної інтеграції, формалізацію критеріїв ефективності, розробку архітектури мультитимарної системи та експериментальну перевірку запропонованого методу в реальних сценаріях використання. Мультитимарна

система включає технічного, економічного та безпекового агентів, координацію яких забезпечує спеціалізований агент-координатор, що використовує LLM для обґрунтування прийнятих рішень та формування рекомендацій.

Практична цінність полягає у створенні комплексного підходу щодо проєктування віртуальної мультимарної платформи, який підвищує адаптивність та ефективність виконання обчислювальних процесів; розробці програмного забезпечення, що автоматизує процеси проєктування і переналаштування віртуальної мультимарної платформи; впровадженні розробленої системи у виробничому середовищі, що підтвердило зниження витрат у середньому на 47% та покращення продуктивності на 25%; успішному практичному впровадженні запропонованого методу в корпоративному середовищі Continental Automotive Technologies GmbH (Німеччина) в межах проєктів PRIME-MES, де підхід був застосований для оптимізації хмарної інфраструктури, контролю витрат на зберігання даних у data lake та розподілу робочого навантаження, що забезпечило підвищення продуктивності, стійкості та ефективності хмарних операцій на рівні підприємства; розробці програмних рішень, які інтегровано в системи управління обчислювальними ресурсами для покращення гнучкості, підвищення рівня автоматизації та швидкості розгортання інфраструктури; впровадженні розроблених підходів у навчальні та дослідницькі програми для підготовки фахівців у сфері хмарних обчислень та автоматизації процесів проєктування розподілених систем у мультимарному середовищі.

Ключові слова: мультимарні обчислення, гетерогенні середовища, мультимарне середовище, динамічний розподіл обчислень, мультиагентні системи, великі мовні моделі, управління хмарами, багатокритеріальна оптимізація, штучний інтелект, якість обслуговування, угоди про рівень обслуговування.

ABSTRACT

Caceres A. A comprehensive approach to designing computational infrastructure in a heterogeneous multi-cloud environment. – Qualifying scientific work on manuscript rights.

Thesis for graduation scientific degree of Philosophy Doctor by specialty 172 – Telecommunications and radio engineering. – Educational and Scientific Institute of Telecommunication Systems of KPI named after Igor Sikorsky, Kyiv, 2025.

The rapid development of cloud computing has transformed data storage and processing, offering scalability, economic efficiency, and increased accessibility. The need for highly efficient computation distribution methods in heterogeneous cloud infrastructures is hugely relevant. Traditional approaches to distributing computational tasks in cloud environments often fail to adequately address the challenges posed by the dynamic nature of cloud environments, leading to inefficient resource utilization, increased costs, and potential vulnerabilities. This dissertation presents a comprehensive study of modern multi-cloud architectures, multi-criteria optimization methods, and intelligent decision-making systems, culminating in the development of an innovative integrated method for forming a computational infrastructure in a heterogeneous multi-cloud environment.

The research aims to enhance the automation, reliability, and objectivity of evaluating deployment options for multi-cloud architecture in the processing of computational tasks within a multi-cloud environment, ensuring adaptation to a dynamically changing set of system operation criteria.

Special attention is given in this work to the role of cloud computing infrastructure in modern information and communication networks (ICNs). The proposed approach demonstrates that multi-cloud architectures are crucial for providing the scalability, flexibility, and reliability of computations required for the operation of next-generation

telecommunications systems, particularly in 5G environments. A method for dynamically distributing computational tasks in multi-cloud environments, considering the specifics of telecommunications loads, is formulated and implemented, thereby improving resource management efficiency in ICNs.

Multi-cloud computing has become the dominant approach for deploying and delivering services over global networks. Its advantages contribute to widespread adoption across various sectors. However, the evolution of cloud computing, characterized by service heterogeneity, interface incompatibility, and deployment variability, creates significant challenges in managing computational processes. The dynamic nature of cloud environments, combined with a multitude of interdependent factors that affect performance and economic efficiency, requires innovative approaches to optimizing computational task distribution and selecting the most suitable cloud provider.

The problem of computation distribution in multi-cloud environments is multifaceted. As organizations increasingly adopt strategies using multiple cloud providers to gain competitive advantages, they face challenges in ensuring performance, optimizing costs, securing data, and managing resources efficiently. The complexity arises from the need to balance multiple objectives, such as minimizing latency, maximizing throughput, ensuring system reliability, and complying with regulatory requirements while adapting to dynamic changes in cloud service parameters.

Research objectives include analyzing modern concepts and methods for selecting cloud services in multi-cloud environments; defining a set of criteria for designing a virtual multi-cloud platform; exploring the feasibility of applying LLM models to support decision-making; determining the integrated performance indicator of the formed virtual multi-cloud architecture; designing and creating a prototype system for integrated assessment of computational task performance efficiency; and conducting a comprehensive evaluation of the effectiveness of the developed approach.

The scientific novelty lies in the fact that, for the first time, a comprehensive method for forming computational infrastructure in a heterogeneous multi-cloud environment is proposed, which uses an integrated performance indicator for computational task execution and combines the advantages of reinforcement learning and multi-criteria evolutionary algorithms; a mathematical model of multi-cloud interaction of computational services in a dynamically changing environment is proposed; a method for determining the integrated performance indicator using an LLM model to analyze the state of the computational environment is developed; and a method for determining the components of the integrated performance indicator is proposed, which differs by using LLM for multi-criteria analysis.

As a result of the work, the following solutions are proposed: a comprehensive method for forming computational infrastructure in a heterogeneous multi-cloud environment using an integrated performance indicator; a mathematical model of the integrated performance indicator for computational task execution; a multi-agent decision-support system based on LLM; an architecture optimization algorithm for multi-cloud infrastructure; a model for dynamic adaptation to changing environmental parameters; and a software prototype of a system for automated multi-cloud platform design.

The research methodology encompasses a comprehensive analysis of existing approaches to multi-cloud integration, the formalization of efficiency criteria, the development of multi-agent system architecture, and the experimental validation of the proposed method in real-world usage scenarios. The multi-agent system comprises technical, economic, and security agents, all of which are coordinated by a specialized coordinator agent that utilizes an LLM to justify decisions and formulate recommendations.

The practical value lies in the creation of a comprehensive approach to designing a virtual multi-cloud platform, which enhances adaptability and efficiency in computational

processes; the development of software that automates the design and reconfiguration processes of the virtual multi-cloud platform; the implementation of the developed system in a production environment, which confirmed a 47% average cost reduction and a 25% improvement in performance; the successful practical implementation of the proposed method in the corporate environment of Continental Automotive Technologies GmbH (Germany) within the PRIME-MES platform projects, where the approach was applied to optimize cloud infrastructure, control data storage costs in the data lake, and distribute workloads, leading to increased productivity, resilience, and efficiency of cloud operations at the enterprise level; the development of software solutions integrated into computational resource management systems to improve flexibility, increase automation levels, and speed up infrastructure deployment; the incorporation of the developed approaches into educational and research programs for preparing specialists in cloud computing and the automation of distributed system design processes in multi-cloud environments.

Keywords: multi-cloud computing, heterogeneous environments, multi-cloud environment, dynamic task allocation, multi-agent systems, large language models, cloud management, multi-criteria optimization, artificial intelligence, quality of service, service level agreements.

СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові результати дисертації

1. Caceres, A., & Globa, L. (2025). AHP-based multi-criteria analysis of multi-cloud data management techniques. *Radioelectronic and Computer Systems*, 1, Article 04. <https://doi.org/10.32620/reks.2025.1.04>.
2. Caceres, A., & Globa, L. (2025). Аналіз підходів доступу до даних у мульти-клауд середовищі [Analysis of data access approaches in a multi-cloud environment]. *Radio Electronics, Computer Science, Control*, 1. Retrieved from <https://ric.zp.edu.ua/article/view/324539/314660>.
3. Caceres, A., & Globa, L. (2024). Підхід до формування мултихмарного середовища на основі багатокритеріального аналізу та онтологічного моделювання [A multi-criteria and ontology-based approach to multi-cloud environment selection]. *Системи управління, навігації та зв'язку*, 4. <https://doi.org/10.26906/SUNZ.2024.4.084>.
4. Caceres, A., & Globa, L. (2024). Інтеграція ШІ та мултиагентних систем для багатокритеріального аналізу у мултихмарному середовищі [Integration of AI and multi-agent systems for multi-criteria analysis in a multicloud environment]. *Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки*, 35(74), № 6, Частина 2. Retrieved from https://www.tech.vernadskyjournals.in.ua/journals/2024/6_2024/part_2/12.pdf.
5. Caceres, A., & Globa, L. (2022). State-of-the-art architectures for interoperability of heterogeneous clouds. In *Proceedings of the 2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)* (pp. 704–709). IEEE. <https://doi.org/10.1109/TCSET55632.2022.9766965>.

ЗМІСТ

АНОТАЦІЯ.....	2
ABSTRACT.....	6
СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ.....	10
СПИСОК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ	16
ВСТУП.....	18
РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМ ТА АКТУАЛЬНІ ПІДХОДИ ДО ПРОЄКТУВАННЯ ОБЧИСЛЮВАЛЬНОЇ ІНФРАСТРУКТУРИ У ГЕТЕРОГЕННИХ МУЛЬТИХМАРНИХ СЕРЕДОВИЩАХ.....	26
1.1 Особливості обчислювального навантаження та вимоги щодо проєктування мультимарного середовища	26
1.2 Організація мультимарного середовища та підтримка його функціонування	32
1.2.1 Особливості мультимарної взаємодії	38
1.3 Переваги та обмеження рішень щодо мультимарної взаємодії.....	40
1.4 Сумісність обчислень у мультимарному середовищі та її вплив на гетерогенні мультимарні системи.....	45
ВИСНОВКИ	48
РОЗДІЛ 2 ФОРМАЛІЗАЦІЯ ПРОЦЕСУ ВЗАЄМОДІЇ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ У ГЕТЕРОГЕННОМУ МУЛЬТИХМАРНОМУ СЕРЕДОВИЩІ	49
2.1 Критерії оцінювання ефективності мультимарної взаємодії.....	49
2.2 Математична модель мультимарного середовища	50
2.3 Цільові функції оптимізації та їхній зв'язок з критеріями ефективності...	51
2.4 Обмеження, які накладаються на параметри моделі мультимарної взаємодії.....	53
2.5 Класифікація рішень побудови мультимарного середовища.....	54
2.6 Формальний опис критеріїв визначення сумісності сервісів різних провайдерів мультимарного середовища та способів їх взаємодії	56
2.7 Проблематика сумісності у мультимарному середовищі	58
2.8 Оцінка ефективності рішень щодо побудови мультимарного середовища	

2.8.1 Інтегрований показник оцінки ефективності архітектурних рішень	60
2.8.2 Нормалізація компонентів моделі.....	60
2.8.3 Стохастична модель динамічної адаптації вагових коефіцієнтів	61
2.8.4 Багатокритеріальна оптимізація при проєктуванні мультихмарної інфраструктури	62
2.8.5 Стратегія взаємодії сервісів при проєктуванні мультихмарної інфраструктури	63
ВИСНОВКИ	64
РОЗДІЛ 3 МЕТОДОЛОГІЯ ПРОЄКТУВАННЯ АРХІТЕКТУРИ ОБЧИСЛЮВАЛЬНОЇ ІНФРАСТРУКТУРИ У ГЕТЕРОГЕННОМУ МУЛЬТИХМАРНОМУ СЕРЕДОВИЩІ.....	65
3.1 Постановка задачі	65
3.1.1 Формалізація проблеми вибору архітектури мультихмарної інфраструктури	67
3.1.2 Математична модель вибору архітектури для мультихмарного розподілу задач	67
3.2 Проєктування взаємодії сервісів у мультихмарному середовищі.....	69
3.2.1 Формальна модель динамічної адаптації мультихмарної інфраструктури	71
3.3 Проєктування мультихмарної архітектури.....	72
3.3.1 Основні компоненти системи	74
3.3.2 Взаємодія між компонентами	75
3.4 Метод багатокритеріального аналізу для автоматизованого вибору оптимальної хмарного інструментарію	76
3.4.1 Формалізація процесу вибору інструментарія мульти-хмарної взаємодії	80
3.4.2 Побудова моделі АНР.....	80
3.4.3 Онтологічна модель для побудови мультихмарного середовища.....	83
3.5 Моделі машинного навчання та LLM для побудови мультихмарного середовища	85
3.5.1 Алгоритм роботи LLM.....	87

	13
3.5.2 Мультиагентна система на основі LLM.....	93
ВИСНОВКИ	96
РОЗДІЛ 4 КОМПЛЕКСНИЙ МЕТОД ДИНАМІЧНОГО ФОРМУВАННЯ ОБЧИСЛЮВАЛЬНОЇ ІНФРАСТРУКТУРИ У МУЛЬТИХМАРНОМУ СЕРЕДОВИЩІ	97
4.1 Гібридна структура проєктування мультихмарної інфраструктури	97
4.1.1 Математична модель інтегрованого показника ефективності виконання обчислювальних завдань	97
4.1.2 Формалізація простору станів та дій для проєктування мультихмарної інфраструктури	100
4.1.3 Функції винагороди на основі багатокритеріальної оптимізації.....	101
4.1.4 Механізми адаптації та навчання	103
4.2 Комплексний метод формування обчислювальної інфраструктури.....	104
4.2.1 Архітектура мультиагентної системи підтримки прийняття рішень .	105
4.2.2 Алгоритм оптимізації архітектури мультихмарної інфраструктури..	110
4.2.3 Модель динамічної адаптації до зміни параметрів середовища.....	113
4.3 Побудова агентів на основі LLM для підтримки прийняття рішень	115
4.3.1 Процес взаємодії LLM-агентів у мультихмарному середовищі.....	117
4.3.2 Алгоритм узгодження висновків агентів та формування остаточного рішення	120
4.3.3 Сценарії використання мультиагентної системи	124
ВИСНОВКИ	133
РОЗДІЛ 5 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНОГО ПІДХОДУ	135
5.1 Реалізація прототипу інтелектуальної системи для проєктування мультихмарної інфраструктури.....	135
5.1.1 Архітектура та компоненти програмної реалізації	135
5.1.2 Реалізація програмного компоненту підтримки прийняття рішень...	140
5.1.3 Інтеграція методів машинного навчання та LLM	142
5.2 Методика експериментального дослідження	144

5.2.1	Визначення умов експерименту	145
5.2.2	Вибір та обґрунтування метрик оцінювання	146
5.2.3	Опис експериментального середовища та тестових сценаріїв	147
5.3	Експериментальна перевірка ефективності запропонованого підходу	148
5.3.1	Сценарій 1. Розгортання web-застосунку з REST API	149
5.3.2	Сценарій 2. ETL-система обробки даних.....	151
5.4	Практичне впровадження та апробація в промисловому середовищі	153
5.4.1	Опис умов та процесу впровадження.....	153
5.4.2	Результати практичного використання	154
5.5	Аналіз результатів та рекомендації щодо подальшого розвитку	158
5.5.1	Узагальнення результатів експериментального дослідження	158
5.5.2	Виявлені обмеження та шляхи їх подолання.....	162
5.5.3	Рекомендації щодо масштабування та вдосконалення системи.....	164
ВИСНОВКИ		166
ЗАГАЛЬНІ ВИСНОВКИ.....		167
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....		170
Додаток 1.	Приклади реалізації агентів.....	181
Додаток 2.	Промпт-шаблон для технічного агента	192
Додаток 3.	Архітектура інтеграції LLM	193
Додаток 4.	Аналіз вхідних вимог, виконаний агентом-координатором	194
Додаток 5.	Визначення можливих варіантів розгортання технічним агентом 195	
Додаток 6.	Аналіз вартості різних варіантів розгортання економічним агентом 197	
Додаток 7.	Аналіз відповідності рішень вимогам безпеки, виконаного безпековим агентом.....	199
Додаток 8.	Агрегація агентом-координатором результатів та формулювання фінальної рекомендації	201
Додаток 9.	Аналіз вхідних вимог, виконаний агентом-координатором (сценарій 2)	203

Додаток 10.	Формулювання рекомендацій технічним агентом	204
Додаток 11.	Аналіз вартості різних варіантів розгортання економічним агентом 206	
Додаток 12.	Аналіз відповідності рішень вимогам безпеки, виконаного безпековим агентом	208
Додаток 13.	Агрегація агентом-координатором результатів та формулювання фінальної рекомендації	210
Додаток 14.	Аналіз технічного агента	212
Додаток 15.	Аналіз економічного агента	214
Додаток 16.	Аналіз безпекового агента	217
Додаток 17.	Процес прийняття рішення агентом-координатором	219
Додаток 18.	Безсерверна архітектура	222
Додаток 19.	Аналіз відгуків фахівців та експертних оцінок	223
Додаток 20.	Список публікацій за темою дисертації та відомості про апробацію результатів дисертації	224
Додаток 21.	Акт впровадження	226

СПИСОК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

AHP	Analytic Hierarchy Process
AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
CAL	Cloud-Agnostic Libraries
CDMI	Cloud Data Management Interface
CI/CD	Continuous Integration / Continuous Deployment (або Delivery)
DMP	Data Management Platforms
EC	Edge Computing
ETL	Extract, Transform, Load
FaaS	Functions as a Service
FIS	Fuzzy Inference Systems
GCP	Google Cloud Platform
GDPR	General Data Protection Regulation
IaaS	Infrastructure as a Service
IAM	Identity and Access Management
IaC	Infrastructure as Code
LLM	Large Language Model
MAS	Multi-Agent System
MCDA	Multi-Criteria Decision Analysis
ML	Machine Learning
NFV	Network Functions Virtualization
OCCI	Open Cloud Computing Interface
PaaS	Platform as a Service
PCI DSS	Payment Card Industry Data Security Standard

QoS	Quality of Service
RAG	Retrieval-Augmented Generation
SaaS	Software as a Service
SDK	Software Development Kit
SDN	Software-Defined Networking
SG	Storage Gateways
SLA	Service Level Agreement
TOPSIS	Technique for Order of Preference by Similarity to Ideal Solution
UDDI	Universal Description, Discovery and Integration
VPC	Virtual Private Cloud
VPN	Virtual Private Network
ІКМ	Інформаційно-комунікаційні мережі
ПЗ	Програмне забезпечення
ІІІ	Штучний інтелект

ВСТУП

Актуальність теми. В умовах зростаючої складності інформаційних систем та дедалі зростаючих обсягів обчислювальних завдань, потреба у високоефективних засобах розподілу обчислень у гетерогенних хмарних інфраструктурах є надзвичайно актуальною [1-3], що зумовлює необхідність розроблення комплексних підходів до проектування гнучких мультихмарних інфраструктур, здатних ефективно розподіляти обчислювальні ресурси між різними провайдерами. Хмарні обчислення забезпечують зручний і масштабований доступ до обчислювальних потужностей, однак, зважаючи на суттєві відмінності між платформами, інтеграція та розподіл обчислень у багатохмарних середовищах залишається значним викликом. Практика свідчить, що користувачі здебільшого обмежені вибором одного провайдера, характеристики якого задовольняють їхнім вимогам на конкретному етапі, при цьому вимоги щодо продуктивності, економічної ефективності та доступності обчислювальних ресурсів зазнають динамічних змін протягом життєвого циклу розробки та експлуатації програмного забезпечення. Сучасні методи міграції та інтеграції між платформами часто потребують значних зусиль з боку фахівців та спеціально розробленого програмного забезпечення, що ускладнює та затримує процеси розробки [4-6].

Відсутність автоматизованих підходів для комплексного проектування обчислювальної інфраструктури, що забезпечують не лише ефективний розподіл обчислень, але й автоматизоване проектування архітектур взаємодії необхідних сервісів, обмежує гнучкість систем та підвищує вартість обслуговування. Дослідження та розробка нових методів, що дозволяють ефективно інтегрувати різні хмарні платформи для створення адаптивної та високопродуктивної інфраструктури, є критично важливими для подальшого розвитку галузі.

Зв'язок роботи з науковими програмами, планами, темами.

Дисертаційна робота виконувалась згідно з планом наукових досліджень кафедри інформаційно-комунікаційних технологій та систем Навчально-наукового Інституту телекомунікаційних систем:

- в рамках держбюджетної теми: 2218п «Гетерогенна мережа збору, передачі та обробки інформації для системи розподіленої генерації MicroGrid» (номер державної реєстрації 0119U001184);
- в рамках держбюджетної теми: 2313п «Побудова інформаційно-аналітичної платформи для супроводження функціонування кіберфізичних систем» (номер державної реєстрації 0120U102298);
- в рамках міжнародного проєкту “IDEA-East-Hub: International Innovation Hub for Data Science and renewable Energy”, DAAD Program - HAW.International Modul за співпрацею НТУУ «КПІ ім. І. Сікорського» та Анхальтським університетом прикладних наук Кьотен, Німеччина (номер договору 57603761).

Мета і задачі дослідження. Метою даної дисертаційної роботи є підвищення рівня автоматизації, достовірності та об'єктивності оцінювання варіантів розгортання мультихмарної архітектури для обробки обчислювальних завдань у багатохмарному середовищі, яка забезпечує адаптацію до динамічно змінюваного набору критеріїв функціонування системи. Для досягнення поставленої мети передбачено розв'язання таких наукових задач:

1. Провести аналіз сучасних концепцій і методів вибору хмарних сервісів для виконання заданих обчислень у багатохмарних середовищах, виявити їхні обмеження та критичні фактори.
2. Визначити набір критеріїв, які забезпечують проєктування за динамічно змінюваним набором критеріїв віртуальної мультихмарної платформи для виконання обчислень з урахуванням наявних параметрів середовища та випадкового характеру надходження обчислювального навантаження на обробку.

3. Дослідити доцільність застосування LLM-моделей для підтримки прийняття рішень у складних обчислювальних сценаріях у разі автоматизованого процесу визначення відповідності вимог щодо виконання обчислень високої інтенсивності з нерівномірним розподілом навантаження та потребою в масштабуванні ресурсів у реальному часі та ресурсів, що пропонуються у мультихмарному середовищі різними cloud-провайдерами.

4. Визначити інтегрований показник ефективності сформованої віртуальної мультихмарної архітектури для виконання обчислювальних завдань у мультихмарному середовищі, який враховує вплив кожного критерія в умовах динамічної зміни як вимог споживача, так і параметрів середовища.

5. Спроектувати та створити прототип системи, яка здатна здійснювати інтегровану оцінку очікуваної ефективності виконання обчислювальних завдань на вибраних хмарних платформах, а також здатна задовільнити динамічно змінюваний набір вимог кінцевого споживача щодо обчислювальної інфраструктури.

6. Провести комплексну оцінку ефективності розробленого підходу на основі експериментальних випробувань та аналізу їх результатів.

Об'єктом дослідження є процес проєктування за динамічно змінюваним набором вимог віртуальної хмарної платформи для виконання обчислень з урахуванням наявних параметрів середовища та випадкового характеру надходження обчислювального навантаження на обробку.

Предметом дослідження є методи визначення оптимальної стратегії проєктування за динамічно змінюваним набором критеріїв віртуальної хмарної платформи у гетерогенних хмарних середовищах на основі моделей та методів глибокого машинного навчання.

Методи дослідження, застосовані для вирішення поставлених завдань:

1. Методи системного аналізу та синтезу – для аналізу об'єкта дослідження, формалізації процесів проєктування віртуального хмарного середовища, виділення

ключових характеристик і взаємозв'язків між ними.

2. Методи багатокритеріального аналізу – для розробки та оцінки моделей прийняття рішень щодо вибору оптимальної хмарної платформи, що відповідає заданим критеріям ефективності, вартості та продуктивності.

3. Методи глибокого машинного навчання – для створення моделей підтримки прийняття рішень, що забезпечують динамічний вибір середовища для виконання обчислень з урахуванням змінних вимог до віртуальної хмарної платформи у мультихмарному середовищі.

4. Методи програмної інженерії та автоматизації – для розробки та реалізації прототипу системи автоматизованого проектування за динамічно змінюваним набором критеріїв віртуальної мультихмарної платформи за допомогою інструментів програмування та засобів автоматизації інфраструктури, інтеграції з API хмарних провайдерів.

5. Методи імітаційного моделювання та експериментального тестування – для перевірки ефективності запропонованих моделей і методів, їх порівняння з існуючими аналогами та аналізу отриманих результатів у реальних сценаріях використання.

Наукова новизна отриманих результатів:

1. Вперше запропоновано комплексний метод щодо формування обчислювальної інфраструктури у гетерогенному мультихмарному середовищі, який використовує інтегрований показник ефективності виконання обчислювальних завдань для визначення в автоматизованому режимі місця розташування і типу сервісів, які пропонуються хмарними провайдерами, поєднує переваги методів навчання з підкріпленням і багатокритеріальних еволюційних алгоритмів для пошуку ефективних рішень із урахуванням динаміки зміни як стану інфраструктури мультихмарного середовища так і вимог кінцевого користувача, що дозволяє підвищити продуктивність, захищеність та знизити витрати виконання

обчислювальних задач.

2. Запропоновано математичну модель мультихмарної взаємодії обчислювальних сервісів у динамічно змінюваному середовищі, яка враховує динаміку змін вимог кінцевого користувача та характеристик наявних хмарних ресурсів, що дозволяє автоматизувати переналаштування мультихмарної платформи.

3. Запропоновано метод визначення інтегрованого показника ефективності виконання обчислювальних завдань у динамічно змінюваному середовищі, який вирізняється врахуванням динаміки зміни як стану інфраструктури мультихмарного середовища так і вимог кінцевого користувача, що дозволяє автоматизувати процес проєктування та переналаштування віртуальної мультихмарної платформи.

4. Запропоновано метод визначення складових інтегрованого показника ефективності виконання обчислювальних завдань у динамічно змінюваному мультихмарному середовищі, який відрізняється використанням LLM моделі для аналізу стану обчислювального середовища за сукупністю критеріїв проєктування віртуальної мультихмарної платформи із динамічно змінюваним набором вимог, що дозволяє враховувати ступінь впливу кожного критерія та підвищити достовірність і об'єктивність прийняття рішень.

Практичне значення одержаних результатів:

1. Усі теоретичні розробки дисертації представлені у вигляді комплексного підходу щодо проєктування віртуальної мультихмарної платформи, який використаний для ефективного вибору набору хмарних сервісів та автоматизованого розгортання необхідних сервісів, які пропонуються хмарними провайдерами, що підвищує адаптивність та ефективність виконання обчислювальних процесів.

2. На основі удосконалених підходів до процесу формування мультихмарної

платформи за допомогою сучасних інструментів програмної інженерії, таких як LangChain і LLM, та використання мов програмування, зокрема Python, розроблено ПЗ, яке дозволило автоматизувати процеси проєктування і переналаштування віртуальної мультихмарної платформи із динамічно змінюваним набором вимог.

3. На основі запропонованої моделі і методу розроблено програмні рішення, які інтегровано в системи управління обчислювальними ресурсами віртуального мультихмарного середовища із динамічно змінюваним набором вимог для покращення гнучкості, підвищення рівня автоматизації та швидкості розгортання інфраструктури, зменшуючи залежність від ручних процесів та людського фактору.

4. Розроблено прототип системи управління обчислювальними ресурсами віртуального мультихмарного середовища із динамічно змінюваним набором вимог кінцевого користувача, який демонструє ефективність застосування запропонованих методів і моделей під час проєктування і розгортання обчислювальної інфраструктури, що підтверджує можливість практичного впровадження автоматизованих рішень у реальних сценаріях використання багатохмарних інфраструктур.

5. Виконано практичне впровадження розробленої системи управління обчислювальними ресурсами віртуального мультихмарного середовища із динамічно змінюваним набором вимог на інфраструктурі у виробничому середовищі великої транснаціональної корпорації, проведено тестування в корпоративному середовищі. Експертна оцінка, проведена спільно з інженерами провідної хмарної платформи, підтвердила ефективність запропонованого рішення.

6. Розроблені підходи впроваджено у навчальні та дослідницькі програми кафедри ІТТ НТУУ «КПІ» імені Ігоря Сікорського для підготовки фахівців у сфері хмарних обчислень, автоматизації процесів проєктування розподілених систем у мультихмарному середовищі, що сприяє покращенню практичних навичок роботи з багатохмарними середовищами.

Особистий внесок здобувача. Дисертаційна робота узагальнює результати теоретичних та експериментальних досліджень, проведених автором самостійно. Автором було розроблено формалізовану модель проєктування обчислювальної інфраструктури ресурсами віртуального мультихмарного середовища із динамічно змінюваним набором вимог, яка враховує багатокритеріальні аспекти вибору платформи для виконання обчислень [7-10]. Запропоновано концептуальну модель прийняття рішень на основі обчислень із застосуванням методів нечіткої логіки, що забезпечує автоматизоване проєктування та переналаштування віртуальної мультихмарної платформи із динамічно змінюваними вимогами, а також автоматичне розгортання обчислювальних завдань в ній. Автор впровадив комплексний метод формування обчислювальної інфраструктури у гетерогенному мультихмарному середовищі, який дозволяє автоматизувати розгортання обчислень інтегрувавши сучасні інструменти програмної інженерії, зокрема Python, LangChain і LLM [11]. Для оцінки ефективності розроблених методів і моделей було проведено імітаційне моделювання роботи запропонованої системи управління обчислювальними ресурсами віртуального мультихмарного середовища, що підтвердило їх дієвість у гетерогенних інфраструктурах. Експериментальна перевірка прототипу системи здійснювалась на інфраструктурі у виробничому середовищі великої транснаціональної компанії за участю інженерів провідних консалтингових компаній, що дозволило підтвердити ефективність розробленого комплексного методу в реальних умовах використання. Автор також виконав аналіз та узагальнення результатів тестування, що допомогло визначити ключові переваги та обмеження запропонованого комплексного методу формування обчислювальної інфраструктури у гетерогенному мультихмарному середовищі.

Апробація результатів дисертації. Розроблений прототип системи управління обчислювальними ресурсами віртуального мультихмарного середовища було апробовано шляхом практичного розгортання на інфраструктурі

у виробничому середовищі великої транснаціональної корпорації, що дозволило перевірити їх ефективність у реальних умовах експлуатації. Оцінка результатів була підтримана експертами галузі, зокрема інженерами провідних консалтингових компаній, що забезпечило отримання зворотного зв'язку та підтвердило практичну значущість розробленого комплексного методу. Результати роботи та ключові положення були представлені й одержали схвалення на:

- міжнародній конференції PyCon Poland 2023 (м. Краків, Польща, 2023 р.);
- міжнародній конференції Voxxed Days Cluj 2023 (м. Клуж-Напока, Румунія, 2023 р.);
- міжнародній конференції PyCon CZ 2023 (м. Прага, Чехія, 2023 р.);
- міжнародній конференції Ruscon IT 2023 (м. Флоренція, Італія, 2023 р.);
- міжнародній конференції EuroPython 2024 (м. Прага, Чехія, 2024 р.);
- міжнародній конференції RusconDE 2024 (м. Берлін, Німеччина, 2024 р.);
- технічній конференції Ruscon Wroclaw 2024 (м. Вроцлав, Польща, 2024 р.).

Публікації. За темою дослідження опубліковано статті у фахових журналах і збірниках матеріалів конференцій: 4 статті у наукових фахових виданнях України, 1 тези у матеріалах міжнародної конференції TCSET (Scopus), та 8 виступів на міжнародних конференціях розробників.

Структура та обсяг дисертації. Дисертаційна робота складається зі вступу, п'ятих розділів, висновків, списку використаних джерел із 93 найменувань, 21 додаток. Загальний обсяг роботи 227 сторінок, з яких 170 сторінок основного тексту, 10 сторінок використаних джерел та 47 сторінок додатків. Робота містить 26 рисунків, 6 таблиць.

РОЗДІЛ 1

АНАЛІЗ ПРОБЛЕМ ТА АКТУАЛЬНІ ПІДХОДИ ДО ПРОЄКТУВАННЯ ОБЧИСЛЮВАЛЬНОЇ ІНФРАСТРУКТУРИ У ГЕТЕРОГЕННИХ МУЛЬТИХМАРНИХ СЕРЕДОВИЩАХ

1.1 Особливості обчислювального навантаження та вимоги щодо проєктування мультихмарного середовища

З початку виникнення терміну «хмарних обчислень» і до теперішнього часу використання хмарних сервісів і ресурсів зазнало великої трансформації. Пандемія COVID-19 прискорила перехід бізнесів в онлайн-середовище, що суттєво вплинуло на підходи до проєктування обчислювальних інфраструктур, які мають забезпечувати ефективний розподіл навантаження між різними хмарними сервісами.

Розвиток інформаційно-комунікаційних мереж останніх поколінь супроводжується зростанням обсягів обробки даних, необхідністю забезпечення низьких затримок, високої надійності та масштабованості. У цьому контексті хмарні обчислення, а особливо мультихмарні архітектури, відіграють вирішальну роль в еволюції ІКМ. Мультихмарна інфраструктура забезпечує можливість одночасного використання ресурсів від кількох хмарних провайдерів, що дозволяє адаптивно розподіляти обчислювальні навантаження, балансувати трафік, оптимізувати витрати та забезпечувати відмовостійкість телекомунікаційних сервісів. Для операторів ІКМ це створює нові можливості: з одного боку – зменшення експлуатаційних витрат, з іншого – реалізація інтелектуального управління мережею через аналіз великих даних (Big Data) та автоматизоване масштабування обчислювальних ресурсів. Особливе значення мультихмарна модель набуває у контексті впровадження мереж 5G, де периферійні обчислення (Edge computing) разом із хмарними дата-центрами забезпечують критично

важливу інфраструктуру для надання сервісів з мінімальними затримками. Інтеграція мультимарних обчислень з мережецентричними підходами (такими як SDN і NFV) дозволяє реалізовувати логічний поділ мережі (network slicing), що критично важливо для телекомунікаційного середовища з високою динамікою вимог. Таким чином, мультимарні обчислення не є окремим напрямом ІТ, а безпосередньо інтегруються у структуру сучасних телекомунікаційних систем. Вони перетворюються на необхідний компонент, що забезпечує реалізацію інтелектуальних функцій управління, обробки даних та масштабування в динамічному середовищі. Саме тому дослідження ефективного проєктування обчислювальної інфраструктури у мультимарному середовищі має прямий зв'язок із розвитком телекомунікаційних технологій.

Сучасний стан галузі хмарних технологій характеризується наявністю різних архітектурних рішень для організації інфраструктури. У минулому користувачі переважно орієнтувалися на послуги одного постачальника, однак внаслідок посиленої конкуренції та спеціалізації сервісів, спостерігається тенденція до використання послуг різних хмарних провайдерів [12]. Це явище можна проілюструвати на прикладі спеціалізованих хмарних сервісів для розпізнавання медичної термінології. Розробники медичних програм надають перевагу таким спеціалізованим сервісам, водночас зберігаючи дані у постачальників, що пропонують найбільш економічно вигідні умови. Це формує типову модель, коли масиви даних зберігаються в одному хмарному середовищі, тоді як їх обробка здійснюється в іншому. Результати обробки можуть розміщуватися в третьому середовищі, яке пропонує оптимальні умови для аналітичної роботи. Відповідно, розробники прагнуть оптимізувати хмарну інфраструктуру через використання гетерогенних ресурсів від декількох постачальників [13].

Підприємства по всьому світу все частіше використовують у своїй діяльності кілька хмарних провайдерів. Звіт Flexera за 2024 рік показує 89% таких користувачів [14] (Рисунок 1.1).

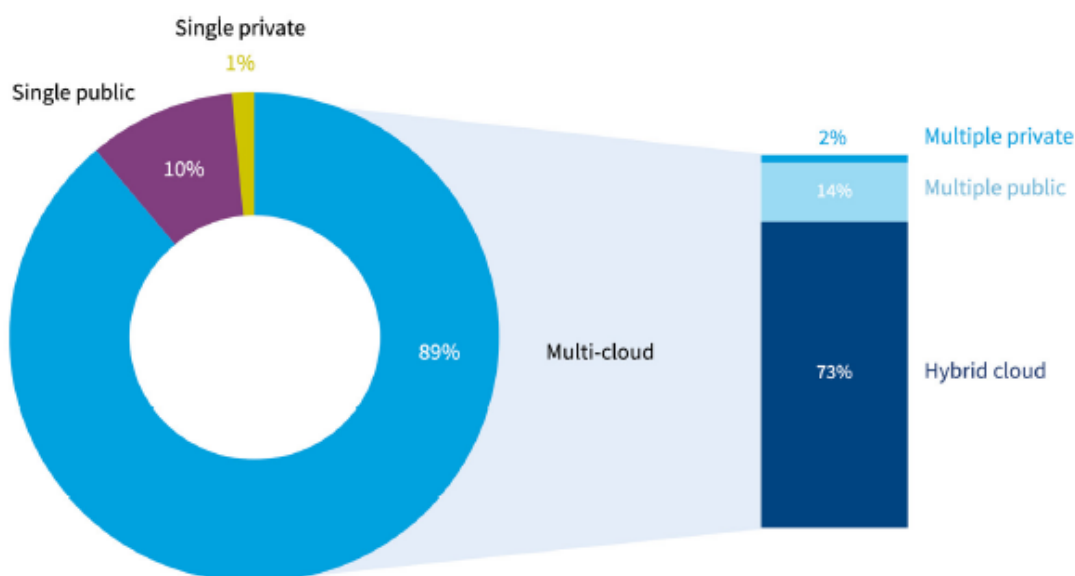


Рисунок 1.1 [13] Хмарна стратегія для всіх організацій, звіт Flexera

Існують наступні моделі розгортання хмари [15]:

- Приватна хмара – зазвичай запускається і використовується однією організацією, розташованою, частіше за все, локально. Тобто інфраструктура, обладнання та програмне забезпечення приватної хмари обслуговуються та використовуються конкретним бізнесом.

- Публічна хмара – загальнодоступна, надає ресурси для обчислення, зберігання, середовища для розробки та розгортання. Належать і керуються сторонніми постачальниками.

- Гібридна хмара – поєднує використання приватних і публічних хмар. У випадку використання як мінімум двох публічних хмар, говорять про мультихмару (Рисунок 1.2).

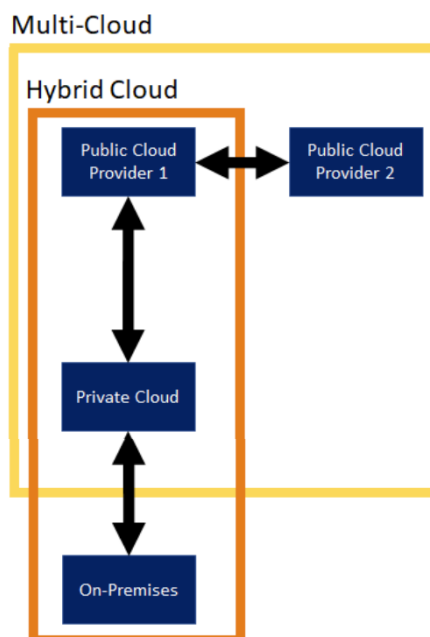


Рисунок 1.2 [16] Загальне представлення мультихмари і гібридної хмари

Гібридна хмарна архітектура включає локальну інфраструктуру (on-premises), тоді як багатохмарна архітектура її не включає [16].

Мультихмарність характеризується послідовним або одночасним використанням сервісів від різних провайдерів для виконання програми [17] (Рисунок 1.3). В межах даної роботи будемо вважати систему мультихмарною, якщо вона використовує два або більше хмарних сервісів від будь-якої кількості різних хмарних провайдерів в одній мережевій архітектурі. Це стосується розподілу хмарних активів, програмного забезпечення, додатків тощо між кількома хмарними середовищами.

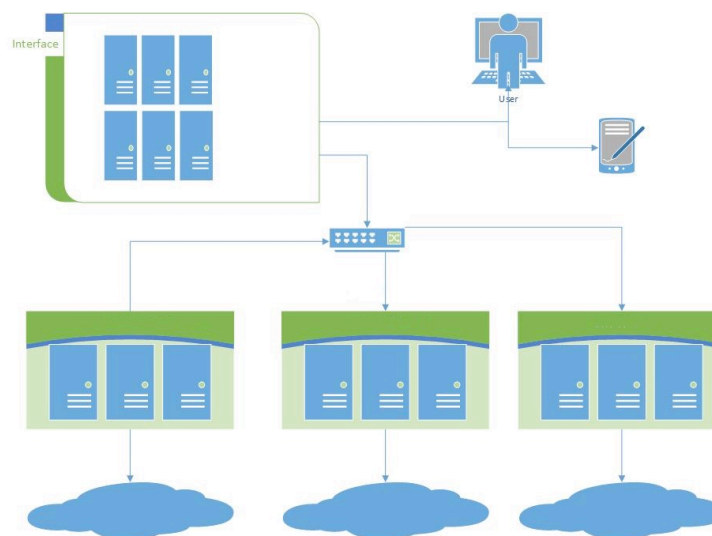


Рисунок 1.3 [17] Абстрактний вигляд мультихмари

Для керування додатками у різних хмарах існують наступні архітектури [16]:

1. Розподіл компонентів програми в різних хмарних середовищах. Наведена на рис 1.4 архітектура розбиває компоненти для інтерфейсу та серверної частини та розгортає в різних хмарах.

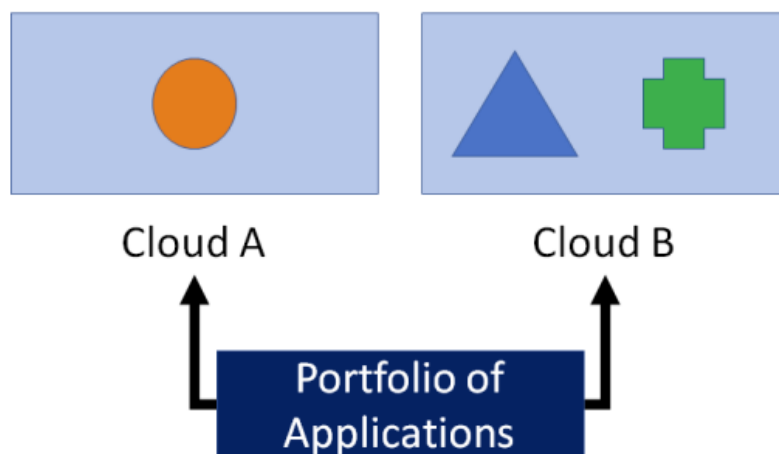


Рисунок 1.4 [16] Розподіл компонентів програми в різних хмарних середовищах

2. Збереження всіх компонентів в обох хмарах. Така архітектура підійде, коли потрібно створити стійкість і аварійне відновлення. Можемо налаштувати систему

як активну-активну, коли обидві хмари приймають трафік у будь-який час, або активну-пасивну, де одна хмара працює як основна та приймає живий трафік. Вторинна хмара оброблятиме трафік, лише якщо виникне проблема з основною хмарою, і вона перестане відповідати (рис 1.5).

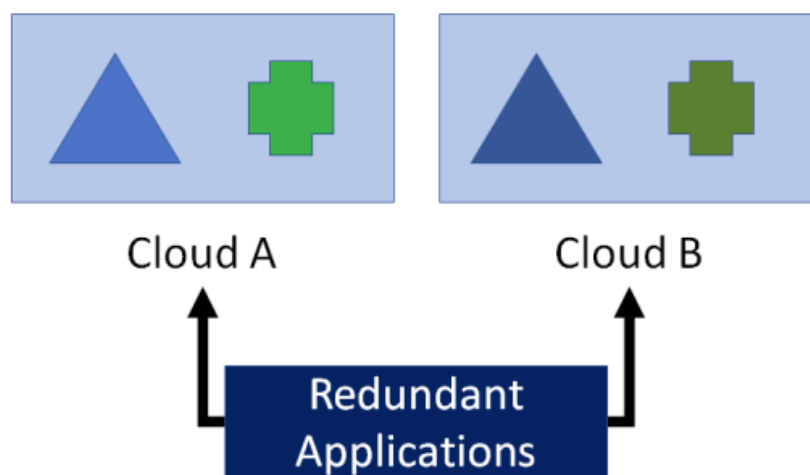


Рисунок 1.5 [16] Збереження всіх компонентів в обох хмарах

Сучасні багатохмарні архітектури розроблені для одночасного використання кількох постачальників хмарних послуг, пропонуючи організаціям підвищену гнучкість, резервування та свободу від прив'язки до постачальника. Ці архітектури дозволяють розподіляти робочі навантаження в різних середовищах, допомагаючи збалансувати витрати, продуктивність, відповідність та інші операційні потреби.

Незважаючи на численні переваги мультихмарних архітектур, існують проблемні аспекти, які потребують подальшого удосконалення [18-20]:

- Відмінності в API, форматах метрик і політиках безпеки між провідними постачальниками хмарних послуг ускладнюють процес їх уніфікованої взаємодії.
- Відсутність глобального моніторингу у реальному часі.
- Недостатнє врахування багатокритеріальних факторів.
- Сучасні алгоритми часто не забезпечують достатньої адаптивності до змін трафіку, нових регуляторних вимог або коливань вартості обчислювальних

ресурсів.

- Передача стану, даних або контейнерів між хмарами є складною технічно та фінансово, особливо в умовах безперервної роботи сервісів.

1.2 Організація мультихмарного середовища та підтримка його функціонування

Організацію мультихмарного середовища можна представити у вигляді абстрактних рівнів (рис 1.6).

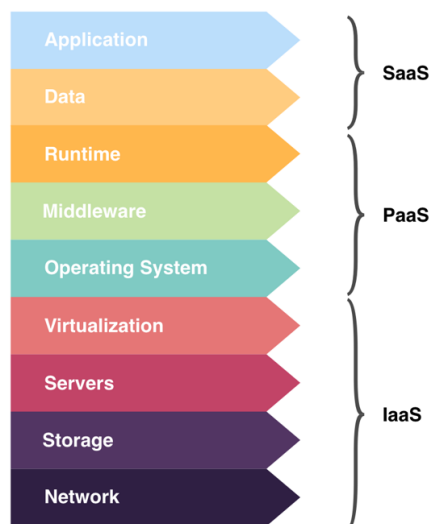


Рисунок 1.6 [10] Абстрактні рівні хмарного стеку

У відповідності до [29] можна виділити дев'ять рівнів абстракції, а саме мережа (нижня частина стека), сховище, сервери, віртуалізація, операційна система, проміжне програмне забезпечення, час виконання, дані та додаток (верхня частина стека).

Підходи до хмарної сумісності можна розглядати як багаторівневі моделі, де кожен рівень має взаємодіяти з наступним рівнем і зі своїм аналогом в іншому провайдері.

Хмарні обчислення доступні в трьох моделях обслуговування рис.1.7 [30]: «Інфраструктура як послуга» (IaaS), «Платформа як послуга» (PaaS) і «Програмне

забезпечення як послуга» (SaaS). У випадках SaaS або PaaS базові ресурси також керуються хмарним провайдером.

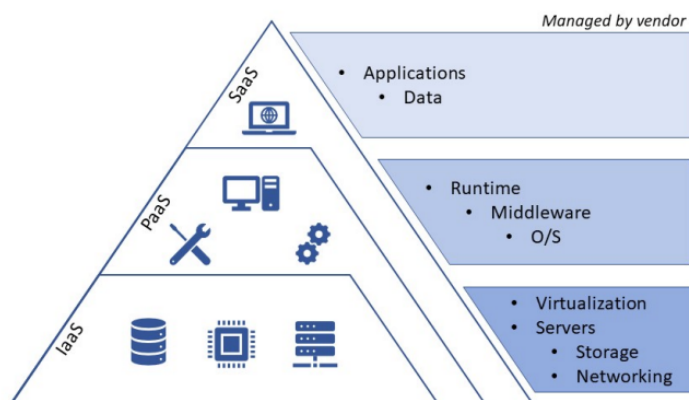


Рисунок 1.7 [30] Послуги, які керуються провайдером хмарної платформи на різних рівнях абстракції

Хмарна сумісність на рівнях PaaS і SaaS залежить від рівня IaaS, що вказує на те, що сумісність на рівні IaaS має ключове значення.

Різні джерела інтерпретують термін «хмарна гетерогенність» по-різному. На нижчому рівні гетерогенність означає, що різні типи процесорів об'єднуються для забезпечення віртуалізованого гетерогенного обладнання в межах одного хмарного провайдера. На вищому рівні гетерогенність має на увазі в контексті мультихмарних технологій поєднання послуг від декількох постачальників. Таким чином, гетерогенність досягається за рахунок використання гіпервізорів та програмних пакетів від різних постачальників [10].

Інтерфейс програми дозволяє приховати системні складності мультихмарності. Розрізнена архітектура складається з різних апаратних засобів, віртуальних машин, операційних систем, приватних хмар, кластерів тощо. Доступ користувачів до багатохмарних служб реалізується через єдиний уніфікований інтерфейс, що надає змогу користувачам зосередитися на робочому процесі.

Для підтримки функціонування мультихмарного середовища, можливо виділити дві групи:

1. Інструментарій підтримки функціонування мультихмарного середовища. На кожному з рівнів мультихмарного середовища інструментарій підтримки його функціонування має бути сумісним. Цей інструментарій не є залежним від хмарних провайдерів. Власник мультихмарного середовища може обирати найбільш ефективний інструментарій для розробки власного гетерогенного програмного середовища.

2. Обчислювальні сервіси, які використовують на рівнях PaaS та SaaS для виконання задач кінцевого користувача.

1.2.1. Інструментарій розгортання мульти-хмарних середовищ

Для розгортання мультихмари існує багато інструментів [21-23]:

1. Уніфіковані API для керування хмарними ресурсами. Інструменти, які абстрагують специфіку кожного хмарного постачальника, дозволяючи програмам працювати в будь-якій хмарі без значних модифікацій (HashiCorp Terraform, Apache Libcloud, Pulumi).

2. Інструменти керування хмарию – забезпечують централізований контроль, моніторинг і автоматизацію для керування ресурсами, витратами та політикою в кількох хмарних середовищах (Apache OpenStack, RightScale (Flexera), CloudHealth by VMware).

3. Контейнеризація в хмарах – спрощення розгортання та керування контейнерними програмами на кількох хмарних платформах (Kubernetes, Red Hat OpenShift, Docker Swarm).

4. Менеджери баз даних і сховищ – надають інструменти для керування базами даних і сховищем у хмарних провайдерах, забезпечуючи доступність, послідовність і сумісність даних у різних середовищах. Це можуть бути мультихмарні шлюзи зберігання і платформи керування даними. MiniIO, Ceph тощо допомагають

керувати ресурсами зберігання на різних хмарних платформах. Крім того, такі бази даних, як NuoDB, FaunaDB і CockroachDB, задовольняють вимоги багатохмарних середовищ, забезпечуючи безпроблемне налаштування баз даних і керування ними.

5. Спеціальні інструменти для хмари – призначені для інтеграції певних служб від одного хмарного постачальника з іншими хмарними середовищами (AWS Outposts, Google Anthos, Azure Arc).

Аналіз поширених інструментів мультихмарної взаємодії дозволяє виокремити наступні програмні засоби.

1. Kubernetes [24] – це платформа керування контейнерами з відкритим вихідним кодом, яка автоматизує розгортання, масштабування та керування контейнерними програмами. Kubernetes складається з рівня керування (керує кластером) і робочих вузлів (Рисунок 1.8). Автоматично виконує такі завдання, як планування контейнерів, масштабування, балансування навантаження та самовідновлення.

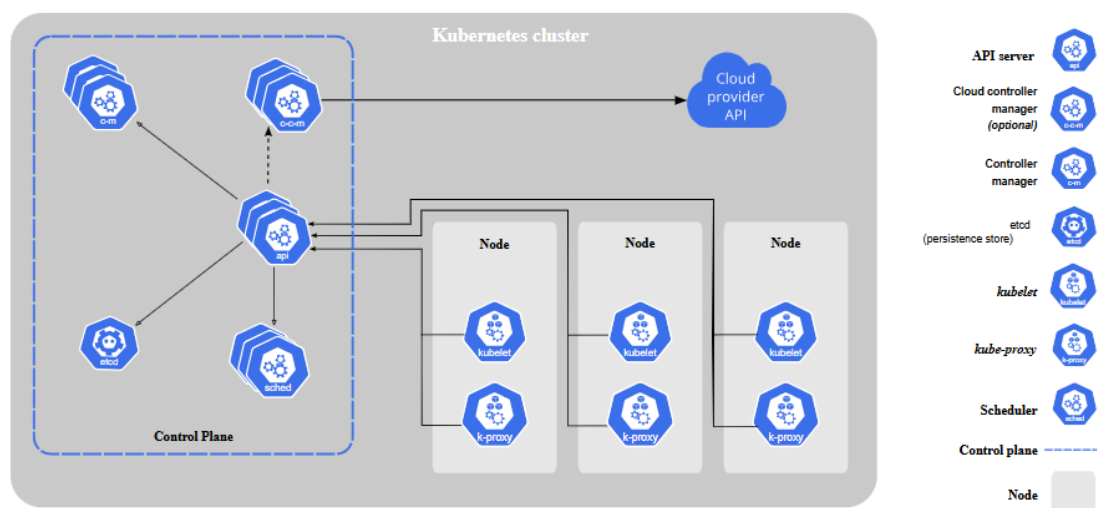


Рисунок 1.8 [21] Компоненти кластеру Kubernetes

Kubernetes дозволяє гнучко розгорнути ці компоненти та керувати ними. Архітектуру можна адаптувати до різних потреб, від невеликих середовищ розробки до розгортання великого виробництва.

2. Terraform [25] – це інструмент «Інфраструктура як код» (IaC), що дозволяє визначати та надавати інфраструктуру на мові декларативної конфігурації. Terraform підтримує широкий спектр хмарних провайдерів і локальні рішення, допомагає застосувати єдиний робочий процес автоматизації, щоб керувати кількома постачальниками інфраструктури та SaaS, а також керувати міжхмарними залежностями. Це спрощує керування життєвим циклом і оркестровку багатохмарної інфраструктури будь-якого масштабу.

3. S3Proxy [26] – приклад мультихмарних шлюзів зберігання (Multi-cloud storage gateways). Суть їх роботи полягає в перетворенні API-інтерфейсів різних хмарних провайдерів в єдиний API, полегшуючи переміщення даних і управління ними. Вони дозволяють користувачам отримувати доступ і керувати даними в різних хмарних провайдерах без необхідності впровадження та підтримки кількох інтеграцій API. Спрощена архітектура доступу до даних через шлюз зберігання проілюстрована на рис. 1.9.

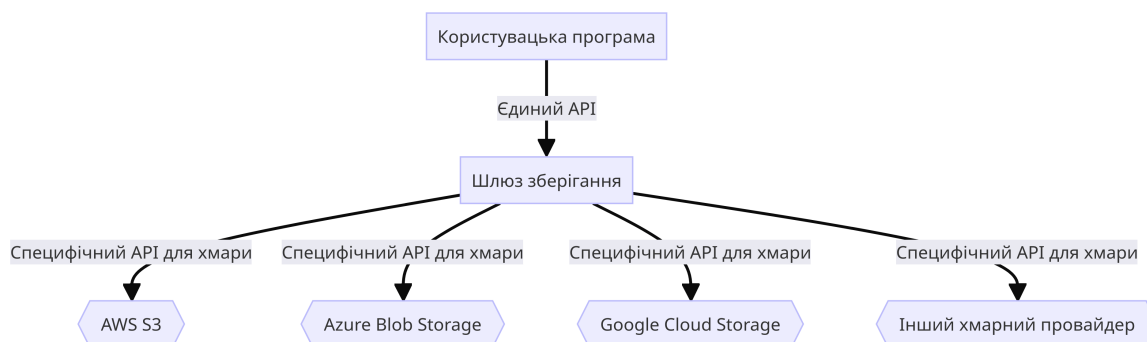


Рисунок 1.9 [8] Спрощена схема мультихмарного шлюзу

4. Apache Nifi [27] – представляє собою платформу керування даними (Data management platforms або DMP). DMP – це програмні рішення, які, на відміну від шлюзів зберігання, призначені для полегшення не лише керуванням даними та

доступу до них у мультихмарних середовищах, але й міграції, захисту та керування даними між різними постачальниками хмарних сховищ. Архітектура та потік доступу до даних дуже схожі на шлюзи зберігання (рис. 1.9).

5. Apache Libcloud [28] – хмарно-незалежна бібліотека (Cloud-agnostic library). Хмарно-незалежні бібліотеки – це програмні інструменти, які забезпечують єдиний уніфікований інтерфейс для взаємодії з декількома службами хмарного сховища на рівні програмування додатків, усуваючи необхідність працювати з кількома спеціальними API для cloud-середовища. Використання таких бібліотек додає значних зусиль для реалізації кінцевого додатку, бо існує потреба в реалізації власного спеціального API, який зазвичай більш обмежений і менш документований, ніж API хмарних провайдерів. Проілюструємо процес інтеграції хмарно-незалежної бібліотеки за допомогою діаграм 1.10 і 1.11.

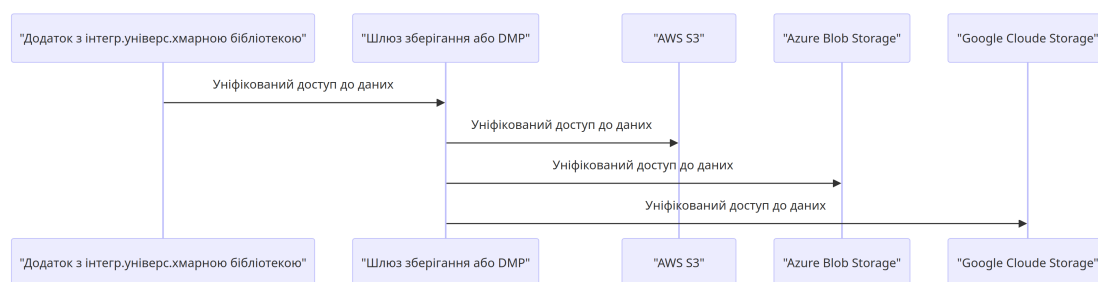


Рисунок 1.10 [8] Діаграма послідовності дій щодо інтеграції хмарно-незалежної бібліотеки в кінцевий додаток

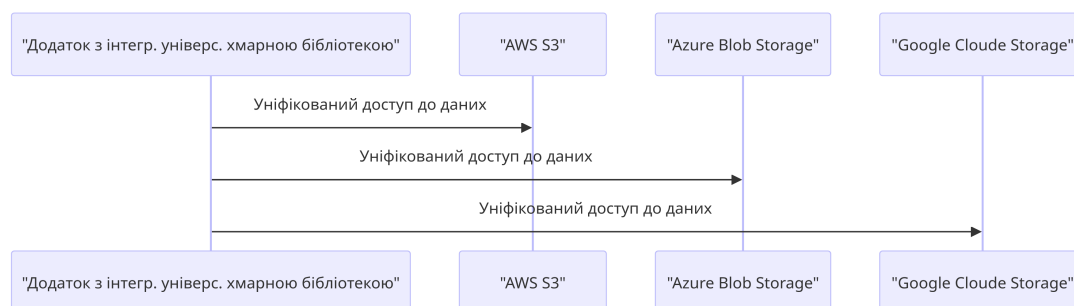


Рисунок 1.11 [8] Діаграма послідовності результатів доступу до даних із інтегрованою хмарно-незалежною бібліотекою

1.2.1 Особливості мультихмарної взаємодії

Зростання впровадження мультихмарних технологій разом із зростаючим попитом на ефективну сумісність даних створює складну проблему для організацій. Різноманітність хмарних платформ, кожна з яких має специфічні для постачальника API, служби та механізми безпеки, збільшує складність мультихмарних архітектур [31]. Складність мультихмарної архітектури зумовлює потребу у впровадженні структурованих і стратегічно обґрунтованих рішень. Рішення для сумісності повинні не тільки відповідати технічним вимогам інтеграції, але й ширшим організаційним цілям щодо масштабованості, продуктивності та вартості [32]. Використання багатохмарного підходу потребує методів, які дозволяють організаціям застосовувати найбільш ефективні стратегії.

Багатохмарна взаємодія стосується того, як додатки та служби обмінюються даними між кількома хмарними провайдерами. Можна виділити такі функціональні вимоги до організації мультихмарної взаємодії [33-35]:

- Синхронізація даних між хмарами забезпечує узгодженість у середовищі. Це можуть бути технології шлюзів хмарних сховищ, хмарно-незалежні бази даних, служби реплікації даних.
- Безпечне та надійне мережеве з'єднання між хмарами має важливе значення. Необхідно забезпечити з'єднання з невеликою затримкою і високою пропускну здатністю.
- Централізоване керування забезпечує уніфіковане уявлення про ресурси різних хмарних провайдерів.
- Наявність інструментарію мультихмарної оркестровки, що дозволяє програмам охоплювати кілька хмар, забезпечуючи стійкість і гнучкість.
- Використання політики безпеки та систем керування ідентифікацією, які стандартизовані в хмарних середовищах для забезпечення узгодженого контролю доступу та захисту даних.

- Застосування багатохмарних інструментів управління витратами для надання інформації про обсяги використання хмарних ресурсів та відповідні витрати, допомагаючи організаціям визначати можливості економії.

Прикладом розповсюдженого обчислювального сервісу є Безсерверна Функція (serverless function) – програмна функція, яку може написати розробник програмного забезпечення прикладної системи для виконання однієї функціональної задачі, яка розміщується та підтримується в інфраструктурі провайдера хмарних обчислень. Безсерверні функції являють собою легку обчислювальну модель, керовану подіями. Вони спрощують багатохмарну взаємодію завдяки абстрагуванню від проблем з інфраструктурою, забезпеченню керованих подіями робочих процесів у хмарах, а також зниженню витрат на обслуговування робочого навантаження [36].

У контексті мультихмар serverless-функції працюють наступним чином:

- запускаються такими подіями, як виклики API, завантаження файлів або зміни бази даних;
- виконуються лише за потреби, що мінімізує витрати;
- кожен постачальник хмарних послуг пропонує багату екосистему, до якої можуть підключитися безсерверні функції.

Безсерверні функції надаються найбільш відомими постачальниками хмарних послуг, серед яких можна назвати Lambda від Amazon Web Services, Cloud Functions від Google Cloud Platform, та Azure Functions від Microsoft Azure.

Наведемо приклади практичного застосування AWS Lambda в мультихмарних сценаріях:

1. Багатохмарне аварійне відновлення. AWS Lambda запускає основну безсерверну функцію для обробки запитів і зберігання даних в AWS DynamoDB.
2. Мультихмарний конвеєр обробки зображень. AWS Lambda активує завантаження файлів у S3, а функція – конвертує зображення у стандартний формат.

3. Багатохмарне машинне навчання. Компанія IoT запускає моделі машинного навчання для прогнозів у реальному часі за допомогою безсерверних функцій у різних хмарах. AWS Lambda обробляє вхідні дані IoT з пристроїв і виконує базову попередню обробку даних датчиків.

Проведений аналіз структури та вимог до проєктування мультихмарного середовища для кінцевого користувача свідчить про:

- Необхідність вибору найбільш ефективного інструментарію підтримки функціонування мультихмарного середовища в залежності від умов, за яких це середовище утворюється.
- Під час функціонування прикладної системи у мульти-хмарному середовищі виникає ситуація, коли умови його функціонування змінюються (зміна об'єму даних, які обробляються; зміна бюджету, вартості хмарних послуг, тощо).
- В зв'язку зі зміною умов функціонування мультихмарного середовища виникає потреба в перепроєктуванні його архітектури, цей процес має виконуватись автоматизовано без ручного перепрограмування з метою зменшення витрат на підтримку програмного комплексу.

1.3 Переваги та обмеження рішень щодо мультихмарної взаємодії

Компанії-постачальники хмарних послуг розробляють різні підходи до інтеграції мультихмарних середовищ. Але, перш за все, компанії-користувачі мають визначитися з цілями і потім вже обирати підходи, які найкраще реалізують їхні вимоги. Для цього варто звернути увагу на наступне [37]:

- аналіз власної системи для визначення необхідних сервісів;
- взаємодія різних хмар: потрібно переконатися, що обрані платформи сумісні і можуть інтегруватися;
- проєктування архітектури майбутньої мультихмарної системи;
- впровадження надійних заходів керування і безпеки;

- вибір рішень для обслуговування і моніторингу;

До основних типів мультихмарних стратегій можна віднести такі підходи [38]:

1. Розподілене хмарне розгортання передбачає розміщення програмних компонентів на кількох платформах з метою підвищення доступності та зменшення затримок.

2. Реплікація та резервне копіювання даних – тиражування даних між кількома хмарними провайдерами, що забезпечує високу доступність і аварійне відновлення.

3. Пом'якшення блокування постачальника. Цього можна досягти шляхом прийняття багатохмарної архітектури, яка використовує сильні сторони кожного хмарного постачальника. Маючи більше можливостей, клієнти хмарних технологій можуть домовитися про кращі умови та ціни та уникнути прив'язки до власних технологій постачальника, які можуть конфліктувати з власною мультихмарною стратегією клієнта.

4. Оптимізація продуктивності – оптимізація часу безвідмовної роботи та продуктивності програм. Багатохмарність дозволяє розгортати програми в областях, які фізично ближче до кінцевих користувачів.

5. Оптимізація витрат – використовуючи моделі ціноутворення кожного постачальника публічної хмари, компанія може отримати найкращу вартість.

6. Відповідність і суверенітет даних – з кількома хмарними постачальниками компаніям легше дотримуватись законів і норм щодо суверенітету даних. Цього можна досягти шляхом розгортання додатків у регіонах, які відповідають місцевим законам і нормам відповідності.

Розглянемо існуючі підходи мультихмарної взаємодії.

Мультихмарні шлюзи зберігання – це апаратне або програмне забезпечення, яке надає плавний доступ до даних, що зберігаються в кількох хмарних провайдерах, діючи як посередник між програмами користувача та службами хмарного зберігання [8, 39].

Доступ до хмарного сховища через шлюз зазвичай має такі переваги:

- Спрощене керування даними за допомогою єдиного уніфікованого API.
- Уникає блокування постачальника та забезпечує стійкість даних шляхом розподілу даних між кількома хмарними провайдерами.
- Потенційно покращена продуктивність доступу до даних за допомогою кешування та методів оптимізації.

При цьому до недоліків можна віднести:

- Додаткові витрати за користування шлюзом.
- Зниження продуктивності або збільшення затримки через додатковий рівень абстракції.
- Залежність від постачальника шлюзу щодо оновлень, виправлення помилок і підтримки.

Розробками у цій сфері є: Cloudian Hyper-Store, Nasuni Cloud File Services, Panzura Freedom, Morro Data CloudNAS, S3Proxy, які оптимізують використання сховища та підвищують безпеку, одночасно забезпечуючи мультихмарний доступ до даних.

Платформи керування даними – це платформи, які абстрагують базові служби хмарного зберігання та забезпечують централізоване управління даними, що дозволяє організаціям запроваджувати політики, автоматизувати процеси та отримувати аналіз своїх даних у кількох хмарах [8, 40].

Переваги платформ управління даними:

- Оптимізують керування даними, обробку та аналітику в багатьох хмарних провайдерах, пропонуючи більш розширені функції, ніж шлюзи зберігання.
- Покращують управління даними та відповідність вимогам, забезпечуючи централізований контроль і видимість даних із різних джерел, у тому числі мультихмарних служб зберігання.
- Підтримують розширену обробку даних і можливості аналітики, які

зазвичай недоступні в шлюзах зберігання.

Недоліки платформ управління даними:

- Можуть бути складнішими у впровадженні та обслуговуванні завдяки додатковим функціям і функціям, які надає платформа.
- Можуть призвести до підвищення вартості використання платформи.
- Деякі платформи можуть не підтримувати всіх хмарних постачальників або певні служби зберігання, що може призвести до потенційних проблем із сумісністю або обмежень.

Серед прикладів DMPs можна назвати Talend Data Fabric, Informatica Intelligent Cloud Services, Azure Synapse Analytics, Google Cloud BigQuery, Dell Boomi AtomSphere, SnapLogic Intelligent Integration Platform, а також рішення з відкритим кодом Apache Nifi.

Інструменти стандартизації API – це група рішень, які дозволяють приховати відмінності API різних хмарних постачальників, тобто використовувати той самий робочий процес для керування декількома провайдерами, полегшуючи керування хмарами [41]. До вказаних інструментів можна віднести і хмарно-незалежні бібліотеки. Ці бібліотеки часто розробляються та підтримуються спільнотою учасників, що робить їх ініціативами з відкритим вихідним кодом і дозволяє будь-яким розробникам додавати відсутні функції та налаштовувати інструмент відповідно до потреб власної програми. Однак, оскільки ці бібліотеки потрібно впроваджувати в кінцеву програму на набагато нижчому рівні, вони можуть потребувати більше зусиль у розробці порівняно з попередніми двома рішеннями. Деякі приклади: JClouds, Pulumi, Apache Libcloud, Terraform, OpenStack та інші.

До переваг інструментів стандартизації API можна віднести:

- Спрощення роботи з різними хмарними провайдерами, надаючи єдиний інтерфейс API.
- Зменшення залежності від єдиного хмарного постачальника завдяки

забезпеченню портативності.

- Покращення управління ресурсами.
- Забезпечення централізованого контролю і моніторингу у багатохмарних середовищах.

Недоліки:

- Обмежений набір функцій.
- Додаткові рівні абстракції можуть викликати затримку.
- Початкове налаштування та конфігурація можуть бути складними та потребувати досвіду.

- Потенційні проблеми безпеки даних із сторонніми інструментами API.

Розглянемо інструменти автоматизованого розгортання, до яких відносяться:

- Terraform широко використовується для автоматизації надання інфраструктури в багатьох середовищах. Наприклад, щоб розгорнути віртуальну приватну хмару (VPC), підмережу та примірник EC2 необхідно виконати наступні кроки:

- ініціалізація Terraform;
- планування розгортання;
- застосування конфігурації.

Як результат, новий VPC, підмережа та примірник EC2 створюються в AWS, повністю автоматизовані Terraform.

- ArgoCD – це сучасний інструмент для автоматизації розгортання застосунків у Kubernetes-кластерах. Він побудований на принципах GitOps, де всі конфігурації зберігаються в Git-репозиторії, і Kubernetes автоматично синхронізує свій стан з цими конфігураціями.

Щоб поєднати Terraform і ArgoCD необхідно виконати наступне:

- Використати Terraform, щоб підготувати кластер Kubernetes.
- Розгорнути ArgoCD у кластері за допомогою Terraform.
- Використати ArgoCD для розгортання програм на основі GitOps.

1.4 Сумісність обчислень у мультихмарному середовищі та її вплив на гетерогенні мультихмарні системи

Впровадження мультихмарних рішень супроводжується викликами сумісності, зумовленими відмінностями у використовуваних архітектурах, стандартах і протоколах різних хмарних провайдерів. Основними причинами труднощів забезпечення сумісності є розбіжності у форматах даних, робочих моделях, програмних інтерфейсах (API) та політиках безпеки.

Існують певні розробки в напрямку стандартизації мультихмарного простору. Інтерфейс керування хмарними даними (Cloud Data Management Interface (CDMI)) [42] – це стандарт, який визначає функціональний інтерфейс для доступу та керування даними, що зберігаються в хмарі. За його допомогою можна створювати, отримувати, змінювати та видаляти файли чи інші дані, а також дізнаватися, як облаштоване хмарне сховище і керувати ним.

Позитивним стороною даної технології можна назвати відсутність прив'язки до одного хмарного постачальника, що дає можливість переміщувати дані між хмарами. Проте в окремих випадках стандарт може негативно вплинути на продуктивність і вимагати додаткових зусиль інтеграції з існуючими системами управління даними.

Відкритий інтерфейс хмарних обчислень (Open Cloud Computing Interface (OSCI)) [43] – це набір правил, які описують, як керувати різними ресурсами в хмарі, такими як сховища, сервери чи мережі. Він допомагає поєднувати сервіси різних хмарних провайдерів і створювати програми, які легко переносити між ними.

Цей інтерфейс спрощує роботу з хмарними ресурсами, підлаштовується під різні потреби користувачів і технології. Однак через його гнучкість може бути складно вибрати і налаштувати потрібну конфігурацію.

Описані стандарти мають деякі спільні переваги, такі як спрощення інтеграції та управління хмарними ресурсами, дозволяє використовувати різні хмарні

платформи без прив'язки до одного провайдера. Основні недоліки визначаються як потенційні проблеми з продуктивністю і складність впровадження. Таким чином існує необхідність у дослідженні альтернативних технік міжхмарної взаємодії.

Згідно з [44, 45] можна виділити організаційні проблеми, пов'язані з багатохмарною інтеграцією:

- потрібно найняти або навчити персонал новим навичкам, специфічним для підтримки публічних хмар;
 - складність політики управління навколишнім середовищем;
 - труднощі з оптимізацією витрат;
 - ризики, пов'язані з проблемами безпеки, даних і конфіденційності
- неузгоджена інфраструктура API, баз даних, мереж і безпеки.

До технічних викликів можна віднести:

- Відсутні стандарти API: загальні API для роботи багатохмарних систем потребують різних видів ресурсів, що надаються різними хмарами. Це складна проблема, оскільки нові ресурси часто додаються до хмарного ринку, тоді як немає UDDI-подібних каталогів, які б повідомляли про всі ресурси, доступні для використання.
- Різні абстракції: усі загальні абстракції, включаючи архітектуру сховища та мережі, відрізнятимуться в різних хмарних провайдерів.
- Обчислення вартості: моделі виставлення рахунків користувачам суттєво відрізняються в різних хмарних провайдерів, що робить майже неможливим динамічну оцінку вартості того самого завдання, яке виконується в тій чи іншій хмарі.

Можна виділити наступні переваги мультихмарних стратегій [16, 45]:

- Аварійне відновлення. Впровадження резервного копіювання, використовуючи відразу декілька хмарних провайдерів.
- Перехід до іншої хмари у випадку збою.

- Оптимізація витрат. Використання найбільш вигідних сервісів постачальників хмарних послуг.

- Суверенітет даних. Багато країн на законодавчому рівні приймають закони, які регулюють розміщення даних. Організації мають розміщувати дані у географічних регіонах, які не суперечать законодавству.

Розглянемо складність інтеграції API на прикладі Crossplane [46, 47]. Crossplane – це інструмент із відкритим вихідним кодом, призначений для керування та уніфікації багатохмарних середовищ за допомогою екосистеми Kubernetes.

- API хмарних провайдерів часто відрізняються своїми параметрами, ієрархіями ресурсів і логікою роботи, до того ж API часто оновлюються, додаються нові функції або припиняючи використовуватися старі. Відповідно Crossplane має постійно оновлювати контролери постачальника відповідно до цих змін, що вимагає значного обслуговування.

- Коли виклик API не виконується через хибну конфігурацію або несумісність, налагодження може бути складним. Помилки часто поширюються через Kubernetes, що призводить до незрозумілих повідомлень і потребує детального знання як Crossplane, так і API постачальника.

- Crossplane інтегрується з API хмарних провайдерів за допомогою облікових записів служб, ролей IAM або ключів API. Неправильно налаштовані дозволи або прострочені облікові дані можуть порушити роботу, особливо в дуже динамічних середовищах.

Рішеннями цих проблем можуть бути наступні кроки:

- Прийняття стандартизованих API.
- Інструменти проміжного програмного забезпечення, як Apache Libcloud або Terraform, можуть служити посередниками для подолання розбіжностей API.
- Динамічне керування обліковими даними.

Crossplane демонструє високу ефективність, проте існують суттєві обмеження його використання, зумовлені різнотипністю інтерфейсів програмування різних постачальників.

ВИСНОВКИ

1. Огляд розподілу обчислювального навантаження в мультихмарному середовищі дозволив виявити низку проблем: різнорідність інтерфейсів та сервісів провайдерів, різні формати метрик, недостатнє врахування багатокритеріальних факторів, проблеми з динамічною адаптацією до змін навантаження

2. Виконано аналіз багатохмарних середовищ, архітектур і існуючих підходів мультихмарної взаємодії. А також вимог щодо обслуговування мультихмарних систем, проблематика їхньої сумісності і спроби стандартизації в цьому напрямку.

3. Проведено огляд існуючих інструментів автоматизації, виявлено їх сильні і слабкі сторони. Серед розглянутих підходів: DMP, мультихмарні шлюзи, інструменти стандартизації API та контейнеризації.

4. Для підтримки функціонування мультихмарного середовища, можливо виділити дві групи:

- Інструментарій підтримки функціонування мультихмарного середовища. На кожному з рівнів мультихмарного середовища інструментарій підтримки його функціонування має бути сумісним. Власник мультихмарного середовища може обирати найбільш ефективний інструментарій для розробки власного гетерогенного програмного середовища.

- Обчислювальні сервіси, які використовують на рівнях PaaS та SaaS для виконання задач кінцевого користувача.

5. Визначено необхідність створення автоматизованої системи, яка б дозволяла організаціям на основі визначених ними критеріїв обирати і розгортати найбільш ефективну для себе багатохмарну стратегію.

РОЗДІЛ 2 ФОРМАЛІЗАЦІЯ ПРОЦЕСУ ВЗАЄМОДІЇ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ У ГЕТЕРОГЕННОМУ МУЛЬТИХМАРНОМУ СЕРЕДОВИЩІ

2.1 Критерії оцінювання ефективності мультихмарної взаємодії

На основі аналізу досвіду хмарних інтеграцій, реалізованих провідними агенціями, зокрема Nordcloud [48], Crayon [49] та Tech-5 [50], а також досліджень структурних особливостей мультихмарних середовищ, було визначено ключові критерії, що комплексно характеризують процес мультихмарної взаємодії [51, 52]. Ці критерії є фундаментальними для побудови формальної моделі оптимізації розподілу обчислень:

- Економічна ефективність (*EE, Economic Efficiency*) – характеризує сукупні витрати на використання хмарних сервісів, включаючи прямі витрати на обчислювальні ресурси, сховища даних, мережеву взаємодію, а також непрямі витрати, пов'язані з управлінням та моніторингом.

- Продуктивність (*P, Performance*) – відображає ефективність виконання обчислювальних завдань, включаючи швидкість обробки даних, час відгуку системи та загальну пропускну здатність. Охоплює час затримки між компонентами розподіленої системи та загальний час виконання обчислювальних завдань.

- Надійність (*R, Reliability*) – визначає стійкість системи до відмов окремих компонентів та здатність підтримувати безперервне функціонування в умовах часткової деградації інфраструктури.

- Сумісність взаємодії (*I, Interoperability*) – характеризує здатність різномірних хмарних сервісів ефективно взаємодіяти між собою через стандартизовані інтерфейси та протоколи.

- Безпека (*S, Security*) – включає аспекти захисту даних, аутентифікації, авторизації та відповідності регуляторним вимогам у розподіленому мультихмарному середовищі.

2.2 Математична модель мультихмарного середовища

Нехай $P = \{p_1, p_2, \dots, p_m\}$ – множина з m хмарних провайдерів, кожен з яких характеризується набором доступних обчислювальних сервісів, цінових політик та функціональних можливостей. Для кожного провайдера $p_i \in P$ визначимо:

- $R_i = \{r_{i1}, r_{i2}, \dots, r_{in}\}$ – множина доступних обчислювальних сервісів, де r_{ij} представляє j -й тип сервісу у провайдера p_i .
- $C_i: R_i \rightarrow R^+$ – функція економічної ефективності, що відображає вартість використання відповідного сервісу.
- $L_i: R_i \rightarrow R^+$ – функція часової затримки, що відображає часову затримку під час використання сервісу.

Визначимо множину обчислювальних завдань $T = \{t_1, t_2, \dots, t_k\}$, які потребують розподілу між хмарними провайдерами. Кожне завдання $t_j \in T$ характеризується:

- $D_j = \{d_{j1}, d_{j2}, \dots, d_{jl}\}$ – множина вимог до сервісів, де d_{ji} визначає потребу в сервісі типу i .
- $Q_j = \{q_{j1}, q_{j2}, \dots, q_{jp}\}$ – множина параметрів якості обслуговування (QoS), включаючи максимально допустиму затримку, вимоги до доступності, безпеки тощо.
- $I_j \subset T$ – множина залежностей від інших завдань, тобто завдань, які повинні бути виконані перед t_j .

Для формалізації процесу розподілу завдань у мультихмарному середовищі введемо матрицю призначення $X = [x_{ij}]$, де:

$$x_{ij} = \begin{cases} 1, & \text{якщо завдання } t_i \text{ призначено провайдеру } p_j \\ 0, & \text{в іншому випадку} \end{cases}.$$

Така формалізація дозволяє описати ключовий аспект проєктування мультимарної інфраструктури – оптимальний розподіл обчислювальних завдань між доступними ресурсами різних провайдерів.

2.3 Цільові функції оптимізації та їхній зв'язок з критеріями ефективності

Визначені в підрозділі 2.1 критерії оцінювання ефективності мультимарної взаємодії безпосередньо трансформуються у відповідні цільові функції оптимізації для формалізації процесу вибору оптимальної архітектури взаємодії сервісів мультимарної системи.

- Мінімізація загальної вартості (*CM, Cost Minimization*) – відповідає критерію економічної ефективності (*EE*):

$$\min Cost(X) = \sum_{i=1}^k \sum_{j=1}^m x_{ij} \cdot C_j(D_i), \quad (2.1)$$

де $C_j(D_i)$ – функція, що обчислює вартість виконання завдання t_i з вимогами D_i у провайдера p_j .

- Мінімізація часу виконання (*ETM, Execution Time Minimization*) – відповідає критерію продуктивності (*P*):

$$\min \max(i, j: x_{ij} = 1) (T_j(t_i) + L_j(D_i)), \quad (2.2)$$

де $T_j(t_i)$ – час виконання завдання t_i у провайдера p_j , а $L_j(D_i)$ – часова затримка, пов'язана з використанням сервісів.

- Максимізація надійності (*RM, Reliability Maximization*) – відповідає критерію надійності (*R*):

$$\max \prod_{i=1}^k \sum_{j=1}^m x_{ij} \cdot A_j(t_i), \quad (2.3)$$

де $A_j(t_i)$ – надійність виконання завдання t_i у провайдера p_j .

- Максимізація сумісності взаємодії (*IM, Interoperability Maximization*) – відповідає критерію сумісності взаємодії (*I*):

$$\max \sum_{i=1}^k \sum_{j=1}^m x_{ij} \cdot I_j(D_i, T), \quad (2.4)$$

де $I_j(D_i, T)$ – метрика сумісності взаємодії для завдання t_i з вимогами D_i та множиною завдань T у системі провайдера p_j .

- Максимізація безпеки (*SM, Security Maximization*) – відповідає критерію безпеки (*S*):

$$\max \sum_{i=1}^k \sum_{j=1}^m x_{ij} \cdot S_j(D_i), \quad (2.5)$$

де $S_j(D_i)$ – рівень безпеки, який забезпечується провайдером p_j для завдання t_i з вимогами D_i .

- Комплексна цільова функція з ваговими коефіцієнтами (*WMOF, Weighted Multi-Objective Function*):

$$IE(X) = w_1 \cdot Cost_{norm}(X) + w_2 \cdot Perf_{norm}(X) + w_3 \cdot Rel_{norm}(X) + w_4 \cdot Inter_{norm}(X) + w_5 \cdot Sec_{norm}(X), \quad (4)$$

де w_1, w_2, w_3, w_4, w_5 – вагові коефіцієнти важливості критеріїв (вартості, продуктивності, надійності, сумісності, безпеки відповідно). Значення вагових коефіцієнтів визначаються пріоритетами конкретного застосування та можуть адаптуватися динамічно відповідно до зміни вимог користувача або умов функціонування системи.

$$\begin{aligned}
Cost_{norm}(X) &= (Cost_{max} - Cost(X)) / (Cost_{max} - Cost_{min}), \\
Perf_{norm}(X) &= (Perf(X) - Perf_{min}) / (Perf_{max} - Perf_{min}), \\
Rel_{norm}(X) &= (Rel(X) - Rel_{min}) / (Rel_{max} - Rel_{min}), \\
Inter_{norm}(X) &= (Inter(X) - Inter_{min}) / (Inter_{max} - Inter_{min}), \\
Sec_{norm}(X) &= (Sec(X) - Sec_{min}) / (Sec_{max} - Sec_{min}).
\end{aligned}$$

2.4 Обмеження, які накладаються на параметри моделі мультимарної взаємодії

Оптимізації розподілу обчислювальних завдань повинні задовольняти наступні обмеження [53, 54]:

- Обмеження доступного ресурсного потенціалу (*CRC, Computational Resource Constraints*):

$$\sum_i x_{ij} = 1, d_{ir} \leq Cap_j(r), \forall j \in \{1, 2, \dots, m\}, \forall r \in R_j, \quad (2.7)$$

де $Cap_j(r)$ – доступний ресурсний потенціал r у провайдера p_j , який характеризує граничну обчислювальну потужність, що може бути надана для виконання завдань. Це обмеження забезпечує, щоб сумарні вимоги всіх призначених завдань не перевищують доступні ресурси провайдера.

- Обмеження призначення (*AC, Assignment Constraints*):

$$\sum_{j=1}^m x_{ij} = 1, \forall i \in \{1, 2, \dots, k\}, \quad (2.8)$$

Кожне завдання має бути призначене рівно одному провайдеру, що гарантує повне виконання всіх обчислювальних завдань без дублювання.

- Обмеження *QoS* (*QC, Quality of Service Constraints*):

$$QoS_j(t_i) \geq q_i, \forall i, j: x_{ij} = 1, \quad (2.9)$$

де $QoS_j(t_i)$ – якість обслуговування, яку провайдер p_j може забезпечити для завдання t_i , а q_i – мінімальні вимоги до QoS для t_i . Це обмеження забезпечує дотримання визначених параметрів якості обслуговування.

- Обмеження залежностей (*DC, Dependency Constraints*):

$$start(t_i) \geq end(t_j), \forall t_j \in I_i, \quad (2.10)$$

де $start(t_i)$ і $end(t_i)$ – час початку і закінчення виконання завдання t_i відповідно. Це обмеження гарантує правильну послідовність виконання взаємозалежних завдань, забезпечуючи запуск завдань, які потребують вхідних даних від інших, лише після завершення останніх. Така послідовність особливо важлива для обчислювальних конвесрів, аналітичних процесів та розподілених транзакцій.

Обмеження даних і їх локалізації (*DLC, Data Locality Constraints*):

$$Loc(Data(t_i)) \in Allowed_Regions(t_i), \quad (2.11)$$

де $Data(t_i)$ – дані, необхідні для виконання завдання t_i , $Loc()$ – функція, що визначає географічне розташування даних, а $Allowed_Regions(t_i)$ – множина регіонів, у яких дозволено зберігати та обробляти дані для завдання t_i .

2.5 Класифікація рішень побудови мультимарного середовища

Існуючі рішення для побудови мультимарного середовища можна класифікувати за різними критеріями [55, 56]. Пропонуємо таксономію, представлену на рис. 2.1, яка представляє підходи за рівнем абстракції, технологічною парадигмою та способом розгортання.

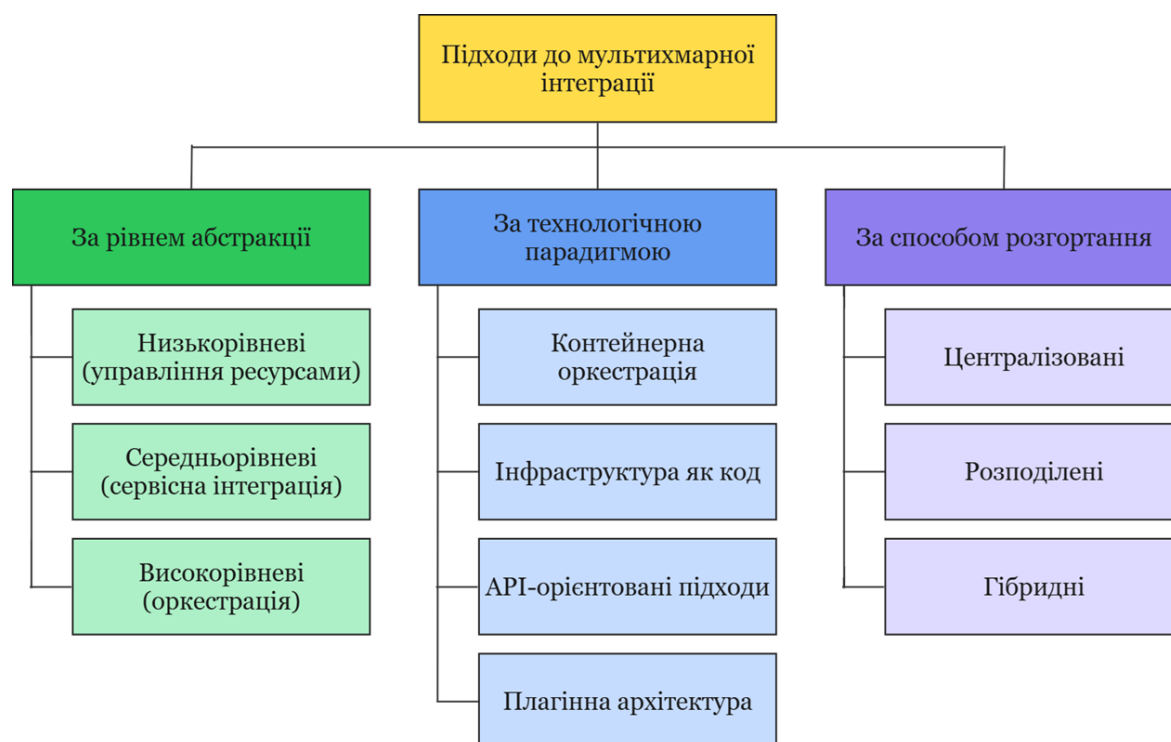


Рисунок 2.1 Таксономія підходів до мултихмарної інтеграції

В таблиці 2.1 представлено структурований порівняльний аналіз основних підходів до мултихмарної інтеграції.

Таблиця 2.1 Порівняльний аналіз підходів мултихмарної інтеграції

Категорія	Підхід	Ключові переваги	Обмеження	Середовище застосування
Контейнерна оркестрація	Kubernetes	<ul style="list-style-type: none"> - Уніфікована абстракція розгортання; - Декларативна конфігурація; - Автоматичне масштабування; - Самовідновлення 	<ul style="list-style-type: none"> - Складність налаштування; - Значна крива навчання; - Необхідність додаткових інструментів для повної мултихмарності 	Мікросервісні архітектури, великі розподілені системи
Інфраструктура як код	Terraform	<ul style="list-style-type: none"> - Декларативний синтаксис; - Управління станом; - Підтримка багатьох 	<ul style="list-style-type: none"> - Обмежена абстракція між провайдерами; - Потенційні проблеми з відкатом змін 	Управління інфраструктурою, автоматизоване розгортання

		провайдерів; - Попереднє планування змін		
Мультихмарні шлюзи	S3Proxy	- Єдиний API для доступу; - Прозора трансляція запитів; - Низька часова затримка	- Обмежений функціонал; - Можливі проблеми сумісності з нативними функціями	Застосунки зі спільними даними у різних хмарах
Платформи керування даними	Apache NiFi	- Візуальне проектування потоків; - Розширюваність; - Масштабованість	- Висока ресурсоемність; - Складність для простих сценаріїв	Обробка великих даних, ETL процеси
Хмарно-незалежні бібліотеки	Apache Libcloud	- Єдиний програмний інтерфейс; - Програмна портативність; - Низький поріг входження	- Підтримка «найменшого спільного знаменника» - Обмеження нативної функціональності	Крос-хмарні застосунки, програмне управління ресурсами
Безсерверні архітектури	AWS Lambda + Azure Functions	- Масштабування від нуля; - Оплата за використання; - Відсутність управління інфраструктурою	- Холодний старт; - Обмеження часу виконання; - Залежність від провайдера	Подієво-орієнтовані архітектури, мікрофункції

2.6 Формальний опис критеріїв визначення сумісності сервісів різних провайдерів мультихмарного середовища та способів їх взаємодії

Для кількісної оцінки ефективності різних підходів до мультихмарної інтеграції визначимо формальні метрики:

- Індекс сумісності взаємодії (*II*, *Interoperability Index*):

$$II(a) = (1/|P|) \sum_{i=1}^p \sum_{j=1}^p (|F_i \cap F_j| / (F_i \cup F_j)) \cdot w_{ij}, \quad (2.12)$$

де a – підхід, P – множина провайдерів, F_i – множина функцій провайдера i , доступних через підхід a , w_{ij} – вага взаємодії між провайдерами i та j .

- Коефіцієнт складності впровадження (*ImpE, Implementation Effort*):

$$ImpE(a) = \alpha \cdot T_{lev}(a) + \beta \cdot T_{conf}(a) + \gamma \cdot T_{maint}(a), \quad (2.13)$$

де T_{lev} , T_{conf} , T_{maint} – оцінка часу на розробку, конфігурацію та підтримку відповідно, а α , β , γ – вагові коефіцієнти.

- Продуктивність (*PM, Performance Metric*):

$$PM(a) = \left(\frac{1}{|O|}\right) \sum_{o \in O} (T_{baseline}(o)/T_a(o)), \quad (2.14)$$

де O – множина типових операцій, $T_{baseline}(o)$ – базовий час виконання операції o , $T_a(o)$ – час виконання операції з використанням підходу a .

- Комплексний показник ефективності (*CEI, Composite Efficiency Index*):

$$CEI(a) = w_{ii} \cdot II(a) + w_{ei} \cdot \left(1 - norm(ImpE(a))\right) + w_{ii} \cdot PM(a) + w_{sec} \cdot Sec(a), \quad (2.15)$$

де w – вагові коефіцієнти, $norm()$ – функція нормалізації, $Sec(a)$ – метрика безпеки для підходу a .

Запропоновані формальні метрики дозволяють провести об'єктивний кількісний аналіз ефективності різних підходів до організації мультихмарної взаємодії та обґрунтувати вибір оптимального рішення для конкретних сценаріїв використання.

Процедура застосування метрик складається з наступних етапів:

1. Визначення множини типових операцій $O = \{o_1, o_2, \dots, o_n\}$, що представляють характерні для цільового застосування шаблонів взаємодії з хмарною інфраструктурою.

2. Експериментальне вимірювання базових показників продуктивності, зусилля впровадження та рівня сумісності для кожного з підходів на множині визначених операцій.

3. Нормалізація отриманих показників для приведення їх до єдиної шкали оцінювання, що дозволяє коректно порівнювати різномірні метрики:

$$\text{norm}(v) = (v - v_{\min}) / (v_{\max} - v_{\min}) \quad (2.16)$$

4. Визначення вагових коефіцієнтів для комплексного показника ефективності відповідно до пріоритетів конкретного сценарію використання.

5. Обчислення комплексного показника для кожного з розглянутих підходів та ранжування альтернатив.

Проведений аналіз з використанням запропонованих метрик дозволяє зробити наступні висновки:

- Для сценаріїв з високими вимогами до гнучкості інфраструктури та мінімальними інвестиціями у впровадження найвищу ефективність демонструють підходи, засновані на принципі інфраструктури як коду, зокрема Terraform.

- У випадках, коли критичним є забезпечення однорідного інтерфейсу для взаємодії з різними хмарними середовищами, оптимальними є рішення з використанням хмарно-незалежних бібліотек.

- Для сценаріїв з інтенсивною обробкою даних у гетерогенному хмарному середовищі найбільш перспективними є платформи керування даними, що забезпечують високий рівень абстракції та автоматизації.

Таким чином, формальні метрики забезпечують теоретичну основу для раціонального вибору підходу до мультихмарної інтеграції та оптимізації процесу розподілу обчислень у гетерогенному хмарному середовищі.

2.7 Проблематика сумісності у мультихмарному середовищі

Гетерогенність хмарних ресурсів різних постачальників призводить до суттєвих невідповідностей у їх характеристиках, що ускладнює ефективне використання мультихмарних середовищ [57, 58]. Для формалізації цієї проблеми пропонуємо модель ресурсних невідповідностей.

Нехай для кожного типу ресурсу r (наприклад, обчислювальні екземпляри, сховища даних, мережеві сервіси) різні провайдери p_1, p_2, \dots, p_m пропонують свої

реалізації r_1, r_2, \dots, r_m . Кожна реалізація характеризується вектором атрибутів:
 $A(r_i) = \{a_{1i}, a_{2i}, \dots, a_{ni}\},$

де a_{ji} – j -й атрибут ресурсу r_i (наприклад, обчислювальна потужність, пам'ять, пропускна здатність, гарантії SLA).

Для оцінки невідповідності між ресурсами можна визначити функцію відстані:

$$D(r_i, r_j) = \sqrt{(\sum_{k=1}^n w_k \cdot (a_{ki} - a_{kj})^2)}, \quad (2.17)$$

де w_k – вагові коефіцієнти, що відображають важливість кожного атрибута.

Загальна невідповідність ресурсів усіх типів між двома провайдерами p_i та p_j може бути визначена як:

$$RD(p_i, p_j) = \sum_{r \in R} D(r_{ir}, r_{jr}) \cdot w_r, \quad (2.18)$$

де R – множина всіх типів ресурсів, r_{ir} – реалізація ресурсу типу r провайдером p_i , а w_r – вага ресурсу типу r в загальній архітектурі системи.

Висока невідповідність ресурсів призводить до:

1. Складності міграції навантаження між провайдерами.
2. Зниження ефективності балансування ресурсів.
3. Необхідності розробки додаткових адаптаційних шарів.
4. Зростання операційних витрат на управління гетерогенними ресурсами.

Регуляторні обмеження можна формалізувати наступним чином:

- $R = \{r_1, r_2, \dots, r_l\}$ – множина регуляторних вимог;
- $C_i = \{c_{i1}, c_{i2}, \dots, c_{il}\}$ – рівень відповідності провайдера p_i кожній вимозі r_j ,

де $c_{ij} \in [0, 1]$;

- $W = \{w_1, w_2, \dots, w_l\}$ – вагові коефіцієнти важливості кожної вимоги.

Загальний індекс регуляторної відповідності провайдера p_i можна визначити як:

$$RC(p_i) = \sum_{j=1}^l w_j \cdot c_{ij}, \quad (2.19)$$

А сумісність між провайдерами з точки зору регуляторних вимог:

$$PCC(p_i, p_j) = (\sum_{k=1}^l w_k \cdot \min(c_{ik}, v_{jk})) / (\sum_{k=1}^l w_k) \quad (2.20)$$

2.8 Оцінка ефективності рішень щодо побудови мультимарного середовища

2.8.1 Інтегрований показник оцінки ефективності архітектурних рішень

Комплексну оцінку ефективності різних архітектурних рішень при проєктуванні мультимарної інфраструктури пропонується виконувати за допомогою інтегрованого показника, який об'єднує окремі показники мультимарної взаємодії у єдину математичну структуру [59]. Інтегрований показник оцінки ефективності формалізується наступним чином:

$$E(S) = w_e \cdot C(S) + w_p \cdot P(S) + w_r \cdot R(S) + w_i \cdot I(S) + w_s \cdot S(S), \quad (2.21)$$

де:

S – стратегія розподілу обчислень;

$C(S)$ – нормалізована оцінка вартості;

$P(S)$ – нормалізована оцінка продуктивності;

$R(S)$ – нормалізована оцінка надійності;

$I(S)$ – нормалізована оцінка сумісності взаємодії;

$S(S)$ – нормалізована оцінка безпеки.

w_e, w_p, w_r, w_i, w_s – вагові коефіцієнти, специфічні для конкретного сценарію використання, причому $\sum w_j = 1$.

2.8.2 Нормалізація компонентів моделі

Для забезпечення коректного агрегування різнорідних метрик у єдину інтегральну оцінку застосовується процедура нормалізації, що приводить усі

компоненти моделі до єдиної шкали вимірювання. Нормалізацію компонентів здійснюємо за методом мінімаксної нормалізації:

$$C_norm(S) = (C(S) - C_min) / (C_max - C_min);$$

$$P_norm(S) = (P(S) - P_min) / (P_max - P_min);$$

$$R_norm(S) = (R(S) - R_min) / (R_max - R_min);$$

$$I_norm(S) = (I(S) - I_min) / (I_max - I_min);$$

$$S_norm(S) = (S(S) - S_min) / (S_max - S_min).$$

де X_min та X_max – мінімальне та максимальне значення відповідної метрики X у множині всіх можливих стратегій розподілу обчислень.

Для метрик, де оптимальним є максимальне значення (надійність, сумісність взаємодії, безпека), при агрегуванні застосовується доповнення до одиниці для приведення задачі до форми мінімізації.

2.8.3 Стохастична модель динамічної адаптації вагових коефіцієнтів

Вагові коефіцієнти w_e, w_p, w_r, w_i, w_s визначають відносну важливість кожного критерію в інтегральній оцінці. У моделях зі статичними параметрами ці коефіцієнти фіксуються на етапі проєктування системи. Однак, для ефективного функціонування в динамічному середовищі запропоновано стохастичну модель адаптації вагових коефіцієнтів:

$$w_i(t + 1) = w_i(t) + \alpha \cdot \nabla_w E(S, t) + \beta \cdot \eta(t), \quad (2.22)$$

де:

$w_i(t)$ – ваговий коефіцієнт для i -го критерію в момент часу t ;

α – коефіцієнт швидкості навчання;

$\nabla_w E(S, t)$ – градієнт функції ефективності за ваговими коефіцієнтами;

β – коефіцієнт стохастичної варіації;

$\eta(t)$ – випадковий шум з нормальним розподілом $N(0, \sigma^2)$.

Ця модель забезпечує динамічну адаптацію вагових коефіцієнтів до змін у середовищі функціонування системи, зберігаючи при цьому умову нормалізації $\sum_i w_i(t) = 1, \forall t$, що досягається через додаткову процедуру нормалізації після кожної ітерації оновлення:

$$\tilde{w}_i(t+1) = w_i(t+1) / \sum_j w_j(t+1). \quad (2.23)$$

2.8.4 Багатокритеріальна оптимізація при проєктуванні мультимарної інфраструктури

Задача вибору оптимальної стратегії взаємодії сервісів при проєктуванні мультимарної інфраструктури формулюється як задача багатокритеріальної оптимізації:

$$S^* = \operatorname{argmin}_{S \in \Omega} E(S), \quad (2.24)$$

де Ω – множина допустимих архітектурних рішень, що визначають ефективний розподіл обчислювальних завдань.

Для вирішення цієї задачі застосовуються методи багатокритеріальної оптимізації, зокрема:

1. Метод скаляризації – формування єдиного критерію оптимізації як зваженої суми окремих критеріїв, що відповідає запропонованій інтегральній моделі.
2. Метод Парето-оптимізації – пошук множини недомінованих рішень, для яких покращення одного критерію неможливе без погіршення хоча б одного іншого критерію.
3. Метод лексикографічного упорядкування – встановлення суворої ієрархії критеріїв за їх пріоритетністю та послідовна оптимізація за кожним критерієм з урахуванням обмежень, отриманих на попередніх кроках.

2.8.5 Стратегія взаємодії сервісів при проєктуванні мультимарної інфраструктури

Динамічне середовище мультимарної інфраструктури характеризується постійними змінами параметрів, таких як доступність ресурсів, вартість послуг, навантаження на систему тощо. Для забезпечення ефективності в таких умовах запропоновано механізм динамічного оновлення архітектурних рішень при проєктуванні мультимарної інфраструктури:

$$S(t + 1) = \operatorname{argmin}_{S \in \Omega(t + 1)} E(S, t + 1), \quad (2.25)$$

де:

$S(t + 1)$ – оптимальна стратегія розподілу в момент часу $t + 1$;

$E(S, t + 1)$ – інтегральна оцінка ефективності стратегії S з урахуванням стану системи в момент часу $t + 1$;

$\Omega(t + 1)$ – множина допустимих стратегій з урахуванням обмежень, актуальних на момент часу $t + 1$.

Для забезпечення стабільності функціонування системи та мінімізації накладних витрат на перерозподіл обчислень вводиться додаткове обмеження на частоту оновлення стратегії:

$$d(S(t + 1), S(t)) \leq \Delta_{\max}, \quad (2.26)$$

де:

$d(S_1, S_2)$ – функція відстані між стратегіями, що визначає ступінь їх відмінності;

Δ_{\max} – максимально допустима зміна стратегії за одну ітерацію.

Ця модель дозволяє формалізувати процес вибору оптимальної стратегії розподілу обчислень у гетерогенному мультимарному середовищі та створює теоретичний фундамент для розробки автоматизованих систем динамічного управління розподілом обчислювальних завдань.

ВИСНОВКИ

1. Формалізовано математичну модель мультихмарного середовища з визначенням множини хмарних провайдерів, обчислювальних завдань, сервісів та обмежень, що створює теоретичний фундамент для подальшої розробки методів проєктування ефективної інфраструктури.

2. Визначено ключові цільові функції оптимізації архітектурних рішень при проєктуванні мультихмарної інфраструктури, включаючи мінімізацію вартості, часу виконання, максимізацію надійності та їх комбінації з врахуванням вагових коефіцієнтів.

3. Запропоновано структуровану таксономію варіантів взаємодії в мультихмарному середовищі та проведено порівняльний аналіз існуючих підходів до його побудови, що дозволило визначити оптимальні рішення для різних сценаріїв використання гетерогенних хмарних середовищ.

4. Формалізовано ключові фактори ресурсних невідповідностей у мультихмарних середовищах та запропоновано відповідні метрики для їх кількісної оцінки.

5. Запропоновано інтегральний показник оцінки ефективності мультихмарних рішень, що дозволяє формалізувати процес вибору оптимальної стратегії розподілу обчислень з урахуванням множини критеріїв.

РОЗДІЛ 3

МЕТОДОЛОГІЯ ПРОЄКТУВАННЯ АРХІТЕКТУРИ ОБЧИСЛЮВАЛЬНОЇ ІНФРАСТРУКТУРИ У ГЕТЕРОГЕННОМУ МУЛЬТИХМАРНОМУ СЕРЕДОВИЩІ

3.1 Постановка задачі

У цьому підрозділі формалізовано задачу проєктування архітектури обчислювальної інфраструктури у гетерогенному мультимарному середовищі, що дозволить ефективно розподіляти навантаження між різними хмарними сервісами з урахуванням їх гетерогенної природи.

Дано:

1. Множина обчислювальних завдань $T = \{t_1, t_2, \dots, t_n\}$, кожне з яких характеризується набором параметрів: обчислювальне навантаження, обсяг даних, вимоги до конфіденційності, пріоритет та максимально допустимий час виконання.

2. Множина сервісів мультимарної платформи $R = \{r_1, r_2, \dots, r_m\}$, що належать різним хмарним провайдерам $P = \{p_1, p_2, \dots, p_k\}$ та характеризуються різною обчислювальною потужністю, доступною пам'яттю, рівнем безпеки, доступністю та вартістю використання..

3. Динамічно змінювані параметри середовища $E = \{e_1, e_2, \dots, e_k\}$, які включають:

- Затримки мережі $L(t)$, що можуть бути виражені як $L(t) = L_{base} + L_{var}(t) + L_{cong}(t)$, де L_{base} – базова затримка мережі, що залежить від географічного розташування ресурсів, $L_{var}(t)$ – варіативна складова затримки, що змінюється протягом часу, $L_{cong}(t)$ – додаткова затримка, спричинена перевантаженням мережі в момент часу t .

- Навантаження на обчислювальні ресурси $U(t)$, що розраховується як: $U(t) = \sum_i (CPU_i(t) + MEM_i(t) + IO_i(t))$, де $CPU_i(t)$ – використання процесора i -

тим завданням в момент часу t , $MEM_i(t)$ – використання пам'яті i -тим завданням в момент часу t , $IO_i(t)$ – інтенсивність операцій введення/виведення i -того завдання в момент часу t .

- Зміни тарифів провайдерів $P(t)$, що може бути представлена як: $P(t) = P_{base} \cdot (1 + \Delta P(t))$, де P_{base} – базовий тариф для конкретного ресурсу, $\Delta P(t)$ – відносна зміна тарифу в момент часу t , що може залежати від попиту, сезонності, спеціальних пропозицій та інших факторів.

- Коефіцієнт доступності ресурсів $A(t)$, який можна розрахувати за формулою: $A(t) = MTBF / (MTBF + MTTR(t))$, де $MTBF$ (Mean Time Between Failures) – середній час між відмовами, $MTTR(t)$ – середній час відновлення, який може варіюватися залежно від навантаження на сервіси технічної підтримки провайдера.

- Індекс енергоефективності обчислень $E(t)$, що розраховується як: $E(t) = COMP(t) / POWER(t)$, де $COMP(t)$ – обчислювальна продуктивність в момент часу t , $POWER(t)$ – спожита енергія в момент часу t .

- Динамічний показник безпеки $S(t)$, що враховує зміни в загрозах безпеки та механізмах захисту: $S(t) = \sum_i w_i \cdot s_i(t)$, де w_i – вага i -того аспекту безпеки, $s_i(t)$ – оцінка i -того аспекту безпеки в момент часу t .

Визначити:

Оптимальну архітектуру взаємодії хмарних сервісів $X = \{x_{ij}\}$ між ресурсами різних хмарних провайдерів, що мінімізує цільову функцію, яка враховує:

- загальний час виконання завдань;
- сумарну вартість використання ресурсів;
- рівень безпеки обробки даних.

3.1.1 Формалізація проблеми вибору архітектури мультимарної інфраструктури

Проблему проєктування обчислювальної інфраструктури у гетерогенному мультимарному середовищі можна формалізувати як задачу оптимального розміщення скінченної множини обчислювальних завдань $T = \{t_1, t_2, \dots, t_n\}$ на множині сервісів мультимарної платформи $R = \{r_1, r_2, \dots, r_m\}$, що належать різним хмарним провайдерам $P = \{p_1, p_2, \dots, p_k\}$.

Кожне завдання $t_i \in T$ характеризується вектором атрибутів $t_i = \langle w_i, d_i, c_i, s_i, \tau_i \rangle$, де:

w_i – обчислювальне навантаження завдання (у FLOP);

d_i – обсяг даних, необхідних для завдання (у байтах);

c_i – вимоги до конфіденційності даних (за шкалою від 0 до 1);

s_i – пріоритет завдання (за шкалою від 0 до 1);

τ_i – максимально допустимий час виконання.

Кожен хмарний сервіс $r_j \in R$ характеризується вектором атрибутів: $r_j = \langle p_j, \lambda_j, \mu_j, s_j, \alpha_j, \gamma_j \rangle$, де:

$p_j \in P$ – хмарний провайдер сервісу;

λ_j – обчислювальна потужність (у FLOPS);

μ_j – доступна пам'ять (у байтах);

s_j – рівень безпеки сервісу (у FLOPS/Вт);

α_j – доступність сервісу (у відсотках);

γ_j – вартість використання сервісу (у грошових одиницях/час).

3.1.2 Математична модель вибору архітектури для мультимарного розподілу задач

Нехай $X = \{x_{ij}\}$ – матриця розміщення завдань, де:

$x_{ij} = \{1, \text{ якщо завдання } t_i \text{ призначено ресурсу } r_j$
 $0, \text{ в іншому випадку}\}$

Модель вибору архітектури для мультихмарного розподілу задач формалізується як задача багатокритеріальної оптимізації $\min F(X) = \{T(X), C(X), S(X)\}$, де:

$T(X)$ – функція загального часу виконання;

$C(X)$ – функція загальної вартості;

$S(X)$ – функція оцінки рівня безпеки.

Ці функції визначаються наступним чином:

$$T(X) = \max(i \in \{1, \dots, n\}) \sum(j = 1 \text{ to } m) x_{ij} \cdot (w_i/\lambda_j);$$

$$C(X) = \sum(i = 1 \text{ to } n) \sum(j = 1 \text{ to } m) x_{ij} \cdot \gamma_j \cdot (w_i/\lambda_j);$$

$$S(X) = \sum(i = 1 \text{ to } n) \sum(j = 1 \text{ to } m) x_{ij} \cdot c_i \cdot s_j,$$

де s_j – рівень безпеки сервісу j , c_i – вимоги до безпеки завдання i .

При цьому необхідно пам'ятати про обмеження:

1. Кожне завдання повинно бути призначене рівно одному ресурсу:

$$\sum(j = 1 \text{ to } m) x_{ij} = 1, \forall i \in \{1, \dots, n\}.$$

2. Обсяг даних, необхідних для виконання завдань на ресурсі, не повинен перевищувати доступну пам'ять:

$$\sum(i = 1 \text{ to } n) x_{ij} \cdot d_i \leq \mu_j, \forall j \in \{1, \dots, m\}.$$

2. Час виконання завдання на призначеному ресурсі не повинен перевищувати максимально допустимий:

$$w_i/\lambda_j \leq \tau_i, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, m\} \text{ такі, що } x_{ij} = 1.$$

3. Рівень безпеки, що забезпечується сервісом, повинен задовольняти вимоги завдання:

$$c_i \leq s_j, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, m\} \text{ такі, що } x_{ij} = 1,$$

де s_j – рівень безпеки, який забезпечує сервіс r_j .

3.2 Проектування взаємодії сервісів у мультимарному середовищі

Розподілена система визначається як комплекс автономних обчислювальних компонентів, що взаємодіють між собою за допомогою мережових протоколів для досягнення спільної обчислювальної мети. Розподілені системи покликані усунути вузькі місця продуктивності та централізовані точки відмови в обчислювальних середовищах. У централізованих обчислювальних системах усі обчислення здійснюються на єдиному обчислювальному ресурсі. Основна відмінність між централізованими та розподіленими системами полягає в шаблоні зв'язку між вузлами системи [56]. У розподілених системах кожен вузол може взаємодіяти з іншим, оминаючи центральний. Це забезпечує стійкість до відмов і дозволяє більш ефективно використовувати ресурси.

Проектування архітектури обчислювальної інфраструктури у мультимарному середовищі стосується оптимального розміщення та взаємодії сервісів між декількома хмарними платформами з можливістю адаптації в реальному або майже реальному часі. Така система розроблена для оптимізації продуктивності, вартості, надійності та масштабованості шляхом використання сильних сторін різних хмарних постачальників та їхніх послуг.

Динамічні робочі навантаження демонструють значні нерегулярні коливання інтенсивності і можуть спостерігатися раптові сплески потреб у ресурсах. Наприклад, вебсайт онлайн-магазину може мати великий трафік під час швидкого розпродажу. Ці робочі навантаження також можуть відрізнятися залежно від конкретних періодів часу, наприклад, сезонів святкових покупок або кінцевих термінів подання податкової декларації і іншої документації. Робочі навантаження, керовані подіями, які ініціюються певними подіями, як обробка вхідних даних із пристроїв IoT або каналів соціальних мереж, і робочі навантаження в реальному часі, такі як фінансова торгівля в реальному часі або системи моніторингу, також є поширеними прикладами динамічних навантажень.

Можна визначити наступні характеристики динамічних навантажень:

- Мінливість у користуванні ресурсами, що ускладнює їх оптимальний розподіл.
- Неоднорідність та споживання різних хмарних ресурсів.
- Вимагають здатності швидко збільшувати або зменшувати ресурси, щоб задовольнити мінливий попит.
- Потребують інтенсивного управління. Оптимальна продуктивність для динамічних навантажень вимагає динамічного регулювання розподілу ресурсів [61].

Динамічні робочі навантаження становлять проблему в багатохмарному середовищі через необхідність керувати ресурсами кількох хмарних постачальників з різними моделями ціноутворення, пропозиціями послуг і характеристиками продуктивності.

Серед сучасних підходів можна виділити:

- Безсерверні архітектури, які дозволяють розробникам розгортати код без керування основною інфраструктурою. Мультихмарні системи використовують безсерверні платформи різних постачальників, таких як AWS Lambda, Google Cloud Functions і Azure Functions, для високодинамічних і керованих подіями завдань. Безсерверні системи відмінно справляються з непередбачуваними робочими навантаженнями завдяки автоматичному масштабуванню та виконанню лише після запуску [62].
- Функції як послуга (FaaS) – це підмножина безсерверних обчислень, де окремі функції розгортаються та виконуються за вимогою. У багатохмарному середовищі платформи FaaS можна використовувати для розподілу конкретних завдань у хмарі, що пропонує найкращу продуктивність або економічну ефективність у певний час. Функція як послуга (FaaS) стає популярним вибором для масштабованого та економічно ефективного розгортання робочих навантажень. Усі

популярні постачальники надають платформи FaaS [63].

- Контейнери створюють легкі, портативні та узгоджені середовища для запуску програм. Такі інструменти, як Docker і Kubernetes, полегшують розподіл робочого навантаження між кількома хмарами, абстрагуючи відмінності базової інфраструктури. Багатохмарні можливості Kubernetes, особливо з такими інструментами, як Anthos або OpenShift, дозволяють безперебійно керувати контейнерними програмами в хмарах [64].

- Cloud Bursting – це гібридний підхід, коли завдання виконуються в основному в приватній хмарі, але використовують публічні хмари під час стрибків попиту. Ця техніка забезпечує економічну ефективність, зберігаючи контроль над критично важливими даними [65].

3.2.1 Формальна модель динамічної адаптації мультихмарної інфраструктури

Динамічну адаптацію мультихмарної інфраструктури можна формалізувати через представлення параметрів системи як функцій часу. Нехай стан системи в момент часу t описується:

$\lambda_j(t)$ – обчислювальна потужність ресурсу r_j в момент часу t ;

$\mu_j(t)$ – доступна пам'ять ресурсу r_j в момент часу t ;

$\gamma_j(t)$ – вартість використання ресурсу r_j в момент часу t .

Функція переходу системи з одного стану в інший може бути представлена як $S(t + \Delta t) = F(S(t), A(t), E(t))$, де:

$S(t)$ – стан системи в момент часу t ;

$A(t)$ – вектор дій системи управління ресурсами;

$E(t)$ – вектор зовнішніх подій;

F – функція переходу, що описує еволюцію системи.

Для оцінки гетерогенності мультимарного середовища можна використовувати наступні метрики:

1. Індекс гетерогенності обчислювальної потужності:

$$H\lambda = \sigma\lambda/\mu\lambda, \quad (3.1)$$

де $\sigma\lambda$ – стандартне відхилення обчислювальної потужності сервісів, а $\mu\lambda$ – середнє значення.

2. Індекс гетерогенності вартості:

$$H\gamma = \sigma\gamma/\mu\gamma, \quad (3.2)$$

де $\sigma\gamma$ – стандартне відхилення вартості використання сервісів, а $\mu\gamma$ – середнє значення.

3. Загальний індекс гетерогенності мультимарної інфраструктури:

$$H_r = \sqrt{w\lambda H\lambda^2 + w\gamma H\gamma^2 + ws Hs^2}, \quad (3.3)$$

де $w\lambda$, $w\gamma$, ws – вагові коефіцієнти, що відображають важливість характеристик обчислювальної потужності, вартості та безпеки відповідно.

3.3 Проктування мультимарної архітектури

Архітектура системи для динамічно розподілених обчислень у багатомарному середовищі об'єднує модульні компоненти, які обробляють розподіл завдань, автоматизацію, прийняття рішень і моніторинг. Запропонована архітектура дозволяє ефективно розподіляти обчислювальні ресурси в межах мережі, що сприяє їх оптимальному використанню і покращенню швидкості реагування.

У традиційних централізованих системах один сервер або процесор виконує всі завдання та операції. Навпаки, розподілені системи розподіляють навантаження між взаємопов'язаними пристроями.

У розподіленій системі кожен вузол (окремий пристрій або сервер) працює незалежно, але співпрацює з іншими для досягнення спільних цілей. Зв'язок і

координація між цими вузлами мають вирішальне значення для безперебійного функціонування системи [66].

Можна виділити чотири поширені типи архітектури розподіленої системи: клієнт-сервер, однорангова, 3-рівнева та N-рівнева [67]:

1. Клієнт-сервер. Дані отримуються з сервера клієнтами, які згодом форматують їх і показують користувачеві. Кінцевий користувач також може вносити поправки на стороні клієнта і відновити їх на сервері, щоб зробити їх постійними. Архітектура клієнт-сервер має одну стандартну конструктивну особливість: централізовану безпеку.

2. Однорангова архітектура (peer-to-peer (P2P)). Працює за принципом децентралізованої системи. У ній немає додаткових комп'ютерів, які використовуються для надання послуг або управління ресурсами. Комп'ютери в системі, які називаються одноранговими, мають рівномірно розподілені обов'язки. Ними можуть користуватися як клієнти, так і сервери.

3. Трирівнева архітектура. Клієнти в трирівневій архітектурі не мають статусу. Щоб спростити розгортання програми, інформація не зберігається на стороні клієнта, а на проміжному рівні. Трирівнева архітектура найбільш поширена для вебдодатків.

4. N-рівнева архітектура. Середній рівень взаємодіє з іншою службою для отримання інформації. N-рівнева архітектура зазвичай використовується, коли програмі або серверу потрібно пересилати запити до додаткових корпоративних служб у мережі [64, 65].

Незважаючи на різниці цих архітектур, розподілена система повинна мати наступне [68]:

- Масштабованість, що означає здатність зростати в міру збільшення навантаження. А це є можливим завдяки додаванню в мережу додаткових процесорів.

- Паралелізм, що означає, що компоненти працюватимуть одночасно, із завданнями, що виконуються без певного порядку та з різною швидкістю.
- Відмовостійкість. Якщо один вузол виходить з ладу, інші можуть продовжувати працювати без порушення загального обчислення.
- Прозорість, щоб кінцевий користувач або сторонній розробник бачив розподілену систему як єдину обчислювальну цілісність, що дозволяло б їм взаємодіяти з певним логічним пристроєм, не турбуючись про всю архітектуру,
- Відтворюваність, яка надає інформацію та повідомлення для спільного використання та забезпечує узгодженість між доступними ресурсами (такими як компоненти програмного або апаратного забезпечення), таким чином підвищуючи доступність, надійність і відмовостійкість.

3.3.1 Основні компоненти системи

Складність розподілених систем зростає в залежності від сценарію використання та вимог до продуктивності. Проте можна виділити наступні необхідні складові системи [69]:

- Розподіл завдань та рівень оркестровки. Центральний модуль, який відповідає за розподіл робочих навантажень на окремі завдання, призначення їх відповідним хмарним ресурсам і керування виконанням завдань. Виконує функції постановки завдань у чергу та планування; оркестровка робочого навантаження в кількох хмарах; перепризначення завдань у режимі реального часу у разі збою ресурсів або неоптимальної продуктивності.
- Модуль прийняття рішень. Визначає оптимальне розміщення завдань на основі таких критеріїв, як вартість, затримка, безпека та постійність даних. Тут присутні спеціальні алгоритми (наприклад, методи багатокритеріального аналізу), які інтегруються з інструментами типу Terraform або Pulumi для розгортання. Реалізуються функції аналізу витрат, продуктивності, безпеки і перевірки відповідності організаційним політикам.

- Інтеграція з інструментами автоматизації. Платформи автоматизації для надання хмарних ресурсів і керування ними. В даному модулі відбувається автоматичне масштабування та деініціалізація ресурсів, а також управління мультихмарною конфігурацією.

- Безпека і моніторинг. Моніторинг у реальному часі для відстеження продуктивності, витрат і виконання завдань у хмарах, а також інструменти за контролем норм безпеки. Функції: відстеження використання ресурсів і показників виконання завдань; виявлення аномалій та вузьких місць у виконанні завдань; зворотній зв'язок з модулем прийняття рішень; керуйте ключами шифрування та контролем доступу.

Динамічно розподілена обчислювальна система має в основі архітектуру, де планування, керування ресурсами, зберігання та зв'язок працюють разом для ефективної обробки робочих навантажень. Система динамічно адаптується до вимог, які є змінюваними. Це робить її ідеальною для хмарних обчислень, аналізу великих даних і робочих навантажень, а також використання моделей ШІ.

3.3.2 Взаємодія між компонентами

Функціональний робочий процес складається з наступних кроків:

1. Робочі навантаження передаються на рівень розподілу завдань, який розбиває їх на завдання та призначає початкові пріоритети.

2. Модуль прийняття рішень оцінює доступні хмарні ресурси за допомогою попередньо визначених критеріїв (наприклад, вартість, продуктивність) і надає ресурси.

3. Завдання плануються на наданих ресурсах за допомогою рівня оркестровки (наприклад, Kubernetes). Контейнери, безсерверні функції або віртуальні машини виконують завдання.

4. Модуль моніторингу відстежує виконання завдань і продуктивність ресурсів. Якщо продуктивність відхиляється від очікуваного рівня, відгук

надсилається до модуля прийняття рішень для перерозподілу чи оптимізації.

5. Інструменти автоматизації збільшують або зменшують ресурси залежно від попиту. У разі збою система динамічно перепризначає завдання іншим хмарам.

Функціональні зв'язки можна представити у вигляді таблиці (Таблиця 3.1):

Таблиця 3.1 Функціональні зв'язки між компонентами

Компонент	Взаємодія	Призначення
Рівень розподілу завдань	Модуль прийняття рішень, рівень оркестровки	Призначення та планування завдання
Прийняття рішень	Рівень розподілу завдань, засоби автоматизації	Оптимальний розподіл ресурсів і забезпечення
Інтеграція з автоматизацією	Модуль прийняття рішень, управління ресурсами	Автоматизація надання і масштабування інфраструктури
Моніторинг і аналітика	Модуль прийняття рішень, виконання завдань	Дані про продуктивність у реальному часі та відгуки
Рівень безпеки та відповідності	Всі компоненти	Забезпечення безпеки та відповідність нормативним вимогам

3.4 Метод багатокритеріального аналізу для автоматизованого вибору оптимальної хмарного інструментарію

На сьогоднішній день доступна широка низка інструментів для реалізації мультихмарної інтеграції. Проте, кожна організація має власні потреби. Однакова стратегія реалізації може демонструвати різну ефективність залежно від специфіки користувацьких потреб.

Перед розгортанням власної мультихмарної системи, відповідно до рекомендацій, представлених у роботах [70, 71] важливо врахувати наступні критерії:

- Вартість. Необхідно чітко розуміти SLA і політику ціноутворення. У різних постачальників можуть значно відрізнятися умови використання кожного з компонентів.

- Складність. Системна архітектура, розподілена між кількома хмарами

(регіонами), створює складність на рівні архітектури та може вимагати змін на рівні додатків, щоб бути більш стійкими до затримок і/або мати можливість спілкуватися з базою даних, яку перенесено до іншої хмара для цілей відновлення після відмови.

- Швидкість. Хмара дає більше можливостей контролювати швидкість або затримку сайту/програми. Наприклад, можна запускати різні типи екземплярів відповідно до потреб програми. Для додатків, які інтенсивно працюють з користувачем, додаткова затримка, яка є результатом міжхмарного/регіонального зв'язку, може бути неприйнятною.

- Хмарна портативність. Хоча може бути простіше використовувати один із інструментів або служб хмарного постачальника, як балансування навантаження або службу бази даних, важливо усвідомлювати, що якщо і коли знадобиться перемістити цей конкретний рівень архітектури до іншого хмарного постачальника, потрібно буде відповідно змінити свою архітектуру.

- Безпека. Для мультихмарної архітектури важливо розуміти, що міжхмарний/регіональний зв'язок здійснюється через загальнодоступний Інтернет і може викликати проблеми безпеки, які потрібно буде вирішити за допомогою певного типу шифрування даних або технології VPN.

Проте критеріїв може бути значно більше. І для вибору мультихмарного підходу бажано задіяти модель, яка найкращим чином дозволила б підібрати рішення під визначені критерії.

Так, у дослідженнях [72-74] запропоновано багатокритеріальні моделі на основі методів АНР і TOPSIS.

В якості підходів представлені:

- Мультихмарні шлюзи;
- платформи керування даними;
- хмарно-незалежні бібліотеки.

В якості критеріїв обрані:

- вартість;
- продуктивність;
- зусилля впровадження;
- безпека.

Також були означені пріоритети критеріїв під корпоративні і академічні задачі. Результат моделювання показав, що для корпорації більш релевантним є використання хмарно-незалежних бібліотек і платформ керування даними, а для академії – шлюзи зберігання даних.

Але варто врахувати, що при збільшенні кількості параметрів і підходів, складність побудови таких моделей значно збільшиться. Існують бібліотеки, які можуть моделювати багатокритеріальні проблеми прийняття рішень, особливо у поєднанні з технікою машинного навчання. Наприклад, Python-бібліотека `scikit-learn`.

Так, з цим інструментом достатньо виконати наступні етапи:

- Підготовка даних. Потрібно буде представити всі досліджувані багатохмарні стратегії і нормалізовані показники критеріїв.
- Надання вагів критеріям за їхньою важливістю.
- Моделювання та рейтинг, за якими стратегії ранжуються на основі зважених критеріїв, що дозволяє зробити оптимальний вибір.

Використання подібних бібліотек має наступні переваги:

- Автоматизація попередньої обробки (наприклад, нормалізація, зважування) і добре масштабування з більшими наборами даних, що є більш придатним для динамічних і складних систем.
- Моделі машинного навчання (наприклад, дерева рішень, кластеризація) здатні виявляти приховані патерни в історичних даних, що дозволяє адаптивно коригувати параметри прийняття рішень.
- Здатність обробляти невизначеність у вихідних даних дозволяє забезпечити

більш стабільні та обґрунтовані рекомендації.

Враховуючи складність хмарної сумісності та різноманітність мультихмарних підходів, модель багатокритеріального аналізу прийняття рішень (MCDA) дозволяє проводити комплексний аналіз, який поєднує об'єктивні та суб'єктивні оцінки різних параметрів.

Як вже зазначалося вище, для мультихмарних систем важливими параметрами є продуктивність, безпека, сумісність, залежність від сервісів, регуляторні вимоги та витрати тощо. Також кількість самих підходів теж є доволі великою. Вважаючи на це, було обрано метод АНР (Analytic Hierarchical Process), який дозволяє оперувати з великою кількістю альтернатив і параметрів і може інтегрувати як якісні, так і кількісні дані [75, 76].

Щоб створити модель MCDA/АНР, визначаємо таку структуру:

- Рівень 1. Визначити найбільш відповідне рішення серед підходів багатохмарного доступу.
- Рівень 2. Критерії: продуктивність, вартість, безпека тощо.
- Рівень 3. Альтернативи: представляють конкретні підходи (багатохмарні шлюзи зберігання даних, платформи керування даними, хмарно-незалежні бібліотеки).

Алгоритм MCDA/АНР має наступні кроки:

1. Для кожного критерію створюється матриця попарних порівнянь альтернатив, де кожний елемент набуває значення від 1 до 9, причому однакові альтернативи приймають значення 1. Іншими цифрами показано перевагу однієї альтернативи над іншою. Чим більше число, тим більша перевага.

2. Далі створюється матриця попарного порівняння критеріїв, де аналогічно оцінюється перевага одного критерія над іншим. Для корпоративного і академічного підходів ці матриці будуть різними.

3. Нормалізація матриць попарного порівняння критеріїв.

4. Обчислення вектору пріоритетів, який покаже ранжування альтернатив (найбільшому значенню відповідає краща альтернатива).

5. Перевірка чутливості отриманих параметрів. Якщо значення не задовольняють певному пороговому значенню, необхідно провести корегування матриць попарного порівняння.

3.4.1 Формалізація процесу вибору інструментарія мульти-хмарної взаємодії

Процес вибору інструментарія мульти-хмарної взаємодії можна формалізувати як задачу багатокритеріального прийняття рішень. Нехай маємо множину альтернатив $A = \{a_1, a_2, \dots, a_n\}$, що представляють різні хмарні платформи, та множину критеріїв $C = \{c_1, c_2, \dots, c_m\}$, за якими оцінюються ці альтернативи.

Кожен критерій c_j має вагу w_j , а для кожної альтернативи a_i та критерію c_j визначено оцінку v_{ij} . Задача формулюється як знаходження альтернативи a^* , яка максимізує загальну корисність $a^* = \operatorname{argmax}(a_i \in A) U(a_i)$, де $U(a_i)$ – функція корисності альтернативи a_i , наприклад, зважена сума:

$$U(a_i) = \sum_{j=1}^m w_j \cdot v_{ij} \quad (3.4)$$

Дана формалізація дозволяє інтегрувати метод АНР, описаний у наступному підрозділі, у загальну модель прийняття рішень.

3.4.2 Побудова моделі АНР

Для вибору найкращої мултихмарної стратегії було проведено експеримент і визначено наступні критерії:

- Продуктивність у значенні швидкості доступу до файлів:

$$P = \frac{\sum_{i=1}^n T_i S_i}{\sum_{i=1}^n S_i}, \quad (3.5)$$

де S_i – розмір i -го файлу,

T_i – час доступу до i -го файлу,

n – кількість файлів.

- Безпека – може складатися із багатьох складових, у даному випадку було використано вагові коефіцієнти:

$$S = W_a + W_s + W_l \quad (3.6)$$

де W_a – керування ідентифікацією та доступом,

W_s – шифрування,

W_l – логування та моніторинг.

- Зусилля впровадження – оцінка консалтингових агенцій в області хмарних обчислень у вигляді людино-годин, використаних для розгортання:

$$ImpE = \frac{1}{m} \sum_{i=1}^m ImpE_i \quad (3.7)$$

де $ImpE_i$ – середня оцінка експерта по впровадженню,

n – кількість експертів.

- Вартість – сума усіх складових вартості надання послуг: вартість за екземпляр, запит, розміщення тощо:

$$C = \sum_{i=1}^K C_i, \quad (3.8)$$

де K – кількість цінових факторів.

У якості підходів було обрано:

- Мультихмарні шлюзи зберігання даних (S3Proxy);
- Платформи керування даними (Apache NiFi);
- Хмарно-незалежні бібліотеки (Apache Libcloud).

Варто відмітити, що для оцінки реальних систем може бути включено набагато більше параметрів і альтернатив.

Для експерименту також було визначено пріоритети критеріїв для двох типів задач: академічних і корпоративних, де C – вартість; P – продуктивність; IE – зусилля впровадження; S – безпека.

- Для академічних рішень від найбільш вагомого: *C, IE, S, P*.
- Для корпоративних рішень від найбільш вагомого: *C, S, P, IE*.

Експериментальні дані, детально описані у [7], представимо у вигляді наступної таблиці (Таблиця 3.2):

Таблиця 3.2 Загальна таблиця критеріїв

Підхід	Середня продуктивність, с	Середня вартість за ТВ, USD	Зусилля впровадження, людино-години	Безпека (вагові коефіцієнти)
Хмарно-незалежні бібліотеки	0,27	117,3	48	3
Мультихмарні шлюзи зберігання даних	0,34	121,7	47	3
Платформи керування даними	7,24	126,1	87	8

Після проходження усіх кроків алгоритму АНР отримано наступний результат (Таблиця 3.3):

Таблиця 3.3 Результати

Підхід	Академічні рішення	Корпоративні рішення
Хмарно-незалежні бібліотеки	3	3
Мультихмарні шлюзи зберігання даних	1	2
Платформи керування даними	2	1

MCDA дозволяє отримати ранжування мультихмарних стратегій, в залежності від користувацьких потреб, проте є ряд недоліків такого моделювання:

- У методі АНР є значний суб'єктивний вплив. При побудові матриць порівнянь може бути велика кількість варіацій, які досить сильно впливають на кінцевий результат
- Збільшення критеріїв і альтернатив значно ускладнить процес побудови

моделі.

- Впровадження і підтримка MCDA можуть бути складними і трудомісткими.

Враховуючи наведені обмеження і недоліки, в роботі запропоновано використати методи машинного навчання для підтримки прийняття рішень у мультимарі.

3.4.3 Онтологічна модель для побудови мультимарного середовища

Маючи набір альтернатив і критеріїв $CR=\{C, P, S, IE\}$, формалізуємо дану предметну область. Для цього пропонується онтологічна модель для формування найкращих стратегій вибору мультимарних підходів, представлена у вигляді орієнтованого графа (рис.3.1). Це дозволяє одночасно відображати велику кількість об'єктів і зв'язків між ними. Вузли графа показують основні поняття або об'єкти [2, 77]. У нашому випадку це багатомарні підходи, критерії та значення пріоритетів. Стрілки показують зв'язки або взаємодію між ними. Кожен вузол з'єднується з іншим. Параметри утворюють ієрархічну структуру, де підходи знаходяться на одному рівні, а критерії та очікувана оцінка – на інших.

Таким чином, онтологія може бути представлена у такому вигляді:

$$O = \langle C, A, V, I, R \rangle, \quad (3.9)$$

де C – набір усіх ключових сутностей або концепцій

$$C = \{CAL, SG, DMP\}, \quad (3.10)$$

де SG – мультимарні шлюзи зберігання даних,

DMP – платформи керування даними,

CAL – хмарно-незалежні бібліотеки.

A – набір атрибутів і характеристик, що описують класи понять:

$$A = \{C, P, S, ImpE\}, \quad (3.11)$$

де C – вартість,

P – продуктивність,

S – безпека,

$ImpE$ – зусилля щодо впровадження.

V – результуючі значення у вигляді рангу пріоритетів:

$$V = \{V_1, V_2, \dots, V_n\}, \quad (3.12)$$

де V_1 – найвищий пріоритет, а V_n – найнижчий.

I – множина екземплярів класів, представляють набір варіантів мультимарної взаємодії:

$$I = \{S3Proxy, Apache NiFi, Apache Libcloud\} \quad (3.13)$$

R – набір відношень між класами, які вказують на зв'язки або взаємодію між поняттями. Кожен підхід оцінюється на основі його ефективності щодо кожного критерію, пов'язуючи їх із потенційними результатами.

Ця модель дозволяє структурувати вибір і створити прозорий процес оцінки. Більш того, дану модель можна розширити на кожному ієрархічному рівні, де можна додати інші мультимарні стратегії та критерії.

Оцінку альтернативних рішень на основі онтологічної моделі можна розділити на такі етапи:

1. Визначення мети і вибір концепцій під конкретний сценарій.
2. Визначення критеріїв оцінки. В описаній моделі – це продуктивність, безпека, зусилля впровадження і вартість.
3. Для кожної альтернативи проводиться оцінка по вказаним критеріям.
4. Ранжування альтернатив на основі оцінок.
5. Прийняття рішення щодо впровадження конкретного підходу.

Онтологія допомагає врахувати важливі нюанси прийняття рішень, наочно показати процес оцінки та створити гнучкий план вибору інструментів для багатохмарного доступу в конкретній організації.

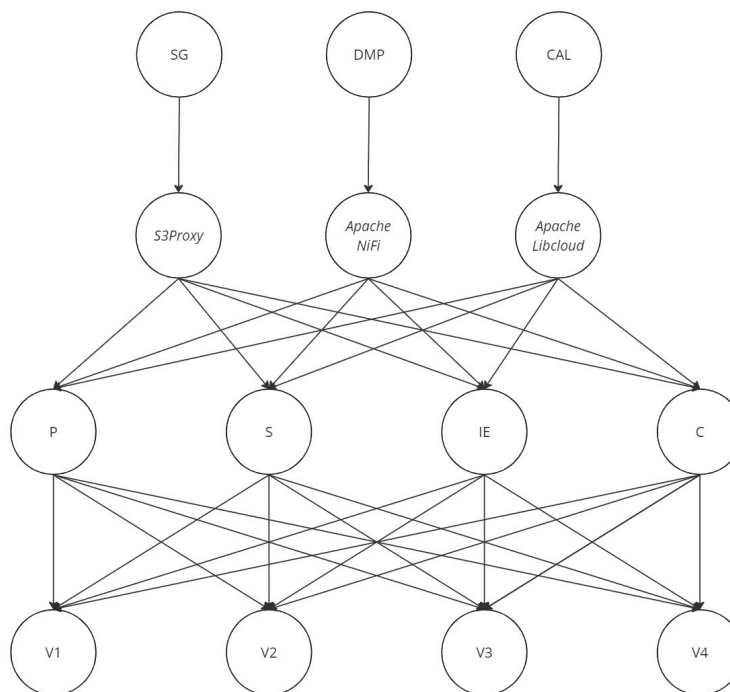


Рисунок 3.1 Онтологічна модель вибору мультихмарного підходу

Таким чином, побудована онтологічна модель є основою побудови сховища зберігання всіх необхідних ресурсів для побудови мультихмарної інфраструктури, та дозволяє формалізувати зв'язки між критеріями та альтернативами, покращуючи прозорість процесу прийняття рішень.

3.5 Моделі машинного навчання та LLM для побудови мультихмарного середовища

Нечітка логіка – це форма обчислень, яка враховує невизначеність і неточність, що робить її придатною для динамічних і неоднорідних середовищ, таких як багатохмарні системи. На відміну від класичної бінарної логіки, нечітка логіка

використовує ступені істинності, зазвичай від 0 до 1. Ця можливість особливо цінна в багатохмарних середовищах, де показники продуктивності, витрати та доступність ресурсів часто коливаються.

Використання в динамічних мультимарних середовищах можна описати наступним чином [78]:

- Визначення критеріїв за допомогою нечітких наборів. Наприклад, зусилля впровадження можна охарактеризувати як «мінімальне», «помірне», «значне».
- Системи нечіткого висновку (Fuzzy Inference Systems (FIS)). Тобто відбувається побудова правил на основі нечітких наборів. Наприклад, якщо критерій А «високий» і критерій Б «помірний», то виберіть Альтернативу №3.
- Динамічна адаптація. Нечітка логіка за своєю суттю є адаптивною та може обробляти зміни в реальному часі, що робить її ідеальною для середовищ, де робоче навантаження зміщується або ресурси стають обмеженими.
- Дефазифікація. Вихідні дані прийняття рішення з нечіткої системи перетворюються на чітке значення або вибір, наприклад вибір конкретного хмарного постачальника або конфігурації.

У той час як нечітка логіка надійна в роботі з невизначеністю, інтеграція машинного навчання (ML) може ще більше покращити процес прийняття рішень завдяки можливостям прогнозування та розпізнаванню шаблонів.

Впровадження машинного навчання в багатохмарному управлінні медіа даними, наприклад, надає переваги [79]:

- Покращена ефективність і продуктивність. Машинне навчання автоматизує трудомісткі завдання, такі як класифікація вмісту та тегування метаданих, що дозволяє медіафахівцям зосередитися на більш стратегічній і творчій діяльності.
- Розширений процес прийняття рішень і стратегічного планування. Машинне навчання дає цінну інформацію про поведінку користувачів, продуктивність вмісту та ринкові тенденції. Аналізуючи величезну кількість даних із різних джерел,

алгоритми машинного навчання можуть виявити закономірності та ідеї, які аналітики можуть не помітити.

В цілому ключовими сферами інтеграції ML можна назвати [80]:

- Прогнозування: передбачення майбутніх потреб в ресурсах або потенційно вузькі місця в хмарних службах. Можливість прогнозувати витрати на основі моделей використання, що дозволяє краще фінансове планування в багатохмарних операціях.
- Оптимізація: політика розподілу ресурсів з часом, взаємодіючи з середовищем і мінімізуючи затримку або вартість.
- Розширення моделей нечіткої логіки: алгоритми машинного навчання, такі як дерева рішень або методи кластеризації, можуть генерувати або уточнювати нечіткі правила на основі історичних даних, зменшуючи ручне втручання.
- Виявлення аномалій: ML можуть ідентифікувати ненормальну поведінку системи, наприклад незвичайні стрибки витрат або погіршення продуктивності, що дозволяє вживати попереджувальні заходи.

3.5.1 Алгоритм роботи LLM

Large Language Model (або LLM) – це сучасний інструмент, що значно розширює можливості роботи з текстом. LLM – це складна ШІ-модель, що працює з текстовими даними. Токенізація є важливим етапом попередньої обробки навчання LLM, яке аналізує текст на нерозкладні одиниці, що називаються токенами. Токени можуть бути символами, підсловами, символами або словами, залежно від процесу токенизації [81].

Модель застосовує багатшарові алгоритми обробки контексту. Це дозволяє розуміти не тільки слова, а й їхній сенс у конкретному запиті. Далі модель отримує доступ до своїх знань – це величезні масиви даних, на яких вона була натренована. І, нарешті, модель формує відповідь, яка є повною та зрозумілою.

Основні етапи роботи моделі представлені на рис.3.2.



Рисунок 3.2 Етапи роботи моделі

Етапи:

1. Токенізація – розділення тексту на окремі елементи, які модель може аналізувати.
2. Аналіз контексту – встановлення зв'язків між токенами для розуміння сенсу тексту.
3. Доступ до знань – витяг релевантної інформації з баз знань або тренувальних даних моделі.
4. Генерація відповіді – формування завершеного тексту на основі отриманих даних.

Математичне представлення роботи LLM можна описати наступним чином:

1. Токенізація. Вхідний текст перетворюється на послідовність токенів:
 $T \rightarrow (t_1, t_2, \dots, t_n)$.

2. Аналіз контексту. На основі токенів модель обчислює контекстні представлення h_i для кожного токена t_i за допомогою багатопарової трансформерної архітектури: $h_i = \text{Transf}(t_1, t_2, \dots, t_n)$.

3. Доступ до знань. Модель використовує ваги W , які зберігають інформацію, отриману під час навчання, для збору релевантної інформації z_i : $z_i = f(h_i, W)$, f – функція доступу до знань.

4. Генерація відповіді. На основі контексту та знань модель формує послідовність вихідних токенів $\{y_1, y_2, \dots, y_m\}$. Фінальний текст R об'єднується з токенів: $R = ".join(y_1, y_2, \dots, y_m)$.

Розглянемо архітектуру Transformer, яка є основою сучасних LLM за джерелом [79].

Більшість моделей трансдукції нейронної послідовності мають структуру кодер-декодер. Кодер відображає вхідну послідовність представлень символів (x_1, x, \dots, x_n) у послідовність неперервних представлень $z = (z_1, z_2, \dots, z_n)$. За умови z декодер генерує вихідні дані послідовність (y_1, \dots, y_m) символів по одному елементу за раз.

Модель Transformer працює за аналогічною схемою. Вона використовує кілька рівнів та повністю з'єднані шари в кодері та декодері (рис.3.3).

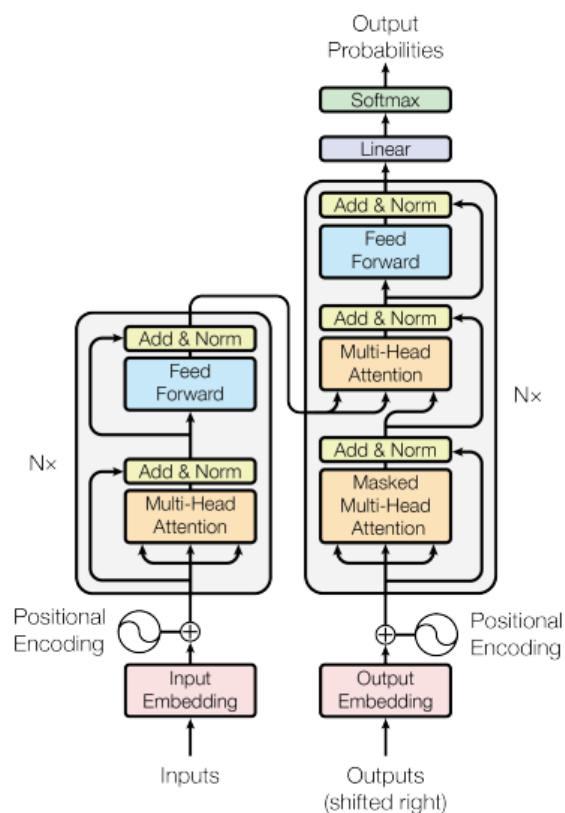


Рисунок 3.3 [82] Transformer – модель архітектури

Кодер складається зі стека з N ідентичних шарів. Кожен шар має два підшари. Перший – це multi-head (багатоголовий) механізм самоконтролю. Він допомагає моделі зосередитися на різних частинах вхідних даних. Другий – мережа прямого зв'язку (feed forward) – обробляє вихідні дані.

Декодер також складається зі стека з N ідентичних шарів. На додаток до двох підрівнів у кожному шарі кодера, декодер вставляє третій підрівень, який дає вказівку multi-head обробити дані із виходу стеку кодера. Багатоголовий механізм уваги дозволяє моделі аналізувати різні аспекти тексту одночасно.

Функцію уваги можна описати як відображення запиту та набору пар ключ-значення на вихідні дані, де запит, ключі, значення та вихідні дані є векторами. Вихід обчислюється як зважена сума значень, де вага, призначена кожному значенню, обчислюється функцією сумісності запиту із відповідним ключем.

Остаточний результат отримується, коли оброблена інформація проходить через лінійний рівень (linear). Функція softmax призначає ймовірності словам, вибираючи найбільш ймовірне наступне слово в послідовності. Зазначена структура дозволяє Transformer обробляти переклади, генерувати текст тощо, не потребуючи повторення.

Загальний алгоритм функціонування запропонованого підходу з використання LLM для аналізу і підтримки прийняття рішень для проєктування мультимарних інфраструктур представлено на рис. 3.4.

Спочатку користувач подає вхідні дані – код проєкту та параметри вимог. Далі перший етапу – екстракції метаданих. Цей етап дозволяє виділити ключову інформацію, яка стане основою для подальшого аналізу.

Система паралельно виконує три важливі блоки аналізу:

- Аналіз залежностей сервісів, щоб зрозуміти, які хмарні сервіси вже використовуються і як вони взаємодіють.
- Регіональні обмеження, які допомагають врахувати географічні вимоги, наприклад, де повинні розташовуватися дані чи сервіси.
- Оцінка потреб у ресурсах, яка дозволяє визначити, які обчислювальні ресурси будуть потрібні для роботи системи.

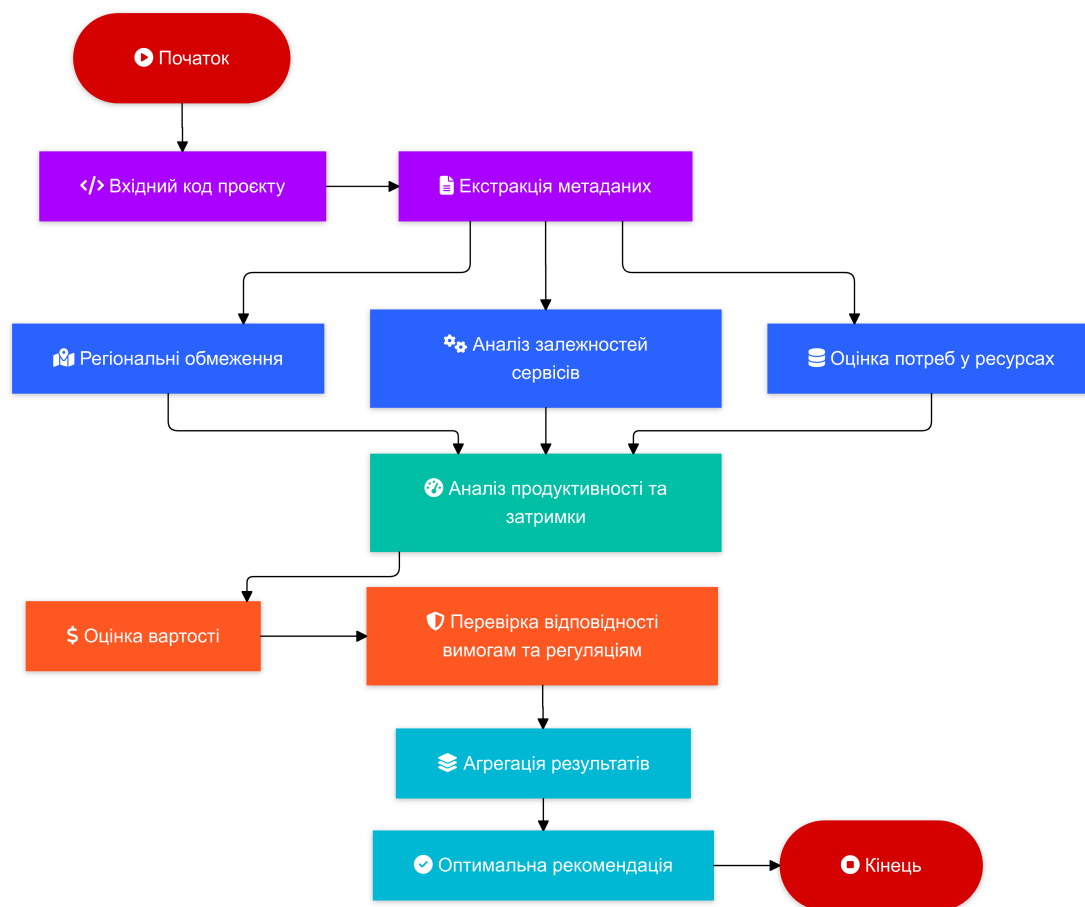


Рисунок 3.4 Етапи роботи алгоритму

Наступний крок – аналіз продуктивності та затримки, який збирає дані з усіх попередніх блоків, щоб оцінити ефективність. На основі цієї інформації виконується оцінка вартості, що включає розрахунок потенційних витрат на хмарні сервіси.

Наступний крок – перевірка відповідності вимогам, що дозволяє впевнитися, що всі обрані сервіси відповідають необхідним стандартам і правилам.

У результаті всі отримані дані проходять через блок агрегації результатів, де вони комбінуються в єдину оцінку. На завершальному етапі ядро прийняття рішень пропонує оптимальну рекомендацію, що дозволяє користувачеві отримати найкраще рішення відповідно до його вимог.

Представлена структура відображає інтеграцію функціональних модулів системи для досягнення єдиної мети – автоматизації та оптимізації процесу вибору хмарних сервісів.

3.5.2 Мультиагентна система на основі LLM

Для вирішення викликів мультихмарної інтеграції пропонується система, яка:

- Адаптується до змін середовища, тобто здатна працювати з динамічними параметрами, оновлюючи критерії та рішення в реальному часі.
- Автоматизує аналіз рішень і таким чином мінімізує ручну участь, спрощуючи прийняття складних рішень.
- Інтегрує багатокритеріальний аналіз: дозволяє враховувати десятки критеріїв та взаємозв'язків.
- Масштабується, тобто здатна працювати з великими обсягами даних та різноманітними компонентами.

Традиційні статичні моделі демонструють обмежені можливості щодо подолання комплексних викликів мультихмарної інтеграції, що зумовлює необхідність запровадження підходу на основі мультиагентної AI-системи (MAS).

У найбільш загальному вигляді мультиагентна інформаційно-аналітична система складається з трьох підсистем: збору та зберігання даних, аналізу даних та взаємодії з користувачем [83].

Структурні компоненти системи характеризуються наступним чином:

1. Підсистема збору та зберігання даних. Пошук та автоматичне завантаження даних, їх обробка та оновлення. Збереження отриманої інформації та підтримка її актуальної структури.
2. Підсистема аналізу даних. Обчислення статистичних даних і створення прогнозів. Набір показників визначається залежно від використовуваних агентів у системі.
3. Підсистема взаємодії з користувачем. Робота з користувачем: перетворення

його запитів у команди, які розуміє система, та наочне представлення результатів аналізу.

Структуру мультиагентних систем формують автономні об'єкти, які класифікуються як агенти. Агенти спільно вирішують завдання, але вони пропонують більшу гнучкість завдяки здатності навчатися та приймати автономні рішення. Агенти використовують взаємодію з сусідніми агентами або з середовищем, щоб вивчати нові контексти, та дії для вирішення призначеного завдання. Щоб розробити MAS, потрібно вирішити різноманітний спектр складних проблем, таких як координація між агентами, навчання і безпека [84].

Мультиагентна AI-система розроблена так, щоб динамічно адаптуватися до змін середовища, автоматизувати складний процес аналізу та враховувати широкий набір критеріїв. Наприклад, коли з'являється новий хмарний сервіс або змінюються параметри безпеки чи продуктивності, система самостійно враховує ці зміни.

Розроблена система представляє собою комплексне адаптивне рішення, що автоматизує підтримку оптимальних конфігурацій для мультихмарних середовищ.

Агент визначається як програмний суб'єкт, інтегрований у середовище, що здійснює аналіз параметрів середовища для формування рішень відповідно до встановлених цільових функцій. На основі цього рішення суб'єкт здійснює необхідні дії щодо навколишнього середовища.

Функціональні характеристики агентів, що забезпечують їх універсальність та ефективність при вирішенні складних задач, включають [85]:

- Комунікабельність. Агенти можуть ділитися своїми знаннями, а також запитувати інформацію в інших агентів, щоб покращити продуктивність у досягненні своїх цілей.

- Автономність. Кожен агент може самостійно виконувати процес прийняття рішень і вживати відповідних дій.

- Проактивність. Кожен агент використовує свою історію, визначені

параметри та інформацію інших агентів, щоб передбачити можливі майбутні дії. Ці передбачення дозволяють агентам вживати ефективних дій, які відповідають їхнім цілям. Ця здатність означає, що один і той самий агент може виконувати різні дії, коли він знаходиться в різних середовищах.

ВИСНОВКИ

1. Проведений аналіз підходів до проєктування обчислювальної інфраструктури у мультихмарному середовищі демонструє можливості оптимізації продуктивності, вартості і надійності шляхом ефективного використання сервісів різних хмарних постачальників. Різноманітність робочих навантажень у мультихмарному середовищі потребує адаптивних методів керування ресурсами, особливо для обробки завдань у реальному часі. Архітектура систем динамічного розподілу обчислень складається з модульних компонентів, які забезпечують розподіл обчислювальних завдань, автоматизоване перерозгортання мультихмарної інфраструктури, її моніторинг і прийняття рішень щодо її переналаштування.

2. Розглянуто модель багатокритеріального аналізу (MCDA) з імплементацією методу аналітичної ієрархії АНР, який дозволяє оцінити альтернативи застосованого інструментарію для побудови мультихмарної інфраструктури за критеріями продуктивності, вартості, безпеки та зусиль впровадження.

3. Розроблено онтологічну модель, що функціонує як основа систематизації ресурсів для побудови мультихмарної інфраструктури та реалізує формалізацію кореляційних залежностей між оціночними критеріями та варіантами рішень, що сприяє підвищенню логічної обґрунтованості процесу прийняття рішень.

4. Проаналізовано використання нечіткої логіки та машинного навчання. Ці методи можуть покращити процес ухвалення рішень у гетерогенних середовищах, забезпечуючи адаптивність до змінних параметрів і прогнозування навантажень.

5. Обґрунтовано доцільність імплементації мультиагентної системи на основі великих мовних моделей (LLM), що забезпечує автоматизацію процесів переналаштування мультихмарної інфраструктури, та здатна враховувати велику кількість критеріїв і змін параметрів середовища у режимі реального часу.

РОЗДІЛ 4

КОМПЛЕКСНИЙ МЕТОД ДИНАМІЧНОГО ФОРМУВАННЯ ОБЧИСЛЮВАЛЬНОЇ ІНФРАСТРУКТУРИ У МУЛЬТИХМАРНОМУ СЕРЕДОВИЩІ

4.1 Гібридна структура проєктування мультимарної інфраструктури

Розв'язання задачі проєктування ефективної обчислювальної інфраструктури у гетерогенному мультимарному середовищі ґрунтується на інтеграції декількох методологічних підходів, що поєднує адаптивність, інтелектуальність та формальну обґрунтованість архітектурних рішень. У даному підрозділі представлено гібридну структуру інтелектуальної системи, що базується на інтеграції методів машинного навчання з підкріпленням (Reinforcement Learning) та багатокритеріальних еволюційних алгоритмів (Multi-Objective Evolutionary Algorithms) для оптимізації процесів розподілу обчислень у динамічному гетерогенному середовищі.

4.1.1 Математична модель інтегрованого показника ефективності виконання обчислювальних завдань

Зростаюча складність і масштаб сучасних хмарних інфраструктур зумовили необхідність розробки інтелектуальних автоматизованих систем для проєктування та керування мультимарними середовищами [51]. При цьому автоматизація має керуватися наступними цілями:

- Динамічність. Система повинна підлаштовувати інфраструктуру в режимі реального часу відповідно до змін робочого навантаження, вимог користувачів або умов постачальника.
- Масштабованість. Плавне масштабування ресурсів між провайдерами, коли потреба в системі коливається.

- Адаптивність. Система має адаптуватися до нових вимог програми, змін SLA або нових можливостей постачальника.

Для досягнення вищезазначених цілей необхідно визначити вимоги, які можна поділити на функціональні та нефункціональні.

Функціональні:

- Гнучке резервування та скасування резервування хмарних ресурсів.
- Взаємодія між різнорідними платформами.
- Моніторинг і оркестровка в реальному часі.

Нефункціональні:

- Безперервність обслуговування. Інтеграція механізмів високої доступності та відмовостійкості.

- Відповідність SLA. Система повинна проактивно керувати інфраструктурою, щоб відповідати угодам про рівень обслуговування.

- Економічність. Алгоритми оптимізації мають збалансовувати продуктивність і бюджетні обмеження.

Основною проблемою багатохмарного дизайну є неоднорідність, яка зумовлена відмінностями в API, моделях обслуговування, структурах ціноутворення та характеристиках продуктивності між провайдерами.

Методи онтології хмарних сервісів і механізми зіставлення сервісів відіграють ключову роль у тому, щоб зробити роботу з хмарними технологіями більш простою та зрозумілою [86, 87]. Автоматизоване прийняття рішень має включати прозорість (розуміння сторонами як і чому приймаються рішення), аргументованість (грунтування на чітко визначеній логіці чи моделях), відтворюваність (прийняття системою послідовних рішень).

Інтегрований показник ефективності виконання обчислювальних завдань у мультихмарному середовищі формалізовано як динамічну функцію від стану системи та часу:

$$IE(S, t) = w_1 \cdot Cost_{norm}(S, t) + w_2 \cdot Perf_{norm}(S, t) + w_3 \cdot Rel_{norm}(S, t) + w_4 \cdot Inter_{norm}(S, t) + w_5 \cdot Sec_{norm}(S, t), \quad (4.1)$$

де:

$IE(S, t)$ – інтегрований показник ефективності для стану системи S у момент часу t ;

$Cost_{norm}(S, t)$ – нормалізована оцінка вартості виконання завдань у стані S у момент t ;

$Perf_{norm}(S, t)$ – нормалізована оцінка продуктивності системи у стані S ;

$Rel_{norm}(S, t)$ – нормалізована оцінка надійності системи у стані S ;

$Inter_{norm}(S, t)$ – нормалізована оцінка сумісності взаємодії компонентів системи;

$Sec_{norm}(S, t)$ – нормалізована оцінка безпеки системи;

w_1, w_2, w_3, w_4, w_5 – динамічні вагові коефіцієнти, що адаптуються з часом відповідно до зміни пріоритетів та умов функціонування системи.

Динаміка зміни стану інфраструктури мультимарного середовища може бути представлена як марковський процес прийняття рішень:

$$S(t + \Delta t) = F(S(t), A(t), E(t)), \quad (4.2)$$

де:

$S(t)$, – стан системи в момент часу t ;

$A(t)$ – дія (рішення), вибрана системою в момент t ;

$E(t)$ – зовнішні події та зміни середовища;

$F()$ – функція переходу системи в новий стан.

4.1.2 Формалізація простору станів та дій для проєктування мультимарної інфраструктури

Для формалізації процесу проєктування обчислювальної інфраструктури у гетерогенному мультимарному середовищі необхідно визначити простір можливих станів системи та простір доступних дій архітектурних рішень. Це дозволяє застосувати методи підкріплювального навчання для автоматизованого прийняття оптимальних рішень.

Простір станів системи S можна представити як багатовимірний вектор, що характеризує поточну архітектуру та стан мультимарної інфраструктури:

$$S = \{CP, WL, NP, QoS, CO\}, \quad (4.3)$$

де:

$CP = \{cp_1, cp_2, \dots, cp_m\}$ – множина доступних хмарних провайдерів з їх характеристиками;

$WL = \{wl_1, wl_2, \dots, wl_n\}$ – множина обчислювальних завдань з їх вимогами та пріоритетами;

$NP = [np_{ij}]$ – матриця поточного розміщення завдань на сервісах, де $np_{ij} = 1$, якщо завдання i призначено провайдеру j , інакше $np_{ij} = 0$;

$QoS = \{qos_1, qos_2, \dots, qos_k\}$ – множина метрик якості обслуговування;

$CO = \{co_1, co_2, \dots, co_p\}$ – множина обмежень та правил проєктування.

Простір дій A включає всі можливі операції, які система може виконати для зміни поточної архітектури мультимарної інфраструктури:

$$A = \{Assign, Migrate, Scale, Optimize\}, \quad (4.4)$$

де:

$Assign(wl, cv)$ – призначення нового завдання wl сервісу cv ;

$Migrate(wl, cv_1, cv_2)$ – міграція існуючого завдання wl від сервісу cv_1 до cv_2 ;

$Scale(wl, r)$ – масштабування ресурсів r , виділених для завдання wl ;

$Optimize(cv, param)$ – оптимізація параметрів $param$ для сервісу a cv .

Функція переходу $F(s, a)$ визначає ймовірність переходу системи зі стану s до стану s' при виконанні дії a . У детермінованому середовищі ця функція однозначно визначає новий стан:

$$s' = F(s, a) \quad (4.5)$$

У стохастичному середовищі функція переходу визначає розподіл імовірностей:

$$P(s'|s, a) = Prob[S(t+1) = s' | S(t) = s, A(t) = a] \quad (4.6)$$

4.1.3 Функції винагороди на основі багатокритеріальної оптимізації

Для оцінки ефективності архітектурних рішень та адаптації параметрів інтелектуальної підсистеми прийняття рішень оптимальним стратегіям проєктування мультихмарної інфраструктури необхідно визначити функцію винагороди, яка враховуватиме всі релевантні критерії оптимізації.

Композитна функція винагороди формується як зважена сума окремих компонентів винагороди, що відповідають різним критеріям оптимізації:

$$R(s, a, s') = w_1 \cdot R_{cost}(s, a, s') + w_2 \cdot R_{perf}(s, a, s') + w_3 \cdot R_{rel}(s, a, s') + w_4 \cdot R_{inter}(s, a, s') + w_5 \cdot R_{sec}(s, a, s'), \quad (4.7)$$

де:

$R(s, a, s')$ – загальна функція винагороди при переході зі стану s до стану s' внаслідок виконання дії a ;

$R_{cost}(s, a, s')$ – компонент винагороди за економічною ефективністю;

$R_{perf}(s, a, s')$ – компонент винагороди за продуктивністю;

$R_{rel}(s, a, s')$ – компонент винагороди за надійністю;

$R_{inter}(s, a, s')$ – компонент винагороди за сумісністю взаємодії;

$R_{sec}(s, a, s')$ – компонент винагороди за безпекою;

w_1, w_2, w_3, w_4, w_5 – вагові коефіцієнти, що визначають пріоритет кожного критерію.

Компоненти функції винагороди формулюються наступним чином:

1. Винагорода за економічною ефективністю:

$$R_{cost}(s, a, s') = -N \cdot (C(s') - C(s))/C_{max}, \quad (4.8)$$

де $C(s)$ – вартість конфігурації у стані s , C_{max} – максимально допустима вартість, N – нормалізуючий коефіцієнт.

2. Винагорода за продуктивністю:

$$R_{perf}(s, a, s') = N \cdot (P(s') - P(s))/P_{max}, \quad (4.9)$$

де $P(s)$ – продуктивність системи у стані s , P_{max} – максимально можлива продуктивність.

3. Винагорода за надійністю:

$$R_{rel}(s, a, s') = N \cdot (R(s') - R(s))/R_{max}, \quad (4.10)$$

де $R(s)$ – надійність системи у стані s , R_{max} – максимально можлива надійність.

4. Винагорода за сумісністю взаємодії:

$$R_{inter}(s, a, s') = N \cdot (I(s') - I(s))/I_{max}, \quad (4.11)$$

де $I(s)$ – рівень сумісності взаємодії у стані s , I_{max} – максимально можливий рівень сумісності.

5. Винагорода за безпекою:

$$R_{sec}(s, a, s') = N \cdot (Sec(s') - Sec(s))/Sec_{max}, \quad (4.12)$$

де $Sec(s)$ – рівень безпеки системи у стані s , Sec_{max} (максимально можливий рівень безпеки).

Ця багатокритеріальна функція винагороди дозволяє системі оптимізувати розподіл обчислювальних завдань з урахуванням компромісів між різними, часто конфліктуючими цілями.

4.1.4 Механізми адаптації та навчання

Для забезпечення динамічної адаптації до змін у середовищі та постійного вдосконалення процесу прийняття рішень, система інтегрує наступні механізми адаптації та навчання:

1. Повторне використання досвіду (Experience Replay). Механізм повторення досвіду зберігає переходи (s, a, r, s') у буфері D та періодично вибирає міні-пакети для оновлення функції цінності Q :

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a)], \quad (4.13)$$

де α – швидкість навчання, γ – коефіцієнт дисконтування.

2. Пріоритетне повторення досвіду (Prioritized Experience Replay). Система приділяє більше уваги важливим переходам, використовуючи пріоритетний вибір з імовірністю:

$$P(i) = p_i^\alpha / \sum_j p_j^\alpha, \quad (4.14)$$

де p_i – пріоритет переходу i , α – коефіцієнт, що визначає ступінь пріоритизації.

3. Трансферне навчання (Transfer Learning). Для прискорення навчання у нових середовищах система використовує знання, отримані з попереднього досвіду:

$$Q_{new}(s, a) = (1 - \tau) \cdot Q_{random}(s, a) + \tau \cdot Q_{old}(s, a), \quad (4.15)$$

де $\tau \in [0, 1]$ контролює ступінь передачі знань.

4. Мета-навчання (Meta-Learning). Система оптимізує сам процес навчання:

$$\theta^* = \operatorname{argmin}_{\theta} \sum_T L_T(f\theta), \quad (4.16)$$

де θ – параметри моделі, T – набір завдань, $L_T()$ – функція втрат для завдання T , $f\theta$ – функція прийняття рішень з параметрами θ .

5. Динамічна адаптація вагових коефіцієнтів. Система динамічно коригує вагові коефіцієнти w_i у функції винагороди на основі ефективності та зворотного зв'язку:

$$w_i(t+1) = w_i(t) + \alpha \cdot \nabla E(w(t)) / \nabla w_i + \beta \cdot \eta(t), \quad (4.17)$$

де α – швидкість навчання, $\nabla E(w(t))$ – градієнт функції ефективності за ваговим коефіцієнтом w_i , β – коефіцієнт стохастичної варіації, $\eta(t)$ – випадковий шум.

З подальшою нормалізацією:

$$w_i(t+1) = w_i(t) / \sum_j w_j(t+1). \quad (4.18)$$

Ці механізми адаптації та навчання забезпечують постійне вдосконалення стратегій розподілу обчислювальних завдань у мультимарному середовищі, дозволяючи системі ефективно реагувати на динамічні зміни у потребах та характеристиках середовища.

4.2 Комплексний метод формування обчислювальної інфраструктури

Комплексний метод формування обчислювальної інфраструктури у гетерогенному мультимарному середовищі об'єднує теоретичні моделі з практичними підходами до побудови ефективної системи підтримки прийняття рішень. У цьому підрозділі представлено архітектуру мультиагентної системи, алгоритми оптимізації розподілу обчислювальних завдань та методи динамічної адаптації до зміни параметрів середовища.

4.2.1 Архітектура мультиагентної системи підтримки прийняття рішень

Для реалізації автоматизованої системи підтримки прийняття рішень у мультихмарному середовищі запропоновано наступну теоретичну архітектуру мультиагентної системи на основі LLM (рис.4.1).



Рисунок 4.1 Архітектура мультиагентної AI-системи

Архітектура складається з наступних компонентів:

1. Інтерфейс користувача – місце, де користувач вводить свої вимоги: продуктивність, бюджет, безпеку тощо. Це точка входу в систему.

2. Обробка вхідних даних – система нормалізує введені дані, щоб вони були готові до аналізу.

3. Мультиагентна AI-система – основний блок аналізу, що складається з [11]:

- Агента розрахунку продуктивності, який аналізує швидкість і стабільність.
- Агента оптимізації вартості, що шукає економічно вигідні рішення.
- Агента розрахунку безпеки, який оцінює ризики та захищеність.
- Агента розрахунку складності впровадження, що визначає трудомісткість інтеграції.

Координацію між агентами забезпечує агент-координатор, який синхронізує інформаційні потоки та передає зведену інформацію до модуля прийняття рішень.

4. Ядро прийняття рішень – центральний механізм, що приймає рішення на

основі результатів роботи агентів, із застосуванням методів MCDA та моделей ІІІ. Передбачено використання LLM для генерації обґрунтованих рішень, пояснення альтернатив, узагальнення великих обсягів даних [88].

Мультихмарне середовище передбачає роботу з різноманітними хмарними провайдерами, кожен з яких має власні API, механізми автентифікації, формати опису сервісів, параметри конфігурації та обмеження. Для усунення залежності від конкретного постачальника та забезпечення інтеоперабельності, у системі реалізовано шар абстракції, що виконується у вигляді адаптерів до API.

Реалізація цього шару має включати абстрактні інтерфейси ресурсів (наприклад, VirtualMachine), адаптери для кожного типу хмарного середовища (AWS, Azure, GCP, OpenStack тощо), які транслюють виклики на відповідні API, реєстр сервісів з метаданими (назви, регіони, типи ресурсів, підтримувані SLA). Цей підхід дозволяє агентам системи виконувати розрахунки, базуючись на єдиній моделі сервісів, а не на специфіках конкретного провайдера.

Для підтримки реактивності та адаптивності системи в реальному часі включено модуль динамічного моніторингу, який виконує збір та обробку телеметричних даних з різних джерел. Його функції:

- Збір показників продуктивності: затримка, навантаження, обсяг використаних ресурсів.
- Виявлення подій безпеки: несанкціонований доступ, невідповідності політикам.
- Відстеження фінансових витрат у режимі реального часу.

Зібрані дані накопичуються у сховищі телеметрії й доступні як для аналізу агентами, так і для LLM/AI-модуля. Система автоматично ініціює сценарії рекомендацій у випадках, коли ключові показники відхиляються від очікуваних значень.

Одним із ключових інноваційних елементів проєктованої архітектури є інтеграція LLM для підтримки складного аналізу, формування обґрунтованих рішень та автоматичного створення документації. Можливості такого модуля:

- Генерація обґрунтувань для рішень з поясненням вибору конфігурацій.
- Узагальнення даних з телеметрії, логів і бази знань для формування аналітичних висновків.
- Пропозиції альтернатив або сценаріїв оптимізації на основі історичних рішень і контексту.

Запропонована архітектура автоматизує процес вибору хмарних конфігурацій із врахуванням усіх ключових критеріїв ефективності. Архітектурна модель, представлена на діаграмі (рис.4.2), відображає структуру основних підсистем, включаючи мультиагентну логіку, моніторинг, адаптери та LLM/AI-компоненти.

Функції агентів визначено таким чином:

1. Технічний агент. Відповідає за технічну оцінку варіантів хмарного розгортання на основі продуктивності, сумісності і масштабованості.
2. Економічний агент. Виконує аналіз вартості рішень, формує прогнози витрат та генерує пояснення за допомогою LLM-модулів. Модель використовує LLM для аналізу складних тарифів і генерації пояснень.

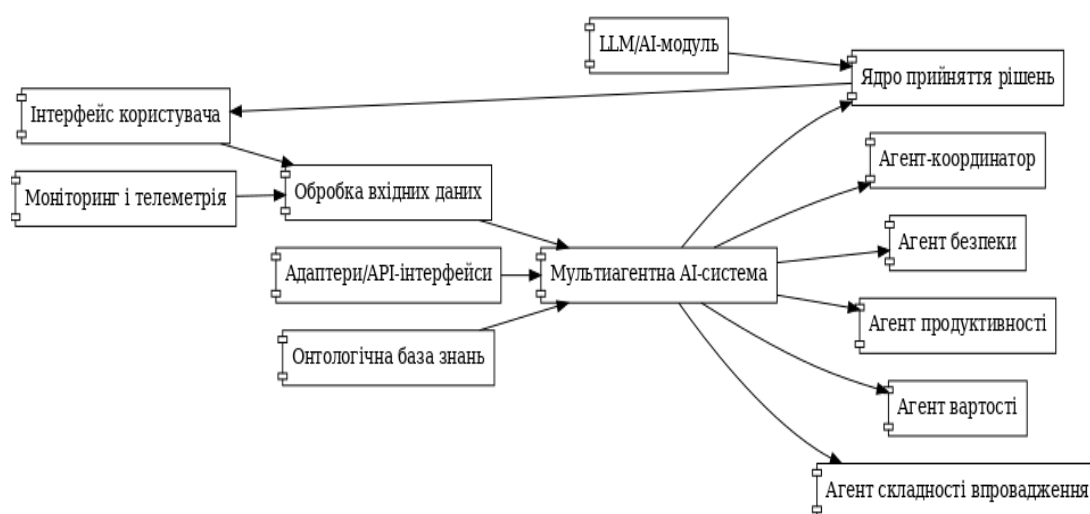


Рисунок 4.2 Діаграма компонентів системи проєктування мультимарного середовища з використання LLM.

3. Безпековий агент. Перевіряє безпекові критерії, такі як відповідність політикам безпеки, оцінка ризиків, геолокаційні обмеження. Оцінка формується на основі онтологічної бази знань та нормативних шаблонів.

Механізм оцінювання альтернатив можна описати наступним чином. Локальні оцінки кожного агента нормалізуються у діапазоні $[0; 1]$ та ґрунтуються на функціях корисності, адаптованих до відповідного критерію. Оцінка формується за допомогою нормалізації, нечітких правил або евристики. Використовуються функції корисності, адаптовані до типу критерію (максимізація продуктивності, мінімізація вартості тощо).

Агенти мультиагентної системи спеціалізуються за наступними функціональними напрямками:

1. Технічний агент (*TA*). Оцінює технічні параметри варіантів розгортання:

$$TA_{norm}(v) = w_{t_1} \cdot perf(v) + w_{t_2} \cdot scalab(v) + w_{t_3} \cdot compat(v), \quad (4.19)$$

де $perf(v)$, $scalab(v)$, $compat(v)$ – нормалізовані оцінки продуктивності, масштабованості та сумісності відповідно, $TA_{norm}(v)$ – нормалізована оцінка технічного агента для варіанту v , w_{t_1} , w_{t_2} , w_{t_3} – відповідні вагові коефіцієнти.

2. Економічний агент (*EA*). Аналізує економічні аспекти варіантів:

$$EA_{norm}(v) = (Cost_{max} - Cost(v)) / (Cost_{max} - Cost_{min}), \quad (4.20)$$

де $EA_{norm}(v)$ – нормалізована оцінка економічного агента для варіанту v , $Cost_{min}$ – мінімальна теоретично можлива вартість, $Cost_{max}$ – максимальна допустима вартість (бюджетне обмеження), $Cost$ – загальна вартість варіанту v (USD/місяць).

3. Безпековий агент (SA). Оцінює аспекти безпеки та відповідності регуляторним вимогам:

$$SA_{norm}(v) = w_{s_1} \cdot data_sec(v) + w_{s_2} \cdot compl(v) + w_{s_3} \cdot resil(v), \quad (4.21)$$

де $SA_{norm}(v)$ – нормалізована оцінка безпекового агента для варіанту v , $data_sec(v)$, $compl(v)$, $resil(v)$ – нормалізовані оцінки рівня захисту даних, відповідності регуляторним вимогам та стійкості до атак, w_{s_1} , w_{s_2} , w_{s_3} – відповідні вагові коефіцієнти.

Загальна оцінка варіанту v формується як зважена сума оцінок агентів:

$$AE(v) = w_{TA} \cdot TA_{norm}(v) + w_{EA} \cdot EA_{norm}(v) + w_{SA} \cdot SA_{norm}(v), \quad (4.22)$$

де w_{TA} , w_{EA} , w_{SA} – вагові коефіцієнти для технічного, економічного та безпекового агентів відповідно, $AE(v)$ – загальна оцінка агентів.

Контекст (вимоги користувача, попередні оцінки тощо) передається між агентами за допомогою спільної онтологічної моделі та робочого контексту, який зберігається у кеш-пам'яті системи. Наведемо такі приклади:

- Технічний агент може попередити економічного про очікуване навантаження.

- Безпековий агент врахує рекомендації щодо геолокацій від інших агентів.

Це дозволяє уникнути ізольованого аналізу та досягти узгодженого, контекстуального оцінювання.

Архітектура мультиагентної системи підтримки прийняття рішень для формування обчислювальної інфраструктури формалізується у вигляді орієнтованого графа:

$$G = (A, I, M), \quad (4.24)$$

де:

$A = \{a_1, a_2, \dots, a_n\}$ – множина агентів системи;

$I = \{i_1, i_2, \dots, i_m\}$ – множина інформаційних потоків між агентами;

$M = \{m_1, m_2, \dots, m_k\}$ – множина модулів зовнішньої взаємодії.

Кожен агент $a_i \in A$ характеризується трійкою:

$$a_i = (Perc_i, Dec_i, Act_i), \quad (4.25)$$

де:

$Perc_i$ – функція сприйняття, що перетворює входні дані у внутрішнє представлення;

Dec_i – функція прийняття рішень на основі внутрішнього стану агента та політики;

Act_i – функція дії, що реалізує прийняті рішення.

Механізм взаємодії агентів в системі здійснюється через обмін повідомленнями:

$$M_{ij} = (sender_i, receiver_j, content, metadata), \quad (4.26)$$

де:

$sender_i$ – агент-відправник повідомлення;

$receiver_j$ – агент-отримувач повідомлення;

$content$ – зміст повідомлення;

$metadata$ – метадані повідомлення (пріоритет, часові мітки тощо).

4.2.2 Алгоритм оптимізації архітектури мультимарної інфраструктури

На основі формалізованої моделі розроблено алгоритм оптимізації розподілу обчислень, що поєднує переваги методів підкріплювального навчання та багатокритеріальних еволюційних алгоритмів для пошуку оптимальних рішень у динамічному гетерогенному мультимарному середовищі.

Алгоритм 1. Оптимізація архітектури мультимарної інфраструктури.

Вхід: множина обчислювальних завдань T , множина хмарних сервісів S , множина критеріїв C , множина обмежень CO .

Вихід: оптимальна архітектура мультихмарної інфраструктури $X = [x_{ij}]$, де $x_{ij} = 1$, якщо завдання i призначено сервісу j , інакше $x_{ij} = 0$.

1. Ініціалізувати Q -функцію $Q(s, a)$ та функцію політики $\pi(a|s)$.
2. Ініціалізувати популяцію рішень $Pop = \{X_1, X_2, \dots, X_N\}$.
3. Ініціалізувати буфер досвіду D .
4. Для кожної епохи $e = 1, 2, \dots, E$ виконати (починається етап підкріплювального навчання):
 5. Для кожного кроку t у епосі виконати:
 6. спостерігати поточний стан s_t ;
 7. вибрати дію a_t з використанням політики $\pi(a|s_t)$
 8. виконати дію a_t , отримати винагороду r_t та новий стан $s_{\{t+1\}}$;
 9. зберегти перехід $(s_t, a_t, r_t, s_{\{t+1\}})$ у буфері D ;
 10. вибрати міні-пакет переходів з D ;
 11. оновити Q -функцію та політику π на основі міні-пакету;
 12. кінець для етапу підкріплювального навчання.
 13. Етап еволюційної оптимізації:
 14. оцінити кожне рішення $X \in Pop$ за критеріями C ;
 15. визначити фронт Парето недовінованих рішень PF ;
 16. вибрати батьківські рішення з PF згідно з оператором відбору;
 17. застосувати оператори схрещування та мутації для створення нових рішень;
 18. оновити популяцію Pop , зберігаючи найкращі рішення;
 19. оновити параметри мультиагентної системи на основі найкращих рішень;
 20. кінець для етапу еволюційної оптимізації.

21. Повернути найкраще рішення X^* з фронту Парето.

Ключові компоненти алгоритму:

1. Інтеграція *RL* та *MOEA*. Розроблений алгоритм базується на комбінації методів підкріплювального навчання (кроки 6-11) та багатокритеріальної еволюційної оптимізації (кроки 14-18) для досягнення Парето-оптимальних рішень.

2. Оцінка рішень. Оцінювання кожного рішення X здійснюється за вектором критеріїв ефективності, що відповідає специфікації моделі:

$$Eval(X) = [f_1(X), f_2(X), \dots, f_k(X)], \quad (4.27)$$

де $f_k(X)$ – оцінка рішення X за i -м критерієм.

3. Визначення Парето-оптимальності. Рішення X_1 домінує над рішенням X_2 (позначається як $X_1 < X_2$), якщо:

$\forall i \in \{1, 2, \dots, k\}: f_i(X_1) \leq f_i(X_2)$ для задач мінімізації (\geq для максимізації);

$\exists j \in \{1, 2, \dots, k\}: f_j(X_1) < f_j(X_2)$ для задач мінімізації ($>$ для максимізації).

Фронт Парето PF містить всі недоміновані рішення.

4. Операції еволюційного алгоритму:

- Відбір. Вибір батьківських рішень з пріоритетом до недомінованих рішень та з урахуванням різноманіття.

- Схрещування. Генерація нових рішень шляхом комбінування батьківських рішень.

- Мутація. Внесення випадкових змін у рішення для дослідження нових областей простору пошуку.

5. Адаптація параметрів. На основі найкращих рішень з популяції система адаптує параметри мультиагентної системи, щоб покращити якість майбутніх рішень.

4.2.3 Модель динамічної адаптації до зміни параметрів середовища

Для забезпечення адаптивності до динамічних змін у мультимарному середовищі розроблено модель динамічної адаптації, що базується на принципах зворотного зв'язку та прогнозування змін параметрів системи.

Модель динамічної адаптації складається з наступних компонентів:

1. Моніторинг стану системи. Стан системи постійно відстежується через збір телеметрії та метрик:

$$M(t) = \{m_1(t), m_2(t), \dots, m_k(t)\}, \quad (4.28)$$

де $m_i(t)$ – значення i -ї метрики в момент часу t .

2. Виявлення відхилень. Система аналізує відхилення поточних значень метрик від очікуваних (або базових):

$$D_i(t) = |m_i(t) - m_i^{e(t)}| / m_i^{\sigma}, \quad (4.29)$$

де $m_i^{e(t)}$ – очікуване значення метрики, m_i^{σ} – стандартне відхилення метрики.

Значне відхилення визначається умовою:

$$D_i(t) > \theta_i, \quad (4.30)$$

де θ_i – пороговий рівень для i -ї метрики.

3. Прогнозування змін. Для проактивної адаптації система прогнозує майбутні значення метрик:

$$m_i^{pred(t + \Delta t)} = f_{pred}(m_i(t), m_i(t - 1), \dots, m_i(t - n)), \quad (4.31)$$

де f_{pred} – функція прогнозування, яка може бути реалізована методами часових рядів, авторегресії або машинного навчання.

4. Вибір стратегії адаптації. На основі виявлених відхилень та прогнозів система вибирає одну з наступних стратегій адаптації:

- Масштабування ресурсів. Збільшення або зменшення кількості ресурсів, виділених для завдання.
- Міграція завдань. Переміщення завдань між хмарними провайдерами.
- Зміна параметрів якості. Зміна рівнів *QoS* для забезпечення балансу між продуктивністю та вартістю.
- Реконфігурація системи. Зміна архітектури розподіленої системи.

5. Реалізація адаптації. Вибрана стратегія адаптації реалізується через механізм управління ресурсами:

$$A(t) = \psi(D(t), M^{pred}(t + \Delta t), S(t), P), \quad (4.32)$$

де $A(t)$ – дія адаптації, ψ – функція вибору дії, $D(t)$ – вектор відхилень, M^{pred} – вектор прогнозів, $S(t)$ – поточний стан, P – політики та обмеження.

6. Оцінка ефективності адаптації. Після виконання адаптивних дій система оцінює їх ефективність:

$$E(A) = \eta(S(t+\Delta t) | S(t), A(t)), \quad (4.33)$$

де η – функція оцінки ефективності, $S(t+\Delta t)$ – стан системи після адаптації.

Результати оцінки ефективності використовуються для коригування параметрів адаптаційної моделі:

$$P(t + 1) = P(t) + \alpha \cdot (E_{target} - E(A)), \quad (4.34)$$

де P – параметри моделі, α – швидкість навчання, E_{target} – цільовий рівень ефективності.

Модель динамічної адаптації функціонує в циклічному режимі, постійно виконуючи моніторинг стану системи та реагуючи на зміни з метою забезпечення оптимальної ефективності розподілу обчислювальних завдань у мультихмарному середовищі.

4.3 Побудова агентів на основі LLM для підтримки прийняття рішень

У даному підрозділі представлено підхід до інтеграції LLM-агентів у мультиагентну систему підтримки прийняття рішень для динамічного формування обчислювальної інфраструктури у мультихмарному середовищі.

Мультиагентна система забезпечує паралельну, незалежну та спеціалізовану оцінку кожного критерію. Завдяки цій модульності можлива адаптація системи до нових вимог або змін у пріоритетах, без потреби перебудови всієї логіки прийняття рішень.

Структура системи може складатися:

- Агенти оцінювання за критеріями (безпеки, продуктивності, вартості, зусиль впровадження).
- Агенти пріоритетів (формування вагових коефіцієнтів на основі введених користувачем, контексту задачі тощо).
- Агент узагальнення (на базі LLM). Він формує пояснення, узагальнює історичні ситуації і поточні або схожі кейси, генерує альтернативи, яких не було в баз знань.
- Ядро прийняття рішень. Застосовує методи MCDA (АНР, TOPSIS або їх гібриди) для узагальнення даних від агентів, формування фінального рішення та передання його LLM для генерації обґрунтування.

Розглянемо приклад архітектури агента оптимізації вартості (рис.4.3). Вхідними даними є вимоги користувача та програмний код обчислювальних задач.

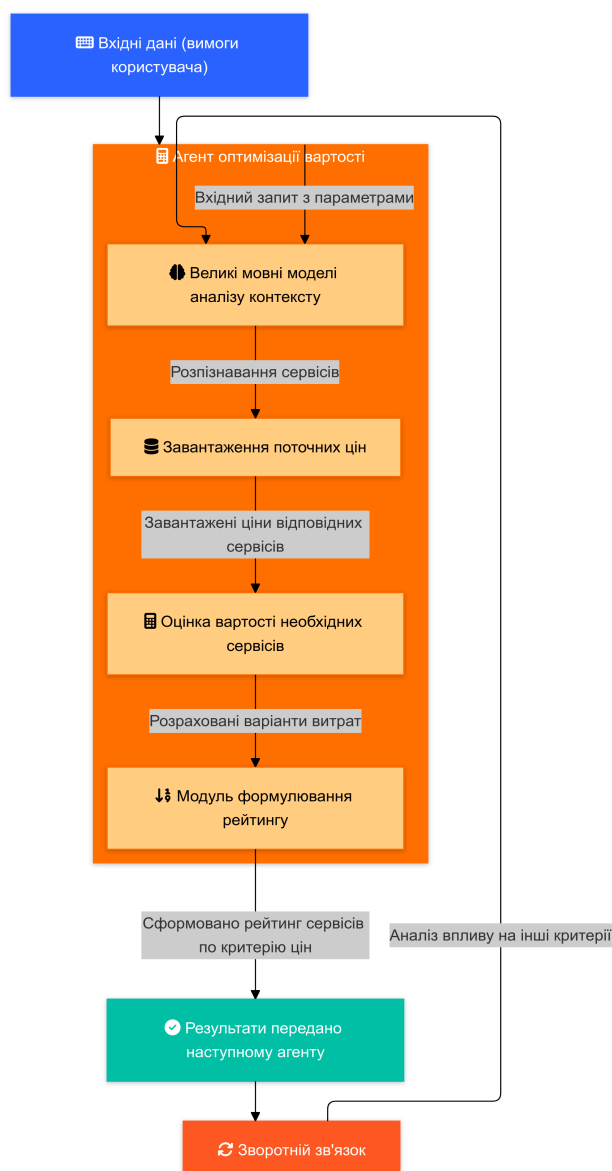


Рисунок 4.3 Архітектура агента оптимізації вартості

Використовуючи LLM, агент розпізнає сервіси, які потрібні для виконання запиту. Далі модуль завантаження поточних цін отримує актуальну інформацію про ціни на ці сервіси. Далі модуль оцінки вартості розраховує варіанти витрат для кожного сервісу. Наприкінці формується рейтинг сервісів за критерієм вартості, і результати передаються далі, щоб допомогти прийняти оптимальне рішення. Це дозволяє автоматизувати процес і враховувати всі важливі фінансові аспекти.

Агент оптимізації вартості виконує декілька ключових етапів, використовуючи сучасні технології, включаючи LLM. На основі вхідних даних модель розпізнає сервіси, які будуть задіяні. Далі ці сервіси порівнюються за поточними цінами, і для кожного з них розраховуються варіанти витрат. На завершальному етапі формується рейтинг сервісів на основі оцінки їхньої вартості, після чого результати передаються для інтеграції з іншими модулями системи.

4.3.1 Процес взаємодії LLM-агентів у мультихмарному середовищі

Інтеграція LLM-агентів у мультиагентну систему вимагає формалізації процесів взаємодії різних типів агентів та координації їх роботи для досягнення спільної мети – оптимального проектування архітектури мультихмарної інфраструктури.

Гібридна система прийняття рішень включає три рівні обробки: формальні правила (чіткі логіки, політики, обмеження), емпіричні дані (телеметрія, історичні спостереження, витрати) та LLM-знання (генеративні можливості моделі, які дозволяють інтерпретувати описові вхідні дані). Це дозволяє системі адаптуватися до неструктурованих або неповних ситуацій [86].

Приклади використання LLM:

- LLM може створити перелік потенційних хмарних сервісів чи шаблонів архітектури на основі опису задачі користувача, навіть якщо деякі сервіси не були явно вказані.
- Адаптація критеріїв до контексту задачі. Наприклад, у випадку тимчасової аналітики вкрай важлива масштабованість, тому LLM змінює ваги критеріїв відповідно до аналізованого контексту.
- LLM формує обґрунтовані пояснення у вигляді тексту, придатного для сприйняття як технічним спеціалістом, так і менеджером [90, 91].

Попри переваги, LLM-системи мають обмеження:

- відповіді можуть змінюватися при однакових запитах без контролю над джерелами знань;
- ризик суперечностей у поясненнях;
- LLM може вигадувати неіснуючі сервіси або формулювати помилкові рекомендації [91, 92].

Для цього в системі реалізуються механізми перевірки LLM-виходу через агентів, а також логічні фільтри, які порівнюють відповіді з базою знань [93].

На рис.4.4 зображено архітектуру гібридної системи, де LLM доповнює мультиагентну систему.

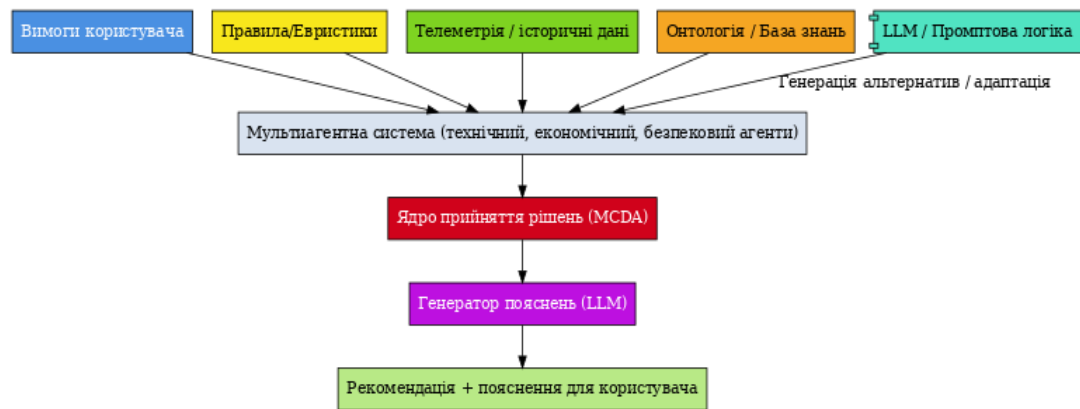


Рисунок 4.4 Архітектура мультиагентної системи прийняття рішень на базі LLM

Процес взаємодії LLM-агентів формалізується як спрямований граф комунікаційних потоків:

$$G_{comm} = (A_{LLM}, E_{comm}), \quad (4.35)$$

де:

$A_{LLM} = \{a_1, a_2, \dots, a_n\}$ – множина LLM-агентів;

$E_{comm} = \{e_{ij} \mid i, j \in [1, n]\}$ – множина комунікаційних потоків між агентами.

Кожен LLM-агент a_i визначається кортежем:

$$a_i = (M_i, P_i, K_i, F_i), \quad (4.36)$$

де:

M_i – мовна модель, що лежить в основі агента;

P_i – множина промтів та шаблонів, що використовуються для формування запитів до LLM;

K_i – база знань агента, що включає домен-специфічну інформацію;

F_i – функціональна спеціалізація агента (наприклад, технічна, економічна, безпекова).

Процес обробки інформації LLM-агентом включає наступні етапи:

1. Підготовка контексту:

$$C = PrepContext(I, K_i, H), \quad (4.37)$$

де I – вхідна інформація, K_i – база знань, H – історія взаємодії.

2. Формування запиту:

$$Q = FormatPrompt(C, P_i), \quad (4.38)$$

де P_i – шаблони промтів агента.

3. Генерація відповіді:

$$R = M_i(Q), \quad (4.39)$$

де M_i – мовна модель агента.

4. Постобробка відповіді:

$$O = PostProcess(R), \quad (4.40)$$

де *PostProcess* – функція постобробки, що може включати екстракцію ключової інформації, валідацію тощо.

Взаємодія між агентами здійснюється через механізм обміну повідомленнями:

$$MsgSend(a_i, a_j, content), \quad (4.41)$$

де a_i – агент-відправник, a_j – агент-отримувач, *content* – зміст повідомлення.

Для координації роботи LLM-агентів використовується спеціальний агент-координатор, який визначає послідовність обробки запитів та агрегує результати:

$$Coordinator = (A_{LLM}, S, R_{agg}), \quad (4.42)$$

де:

A_{LLM} – множина LLM-агентів;

S – стратегія координації, що визначає послідовність взаємодії агентів;

R_{agg} – функція агрегації результатів від різних агентів.

4.3.2 Алгоритм узгодження висновків агентів та формування остаточного рішення

Для прийняття оптимальних рішень у мультимарному середовищі необхідно узгоджувати висновки різних агентів, які можуть бути суперечливими або неповними. Розроблений алгоритм узгодження висновків використовує підходи консенсусу, компромісу та делегування.

Після отримання часткових оцінок від кожного з агентів мультиагентної системи (економічного, технічного, безпекового тощо), необхідно узгодити ці

висновки, з урахуванням ваги кожного критерію, рівня впевненості, а також можливих конфліктів.

Узгодження між агентами може відбуватися за допомогою:

- Консенсусного підходу – агенти ітеративно обмінюються оцінками, коригуючи результати до досягнення прийняттого порогу (наприклад, $\delta \leq 0.05$).
- Компромісного підходу – використовується агрегуюча функція, що не потребує повної згоди, але зберігає баланс між оцінками (наприклад, медіана, нечітка логіка).
- Механізм делегування – при конфлікті деякі агенти передають право прийняття рішення спеціалізованому агенту (наприклад, LLM-агенту узагальнення).

Кожен агент повертає не тільки значення оцінки, але й ступінь впевненості, що залежить від повноти даних, якості джерела, консенсусу з іншими агентами. Фінальна впевненість у рішенні може бути обчислена як середньозважена:

$$C = \frac{\sum w_i c_i}{\sum w_i}, \quad (4.43)$$

де c_i – впевненість агента i ,

w_i – вага критерію.

У разі конфлікту оцінок (наприклад, один агент дає «високу» оцінку, інший – «низьку») може бути застосована одна з наступних стратегій:

- Правила пріоритетності. Наприклад, якщо мета – мінімізація витрат, пріоритет надається економічному агенту.
- LLM-генерація пояснення і сценарного аналізу. Тобто LLM пропонує альтернативні рішення на базі компромісу.
- Виявлення аномалій. Система визначає, чи є конфлікт наслідком помилки в даних.

Фінальне рішення передається модулю генерації пояснень, який за допомогою LLM формує логіку вибору, порівняння альтернатив, коментарі щодо можливих ризиків чи переваг.

Алгоритм 2. Узгодження висновків агентів та формування фінального рішення.

Вхід: множина висновків агентів $C = \{c_1, c_2, \dots, c_n\}$, множина вагових коефіцієнтів $W = \{w_1, w_2, \dots, w_n\}$, поріг консенсусу δ .

Вихід: узгоджене рішення R .

1. Ініціалізувати узгоджене рішення R_0 .
2. Ініціалізувати лічильник ітерацій $i = 0$.
3. Поки не досягнуто консенсусу виконати:
 4. $i = i + 1$;
 5. для кожного агента j виконати:
 6. Обчислити відстань $d_j = \text{Distance}(C_j, R_{i-1})$;
 7. Визначити ступінь впевненості conf_j для висновку C_j ;
 8. Оновити узгоджене рішення: $R_i = \text{AggregateFn}(C, W, \{\text{conf}_j\})$
 9. Перевірити досягнення консенсусу: $\text{converged} = \text{CheckConvergence}(\{d_j\}, \delta)$
10. якщо $i > \text{MAX_ITERATIONS}$ то:
 11. Застосувати стратегію делегування або визначення пріоритетів;
 12. $\text{converged} = \text{true}$;
13. Виконати постобробку та валідацію R_i .
14. Згенерувати пояснення для узгодженого рішення.
15. Повернути R_i .

Формалізація ключових компонентів алгоритму:

1. Функція відстані між висновками агентів:

$$\text{Distance}(C_i, C_j) = \sum_k \alpha_k \cdot d_k(C_i[k], C_j[k]), \quad (4.44)$$

де d_k – метрика відстані для k -ї характеристики, α_k – вага цієї характеристики.

2. Ступінь впевненості агента у своєму висновку:

$$conf_j = f_{conf}(C_j, Data_j, History_j), \quad (4.45)$$

де f_{conf} – функція оцінки впевненості, $Data_j$ – дані, доступні агенту j , $History_j$ – історія висновків агента.

3. Функція агрегації для формування узгодженого рішення:

- Зважене усереднення:

$$R_i = \Sigma_j(w_j \cdot conf_j \cdot C_j) / \Sigma_j(w_j \cdot conf_j). \quad (4.46)$$

- Медіанний підхід:

$$R_i[k] = Median(\{C_j[k]\} \mid j \in [1, n]). \quad (4.47)$$

- Нечіткий підхід:

$$R_i = DefuzzifyFn(FuzzyAggregate(\{C_j\}, \{w_j\}, \{conf_j\})). \quad (4.48)$$

4. Перевірка досягнення консенсусу:

$$CheckConvergence(\{d_j\}, \delta) = true, \text{ якщо } Max(\{d_j\}) < \delta \quad (4.49)$$

або

$$CheckConvergence(\{d_j\}, \delta) = true, \text{ якщо } |R_i - R_{i-1}| < \delta. \quad (4.50)$$

5. Стратегії при відсутності консенсусу:

- Делегування. Вибір висновку агента з найвищим пріоритетом:

$$j * = argmax_j(w_j \cdot conf_j), \quad R_i = C_{\{j*\}} \quad (4.51)$$

Визначення пріоритетів. Застосування жорстких обмежень для вирішення конфлікту:

$$R_i = EnforcePriorityRules(C, P), \quad (4.52)$$

де P – множина пріоритетних правил.

6. Генерація пояснення для узгодженого рішення:

$$Explanation = LLM_{explain}(R_i, C, W, \{conf_j\}, DecisionHistory) \quad (4.53)$$

де $LLM_{explain}$ – функція генерації пояснення з використанням великої мовної моделі.

У випадках, коли агенти мають суперечливі висновки (наприклад, безпековий агент рекомендує конфігурацію, яка не є економічно оптимальною), використовуються наступні механізми вирішення конфліктів:

1. Ранжування альтернатив на основі мультикритеріального аналізу та домінування за Парето.
2. LLM-генерація компромісних рішень, що враховують ключові вимоги всіх агентів.
3. Переформулювання задачі для агента.

4.3.3 Сценарії використання мультиагентної системи

Для демонстрації ефективності розробленої мультиагентної системи розглянемо конкретні сценарії її використання у типових ситуаціях динамічного формування обчислювальної інфраструктури в мультихмарному середовищі.

Інтелектуальна система для підтримки прийняття рішень у мультихмарному середовищі повинна охоплювати повний життєвий цикл розгортання: від початкового планування до постійної адаптації у відповідь на зміну умов.

Розглянемо етапи життєвого циклу розгортання:

1. Планування. Визначаються вимоги користувача, критерії оцінювання, контекст задачі.
2. Вибір (оцінювання альтернатив). Запит обробляється агентами: продуктивності, вартості, впровадження, безпеки.
3. Розгортання (рекомендація). Ядро прийняття рішень формує оптимальний варіант з поясненням.
4. Моніторинг. У виробничому середовищі система отримує оновлення щодо

продуктивності, вартості, змін SLA або навантаження.

5. Адаптація. При зміні параметрів (наприклад, підвищення вартості у провайдера), LLM ініціює перегляд рішень, а система пропонує оновлену рекомендацію без втручання користувача.

Для створення мультиагентної AI-системи було обрано низку сучасних технологій, які дозволяють забезпечити ефективну роботу системи [11]. Основу складають моделі Large Language Models, такі як LLAMA 3, які відповідають за аналіз тексту та прийняття рішень.

Hugging Face Transformers (бібліотека з відкритим вихідним кодом, яка надає попередньо підготовлені моделі трансформаторів для обробки природної мови) дозволяє адаптувати модель до специфічних вимог.

Для координації роботи агентів використовується LangChain, який забезпечує взаємодію між компонентами системи.

Модулі обробки даних, такі як Pandas і NumPy, дозволяють ефективно працювати з інформацією.

Інфраструктурні рішення, включаючи AWS Lambda, Azure Functions та Pulumi, забезпечують масштабованість та гнучкість. Нарешті, для зберігання контексту та структурованих даних використовуються In-Memory Vector DB та PostgreSQL, що дозволяє зберігати необхідну інформацію для роботи мультиагентної системи.

Архітектура мультиагентної ШІ-системи представлена на рис.4.5.

Розглянемо основні етапи:

1. Початок процесу. Усе починається з користувацького запиту, який вводиться через Streamlit. Це зручний інтерфейс, що дозволяє інтерактивно працювати з нашим проєктом. Тут вводиться код або параметри для аналізу.

$$X = \{x_1, x_2, \dots, x_n\}, \quad (4.54)$$

де x_i – параметри, що описують хмарну інфраструктуру.

2. Екстракція метаданих. Вхідні дані передаються в блок екстракції метаданих,

який використовує LLAMA 3 LLM. Ця модель дозволяє розпізнати використані сервіси та інші залежності в проєкті, формуючи початкову аналітику.

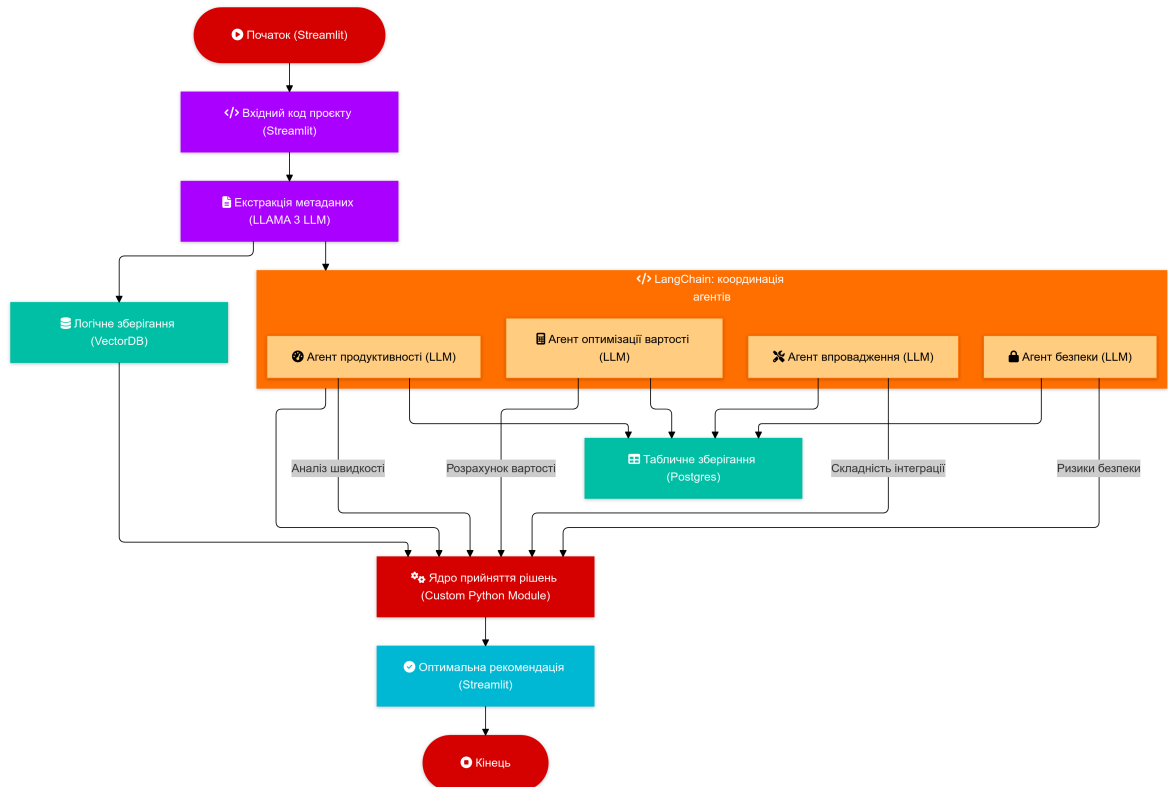


Рисунок 4.5 Архітектура мультиагентної ІІІ-системи

Функція перетворення даних: $f: X \rightarrow M$, де M – множина метаданих, що містить інформацію про сервіси, ресурси та їх характеристики.

3. Зберігання контексту. Дані передаються у VectorDB, який забезпечує логічне зберігання. Це важливий етап для збереження контексту, який використовується при ухваленні рішень.

$$S_{vectorDB} = \{m_1, m_2, \dots, m_k\}, m_k \in M, \quad (4.55)$$

VectorDB зберігає інформацію про попередні запити, що допомагає при майбутньому ухваленні рішень.

4. Робота агентів. Агенти працюють у системі LangChain. Кожен агент є

функцією, що приймає дані та повертає оцінку за своїм критерієм:

- Агент продуктивності (LLM) аналізує швидкість та затримки.

$$P = f_{pr}(M) = \sum_{i=1}^k w_i * p_i, \quad (4.56)$$

де p_i – показники продуктивності (наприклад, затримка, швидкість передачі),
 w_i – вагові коефіцієнти.

- Агент оптимізації вартості (LLM) оцінює економічну ефективність.

$$C = f_{cost}(M) = \sum_{i=1}^k c_i, \quad (4.57)$$

де c_i – вартість використання кожного хмарного сервісу.

- Агент безпеки (LLM) враховує ризики даних та доступу.

$$R = f_{risk}(M) = \sum_{i=1}^k v_i * r_i, \quad (4.58)$$

де r_i – рівень ризику, v_i – критичність ресурсу.

- Агент впровадження (LLM) аналізує складність інтеграції.

$$I = f_{impE}(M) = \sum_{i=1}^k s_i, \quad (4.59)$$

де s_i – складність інтеграції з існуючими рішеннями.

Кожен агент взаємодіє з Postgres, який слугує табличним сховищем для структурованої інформації та результатів обчислень. Запис проміжних даних у PostgreSQL:

$$T = \{(p_i, c_i, r_i, s_i)\}, \quad (4.60)$$

5. Ядро прийняття рішень. Усі дані, зібрані агентами, передаються в Ядро прийняття рішень. Це спеціальний модуль, написаний на Python, який агрегує дані, отримані від агентів, та зберігає їх у VectorDB для складних порівнянь.

$$S_{fin} = \alpha P + \beta \left(\frac{1}{C}\right) + \gamma \left(\frac{1}{R}\right) + \delta \left(\frac{1}{I}\right), \quad (4.61)$$

де $\alpha, \beta, \gamma, \delta$ – вагові коефіцієнти, що можуть змінюватися залежно від бізнес-пріоритетів.

6. Фінальна рекомендація. На основі зібраної інформації система генерує оптимальну рекомендацію, яка відображається у Streamlit. Це може бути рекомендація щодо найкращих сервісів, конфігурацій або платформ.

$$x_{opt} = \arg \max_{x_i} S_{final}(x_i) \quad (4.62)$$

Обраний варіант x_{opt} відображається у Streamlit як рекомендація.

Крім того, мультиагентна система повинна вміти отримувати актуальні дані з хмарних сервісів. Завдяки інтеграції з API хмарних провайдерів можна отримувати дані про вартість обчислювальних ресурсів, конфігурації віртуальних машин, доступних сервісів з відповідними параметрами продуктивності.

Щоб система залишалася актуальною, необхідний механізм динамічного оновлення знань. Наприклад, автоматичне оновлення записів на основі отриманих даних, самонавчання агентів на основі історичних даних, автоматичне виявлення нових патернів.

Сценарій 1. Розгортання вебзастосунку з динамічним навантаженням.

У цьому сценарії необхідно розгорнути вебзастосунок, який характеризується значними коливаннями навантаження протягом доби та має пікові періоди активності.

Вхідні параметри:

- Тип додатку: Web-сервер з базою даних;
- Очікуване пікове навантаження: 10000 запитів за секунду;
- Очікуване звичайне навантаження: 1000 запитів за секунду;
- Вимоги до затримки: < 100 ms для 95% запитів;
- Бюджетні обмеження: \$2000 на місяць.

Процес прийняття рішень у мультиагентній системі:

1. Аналіз вимог та класифікація навантаження.

Технічний агент визначає характеристики навантаження як різко змінні та рекомендує архітектуру з автомасштабуванням:

```
TA_result = {"architecture": "microservices",
             "scaling": "horizontal_autoscaling",
             "confidence": 0.85}
```


2. Оцінка економічної ефективності.

Економічний агент аналізує різні варіанти розподілу навантаження між провайдерами:

```
EA_result = {"primary_provider": "Provider_A",
             "burst_provider": "Provider_C",
             "cost_estimation": 1850,
             "confidence": 0.78}
```

3. Оцінка вимог безпеки.

Безпековий агент визначає вимоги до захисту даних та відповідності:

```
SA_result = {"data_classification": "moderate_sensitive",
             "geo_restrictions": ["EU"],
             "encryption": "in_transit_and_at_rest",
             "confidence": 0.92}
```

4. Узгодження висновків агентів.

Агент-координатор виявляє потенційний конфлікт: рекомендований економічним агентом провайдер не підтримує всі необхідні заходи безпеки. Застосовується процес узгодження за консенсусним підходом, який призводить до компромісного рішення:

```
Final_decision = {
    "primary_provider": "Provider_B",
    "burst_provider": "Provider_C",
    "architecture": "microservices",
    "scaling": "horizontal_autoscaling",
    "security_measures": ["in_transit_encryption", "at_rest_encryption"],
    "cost_estimation": 1920,
    "confidence": 0.83
}
```

5. Генерація пояснення:

LLM-агент генерує пояснення прийнятого рішення:

«Рекомендовано розгортання вебзастосунку з використанням мікросервісної архітектури на провайдері В як основному, з використанням провайдера С для обробки пікових навантажень. Хоча це рішення на \$70 дорожче, ніж початково запропоноване економічним агентом, воно забезпечує всі необхідні заходи безпеки, включаючи шифрування даних у спокої та при передачі. Система автомасштабування дозволить ефективно обробляти піки навантаження до 10 000 запитів за секунду, зберігаючи затримку в межах вимог (<100 ms).»

Сценарій 2. Міграція аналітичної системи до мультимарного середовища.

У цьому сценарії компанія хоче мігрувати існуючу аналітичну систему з власного центру даних до мультимарного середовища для підвищення масштабованості та зниження витрат на інфраструктуру.

Вхідні параметри:

- Тип системи: Аналітична платформа з великими даними;
- Обсяг даних: 50 ТБ;
- Регуляторні вимоги: Дані фінансової звітності мають зберігатися в межах країни;
- Вимоги до продуктивності: Виконання аналітичних запитів не довше 15 хвилин;
- Доступність: 99.9%.

Процес прийняття рішень:

1. Аналіз структури даних та навантаження.

Технічний агент аналізує структуру даних та визначає, що дані можна розділити на «гарячі» (часто використовувані) та «холодні» (архівні):

```
TA_result = {"data_segmentation": {"hot": 15TB, "cold": 35TB},
             "processing_requirements": "high_memory",
```

```
"confidence": 0.88}
```

2. Оцінка економічної моделі.

Економічний агент аналізує вартість зберігання та обробки даних у різних провайдерах:

```
EA_result = {"hot_data_provider": "Provider_A",
             "cold_data_provider": "Provider_B",
             "processing_provider": "Provider_A",
             "monthly_cost": 8500,
             "confidence": 0.82}
```

3. Оцінка відповідності регуляторним вимогам.

Безпечовий агент визначає, що частина даних має зберігатися в локальних дата-центрах:

```
SA_result = {"financial_data_location": "local_region",
             "data_classification": {"public": 20%, "confidential": 80%},
             "compliance_requirements": ["ISO27001", "GDPR"],
             "confidence": 0.95}
```

4. Узгодження висновків та формування фінального рішення:

Агент-координатор узгоджує висновки та формує гібридне рішення:

```
Final_decision = {
    "architecture": "hybrid_multicloud",
    "data_distribution": {
        "financial_data": "Provider_A_local_region",
        "hot_analytical_data": "Provider_A_global",
        "cold_data": "Provider_B_archive_storage"
    },
    "processing": {
        "standard_queries": "Provider_A",
```

```

    "batch_processing": "Provider_B_spot_instances"
  },
  "migration_strategy": "phased_approach",
  "estimated_cost": 7800,
  "confidence": 0.87
}

```

5. Планування процесу міграції:

LLM-агент генерує детальний план міграції з урахуванням всіх аспектів:

«Рекомендована поетапна міграція аналітичної системи з наступними фазами:

1. Підготовча фаза: налаштування ідентифікації та доступу в мультимарному середовищі (2 тижні).
2. Міграція холодних даних до архівного сховища Provider_B (3 тижні).
3. Створення інфраструктури для обробки аналітичних запитів у Provider_A (4 тижні).
4. Міграція гарячих даних та налаштування аналітичних процесів (2 тижні).
5. Впровадження механізмів синхронізації між різними середовищами (2 тижні).
6. Тестування та верифікація (3 тижні).

Запропонована архітектура дозволяє дотримуватись усіх регуляторних вимог при зменшенні щомісячних витрат на інфраструктуру на 22% порівняно з поточним рішенням.»

ВИСНОВКИ

1. Розроблено математичну модель інтегрованого показника ефективності виконання обчислювальних завдань, що враховує динаміку зміни стану інфраструктури мультимарного середовища та дозволяє системі адаптуватися до змін у реальному часі.

2. Формалізовано простір станів та дій для проектування та переналаштування мультимарної інфраструктури, функції винагороди для вирішення задачі багатокритеріальної оптимізації, що дозволило створити систему, здатну знаходити оптимальні архітектурні рішення в складному та динамічному просторі пошуку.

3. Розроблено архітектуру мультиагентної системи підтримки прийняття рішень, в якій спеціалізовані агенти (технічний, економічний, безпековий) оцінюють варіанти розгортання за різними критеріями, а їх висновки узгоджуються через механізми консенсусу, компромісу та делегування.

4. Запропоновано алгоритм оптимізації архітектури мультимарної інфраструктури, що поєднує підкріплювальне навчання та еволюційні алгоритми для ефективного пошуку Парето-оптимальних рішень у багатокритеріальному просторі.

5. Розроблено модель динамічної адаптації до зміни параметрів середовища, яка включає моніторинг стану системи, виявлення відхилень, прогнозування змін та вибір стратегії адаптації для забезпечення стабільної роботи в умовах змінного навантаження.

6. Запропоновано підхід до інтеграції великих мовних моделей (LLM) у процес прийняття рішень, що дозволяє розширити аналітичні можливості системи, забезпечити автоматизоване формування обґрунтувань та пояснень прийнятих рішень.

7. Формалізовано процес взаємодії LLM-агентів у мультимарному середовищі та розроблено алгоритм узгодження їх висновків для формування

фінального рішення, що підвищує надійність та адаптивність системи.

8. Розглянуто сценарії використання мультиагентної системи, що демонструють її ефективність у різноманітних практичних ситуаціях: розгортання web-застосунку з динамічним навантаженням, міграція аналітичної системи кінцевого користувача до мултихмарного середовища.

9. Запропонований комплексний метод забезпечує підвищення ефективності проєктування мултихмарної інфраструктури за рахунок динамічної адаптації до змін умов функціонування, оптимального розміщення завдань між сервісами різних хмарних провайдерів та підтримки прийняття рішень щодо переналаштування мултихмарної інфраструктури на основі багатокритеріального аналізу та машинного навчання.

РОЗДІЛ 5

ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНОГО ПІДХОДУ

5.1 Реалізація прототипу інтелектуальної системи для проєктування мультитимарної інфраструктури

З метою апробації запропонованого методологічного апарату автоматизованого проєктування мультитимарної платформи у гетерогенному мультитимарному середовищі розроблено прототип інтелектуальної системи підтримки прийняття рішень.

Реалізований прототип системи є програмним комплексом, розробленим на основі архітектурної концепції та моделей. Прототип системи включає всі необхідні компоненти для практичного застосування запропонованого підходу: від збору вхідних даних до генерації обґрунтованих рекомендацій щодо розгортання обчислювальної мультитимарної інфраструктури.

5.1.1 Архітектура та компоненти програмної реалізації

Програмна реалізація прототипу системи автоматизованого проєктування мультитимарної платформи побудована на модульній архітектурі, яка забезпечує гнучкість, масштабованість та можливість незалежного розвитку компонентів.

Загальна архітектура системи представлена на рис. 5.1. Вона складається з п'яти основних архітектурних рівнів: інтерфейсного, рівня агентів, аналітичного, інтеграційного та рівня зберігання даних.

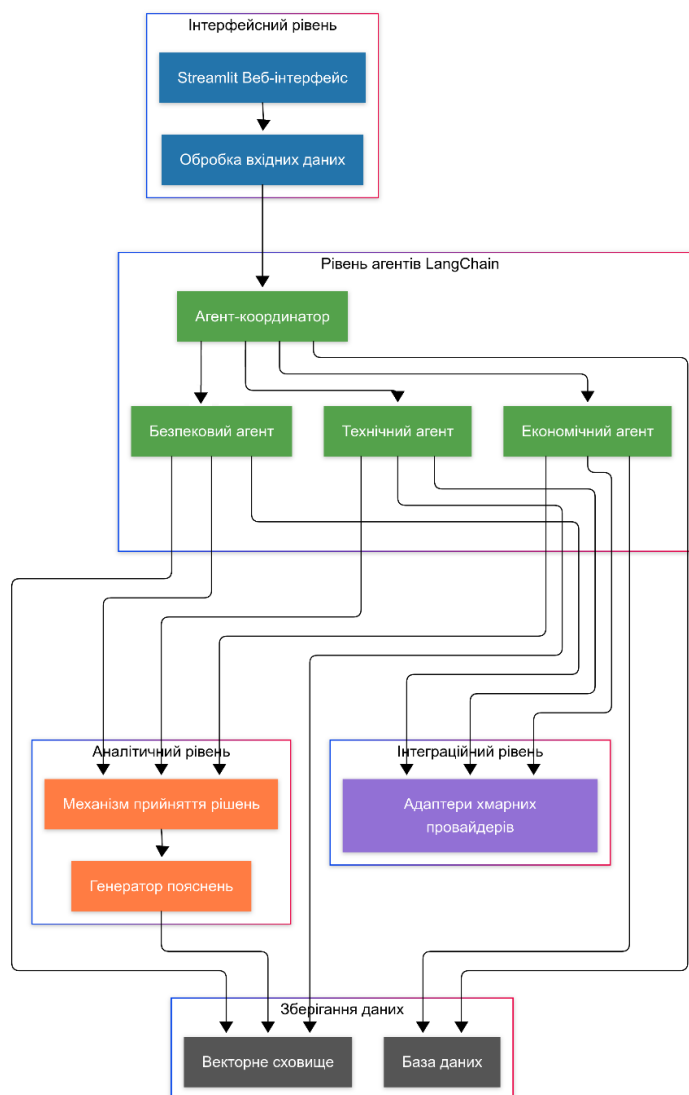


Рисунок 5.1 Загальна архітектура прототипу системи автоматизованого проєктування мултихмарної платформи

Інтерфейсний рівень. Забезпечує взаємодію користувачів із системою та включає наступні компоненти:

- Web-інтерфейс користувача для взаємодії з системою – побудований з використанням фреймворку Streamlit. Включає форми для введення вимог, візуалізацію результатів та інтерактивне налаштування параметрів.

- Модуль обробки вхідних даних – відповідає за валідацію, нормалізацію та структурування вхідних даних перед їх передачею аналітичному рівню.

Streamlit був обраний завдяки можливості швидкої розробки інтерактивних web-інтерфейсів з мінімальними затратами на розробку інтерфейсу користувача, що дозволило сконцентруватися на функціональності базових алгоритмів та моделей. Використання Python як основної мови для серверної частини, так і для інтерфейсу користувача забезпечило єдність технологічного стеку.

Рівень агентів. Є ключовим елементом архітектури та містить реалізацію спеціалізованих агентів, що працюють з різними аспектами проєктування мультихмарної інфраструктури. Окрім уже описаних агентів для практичного впровадження було додано Агент-координатор, який відповідає за координацію роботи спеціалізованих агентів, розподіл завдань та агрегацію результатів. Агент-координатор реалізований як центральний компонент мультиагентної системи з використанням LangChain. Він реалізує наступні функції:

- прийом вхідних запитів та їх структурування;
- розподіл завдань між спеціалізованими агентами;
- моніторинг виконання завдань та обробка помилок;
- агрегація результатів роботи агентів;
- формування фінального рішення.

Взаємодія агентів представлена на рис. 5.2.

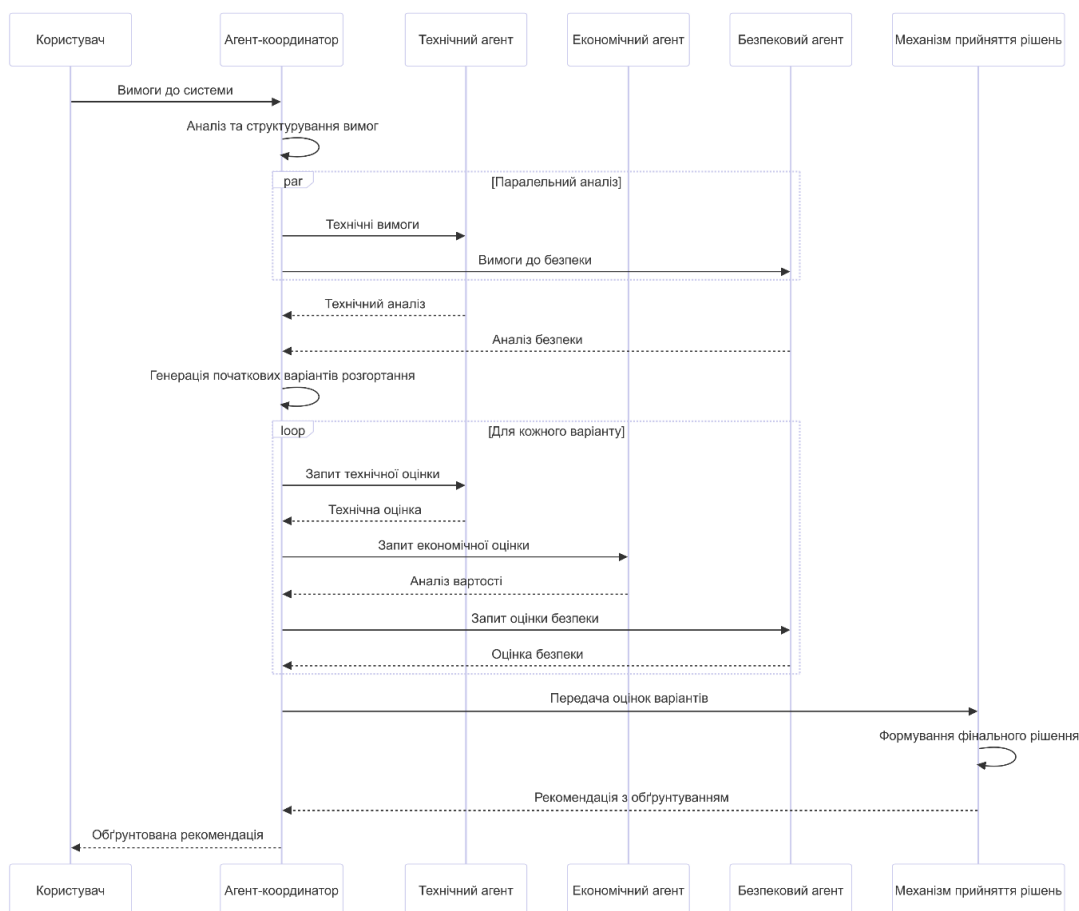


Рисунок 5.2 Діаграма взаємодії агентів

Аналітичний рівень. Містить компоненти, відповідальні за аналіз даних, моделювання та прийняття рішень:

- Механізм прийняття рішень – реалізує формальну модель простору станів та дій, формулюючи задачу проєктування мультимарної інфраструктури як багатокритеріальну оптимізаційну проблему та застосовуючи розроблені методи для її вирішення.
- Генератор пояснень – відповідає за формування зрозумілих пояснень та обґрунтувань рекомендацій системи, використовуючи можливості LLM для генерації текстових описів та візуалізацій.

Механізм прийняття рішень реалізує метод визначення оптимальної рекомендації, комбінуючи підходи навчання з підкріпленням та багатокритеріальної оптимізації для знаходження оптимальних конфігурацій мультихмарної інфраструктури.

Інтеграційний рівень. Забезпечує взаємодію з зовнішніми системами та сервісами:

- Адаптери хмарних провайдерів – набір модулів для взаємодії з API різних хмарних провайдерів (AWS, Azure, GCP тощо), що забезпечують уніфікований інтерфейс доступу до їх сервісів та інформації.

Адаптери реалізовані як обгортки навколо офіційних SDK хмарних провайдерів, що забезпечують єдиний інтерфейс для роботи з різними хмарними платформами. Це дозволяє системі абстрагуватися від специфічних особливостей API конкретних провайдерів та працювати з ними через уніфікований інструментарій.

Рівень зберігання даних. Включає компоненти для зберігання та керування даними системи:

- База даних – використовується для зберігання структурованих даних системи, включаючи інформацію про користувачів, завдання щодо проектування, історію рекомендацій тощо. Реалізована на основі SQLite для спрощення розгортання прототипу.

- Векторне сховище – спеціалізована база даних для ефективного зберігання та пошуку векторних представлень (ембедингів) тексту, що використовується для роботи з LLM. Реалізована з використанням ChromaDB для локального застосування.

Така архітектура забезпечує модульність, гнучкість та масштабованість системи, дозволяючи незалежно розвивати окремі компоненти та адаптувати систему до нових вимог та умов використання.

5.1.2 Реалізація програмного компоненту підтримки прийняття рішень

Програмний компонент підтримки прийняття рішень щодо переналаштування мультихмарної інфраструктури є ключовим компонентом розробленого прототипу, що забезпечує інтелектуальний аналіз різних аспектів проектування мультихмарної інфраструктури та формування обґрунтованих рекомендацій.

Для його реалізації було обрано комбінацію наступних технологій та інструментів:

- LangChain – фреймворк для побудови застосунків на основі мовних моделей, який використовується як основа для створення та координації агентів.
- Python – мова програмування, обрана для реалізації логіки агентів, інтеграцій та обробки даних завдяки її гнучкості, багатому екосистемі бібліотек та зручності для роботи з даними та AI-компонентами.
- SQLite – легка вбудована реляційна база даних, використовується для зберігання структурованих даних системи.
- ChromaDB – локальна векторна база даних, що забезпечує ефективний пошук за схожістю для текстових векторних представлень.

Приклади реалізації агентів наведено у Додатку 1.

Для забезпечення ефективної взаємодії між агентами в системі реалізовані механізми комунікації та координації на основі прямих викликів функцій та спільного доступу до даних. Такий підхід забезпечує простоту реалізації та розгортання системи, зберігаючи при цьому всі необхідні функціональні можливості.

Основні принципи взаємодії агентів:

1. Синхронна комунікація – агенти взаємодіють шляхом прямих викликів методів, що забезпечує простоту реалізації та відлагодження.
2. Спільний доступ до даних – агенти мають доступ до спільних баз даних та

інших ресурсів, що забезпечує узгодженість інформації.

3. Централізована координація – агент-координатор відповідає за розподіл завдань та агрегацію результатів, забезпечуючи узгодженість дій усіх.

Схема обміну даними між агентами реалізована на базі патерну спостерігач (Observer), що дозволяє агентам підписуватися на події та отримувати повідомлення про зміни. Для передачі структурованих даних використовується серіалізація у форматі JSON, що забезпечує незалежність від конкретної мови програмування та платформи.

Для синхронізації роботи агентів використовується механізм транзакцій, що гарантує атомарність виконання складних операцій. Кожна операція фіксується в журналі, що дозволяє відстежувати процес прийняття рішень та забезпечує можливість аудиту.

В реалізації прототипу системи використано бібліотеку LangChain, яка надає готові компоненти для побудови агентних систем на базі LLM. Кожен агент конфігурується за допомогою спеціального шаблону промпту, який містить інструкції для LLM щодо ролі, завдань та формату вихідних даних. Наприклад, шаблон промпту для технічного агента включає інструкції щодо аналізу технічних параметрів хмарних сервісів, оцінки їх продуктивності та масштабованості (Додаток 2).

Комунікація між агентами здійснюється за допомогою структурованих повідомлень, що містять метадані (тип повідомлення, відправник, отримувач) та корисне навантаження (дані). Це дозволяє реалізувати гнучкі схеми обміну повідомленнями, включаючи широкомовні повідомлення та фільтрацію за різними критеріями.

Для підвищення стійкості системи до помилок реалізовано механізм повторних спроб (retry mechanism) з експоненціальною затримкою. Якщо агент не може виконати завдання через тимчасові проблеми (наприклад, недоступність API

хмарного провайдера), система автоматично повторює спробу з поступово збільшуваною затримкою.

Таким чином, реалізовані механізми комунікації та координації агентів забезпечують ефективну взаємодію між компонентами системи, дозволяючи їй працювати як єдине ціле та успішно вирішувати задачі динамічного формування обчислювальної інфраструктури у мультихмарному середовищі.

5.1.3 Інтеграція методів машинного навчання та LLM

Ключовим аспектом розробленої системи є інтеграція великих мовних моделей для підвищення ефективності прийняття рішень. У даному підрозділі описано метод використання LLM в контексті проєктування мультихмарної інфраструктури.

Для реалізації прототипу системи було обрано модель Claude 3 Opus, яка демонструє високу ефективність у задачах розуміння та генерації тексту, а також аналізу структурованих даних. Дана модель має контекстне вікно розміром до 200К токенів, що дозволяє обробляти великі обсяги вхідних даних, включаючи детальні технічні специфікації хмарних сервісів, історичні дані про використання ресурсів та поточні параметри інфраструктури.

Для ефективного використання LLM в системі проєктування мультихмарної інфраструктури були розроблені спеціалізовані техніки промптингу:

1. Chain-of-Thought (ланцюжок міркувань) – спонукає модель покроково аналізувати проблему, що підвищує точність при вирішенні складних завдань оптимізації.

2. Role-Playing (рольова гра) – кожен агент отримує конкретну роль (технічний експерт, економіст, спеціаліст з безпеки), що допомагає моделі фокусуватися на відповідних аспектах проблеми.

3. JSON-структурування відповідей – спеціальне форматування вихідних даних у форматі JSON, що спрощує їх подальшу обробку.

Для підвищення точності відповідей LLM був розроблений механізм «retrieval-augmented generation» (RAG), який доповнює запити до моделі релевантною інформацією з векторної бази даних. Ця база даних містить актуальну інформацію про хмарні сервіси, їх характеристики, вартість та обмеження. При формуванні запиту до LLM система автоматично відбирає найбільш релевантні фрагменти інформації та включає їх у контекст запиту.

Інтеграція LLM з іншими компонентами системи здійснюється за допомогою фреймворку LangChain, який надає гнучкі засоби для побудови ланцюжків обробки даних (chains) та агентів на базі мовних моделей. Архітектура інтеграції LLM описана в Додатку 3.

Взаємодія з LLM здійснюється через API, що дозволяє абстрагуватися від конкретної реалізації моделі та за необхідності замінити її на іншу без суттєвих змін у коді системи. Для підвищення надійності реалізовано механізм обробки помилок та повторних запитів у випадку збоїв.

LLM дозволяє динамічно коригувати ці вагові коефіцієнти на основі аналізу контексту задачі та пріоритетів користувача, що підвищує адаптивність системи до різних сценаріїв використання.

Інтеграція LLM в систему проектування мультихмарної інфраструктури надає ряд суттєвих переваг:

1. Здатність обробляти неструктуровані вхідні дані – система може працювати з вимогами, сформульованими природною мовою.
2. Гнучкість та адаптивність – LLM може адаптуватися до нових типів сервісів та сценаріїв використання без необхідності перепрограмування системи.
3. Генерація зрозумілих пояснень – система надає обґрунтування своїх рекомендацій у форматі, зрозумілому для користувачів різного рівня технічної підготовки.

4. Врахування неявних залежностей – LLM може виявляти неявні залежності між різними аспектами проєктування інфраструктури.

Однак використання LLM також має певні обмеження, які враховані в архітектурі системи:

1. Потенційні неточності та «галюцинації» – LLM може генерувати невірну інформацію. Для мінімізації цього ризику система використовує валідацію відповідей та перевірку фактів через зовнішні джерела даних.

2. Обмеження контекстного вікна – незважаючи на значний розмір контекстного вікна (200K токенів), воно все ж обмежене. Система вирішує цю проблему через техніки ефективного управління контекстом та компресії інформації.

3. Залежність від якості промптів – ефективність LLM суттєво залежить від якості формулювання запитів. У системі реалізовано механізм автоматичного покращення промптів на основі аналізу результатів.

4. Обчислювальні вимоги – взаємодія з LLM вимагає значних обчислювальних ресурсів. Система оптимізує використання LLM через кешування результатів та застосування легших моделей для простіших завдань.

Для подолання цих обмежень в системі реалізовано гібридний підхід, який поєднує можливості LLM з класичними алгоритмами та спеціалізованими компонентами для виконання конкретних завдань.

5.2 Методика експериментального дослідження

Для оцінки ефективності розробленої системи автоматизованого проєктування мультихмарної платформи за динамічно змінюваним набором критеріїв було проведено комплексне експериментальне дослідження, спрямоване на перевірку її функціональності та продуктивності в різних сценаріях використання.

5.2.1 Визначення умов експерименту

Для демонстрації ефективності розробленої мультиагентної системи розглянемо конкретні сценарії її використання у типових ситуаціях динамічного формування обчислювальної інфраструктури в мультихмарному середовищі.

Експериментальне дослідження було спрямоване на досягнення наступних цілей:

1. Оцінка ефективності розробленої системи в порівнянні з традиційними підходами до проєктування мультихмарної інфраструктури.
2. Визначення здатності системи адаптуватися до різних типів навантаження та вимог користувачів.
3. Перевірка точності рекомендацій, що надаються системою, та їх відповідності заданим критеріям оптимізації.
4. Оцінка масштабованості системи та її здатності обробляти складні сценарії з великою кількістю параметрів.
5. Визначення впливу використання LLM на якість та швидкість прийняття рішень.

На основі цих цілей були сформульовані наступні вимоги:

Вимога 1. Мультиагентна система з використанням LLM забезпечує більш оптимальні рішення щодо проєктування мультихмарної інфраструктури порівняно з традиційними підходами, що базуються на фіксованих правилах та евристиках.

Вимога 2. Система здатна ефективно враховувати різноманітні критерії оптимізації (вартість, продуктивність, безпека) та знаходити оптимальний баланс між ними.

Вимога 3. Використання LLM дозволяє системі адаптуватися до нових типів сервісів та сценаріїв використання без необхідності перепрограмування.

Вимога 4. Інтегрований показник ефективності, описаний формулою (3.1), адекватно відображає якість рекомендованих рішень у різних контекстах використання.

5.2.2 Вибір та обґрунтування метрик оцінювання

Для оцінки ефективності розробленої системи були визначені наступні метрики, що охоплюють різні аспекти її функціонування:

1. Економічна ефективність – оцінюється через порівняння вартості рекомендованих рішень з базовими підходами. Нормалізована оцінка: $Cost_{norm} = (Cost_{max} - Cost_{actual}) / (Cost_{max} - Cost_{min})$, де $Cost_{max} = 3000$ USD (бюджетне обмеження) $Cost_{min} = 1200$ USD (теоретичний мінімум).

2. Технічна ефективність – оцінюється через комплексний показник, що враховує продуктивність, масштабованість та надійність рекомендованої конфігурації: $E_{tech} = w_p P + w_s S + w_r R$ – нормалізовані оцінки продуктивності, масштабованості та надійності відповідно, а w_r , w_p , w_s – відповідні вагові коефіцієнти.

3. Безпекова ефективність – оцінюється через відповідність рекомендованих рішень заданим вимогам безпеки та нормативним обмеженням: $E_{sec} = \frac{N_{sat}}{N_{tot}} * 100\%$, де N_{sat} – кількість задоволених вимог безпеки, а N_{tot} – загальна кількість вимог.

4. Час прийняття рішення – вимірюється як час, необхідний системі для формування рекомендації після отримання вхідних даних: $T_{dec} = T_{end} - T_{start}$, де T_{start} – час початку обробки запиту, а T_{end} – час надання відповіді.

5. Якість пояснень – оцінюється експертами за шкалою від 1 до 10 на основі повноти, зрозумілості та обґрунтованості пояснень, наданих системою.

6. Інтегральний показник ефективності – розраховується на основі формули (3.1) з урахуванням нормалізованих значень усіх попередніх метрик.

5.2.3 Опис експериментального середовища та тестових сценаріїв

Експериментальне дослідження проводилось у гібридному середовищі, що включало:

1. Тестове середовище для розгортання – комбінація реальних хмарних ресурсів (AWS, Azure, GCP) та емульованого середовища для тестування різних сценаріїв.

2. Набір еталонних даних – включав історичні дані про вартість, продуктивність та характеристики різних хмарних сервісів.

3. Система моніторингу – дозволяла відстежувати ключові параметри функціонування системи та збирати дані для аналізу.

Для апробації системи були розроблені наступні тестові сценарії:

Сценарій 1. Розгортання web-застосунку з динамічним навантаженням.

Тип застосунку: web-сервіс з балансуванням навантаження.

Параметри: пікове навантаження до 10 000 запитів за секунду, вимоги до латентності < 100 мс.

Додаткові умови: необхідність швидкого масштабування при зростанні навантаження.

Сценарій 2. Міграція аналітичної системи до мультихмарного середовища.

- Тип системи: аналітична платформа з обробкою великих даних.
- Параметри: обсяг даних 50 ТБ, необхідність дотримання регуляторних вимог щодо зберігання даних.
- Додаткові умови: мінімізація втрат даних та простоїв під час міграції.

Сценарій 3. Розгортання системи з суворими вимогами до безпеки.

- Тип системи: фінансовий додаток з обробкою конфіденційних даних.
- Параметри: необхідність відповідності стандартам PCI DSS, GDPR.
- Додаткові умови: географічні обмеження на розміщення даних.

Сценарій 4. Оптимізація існуючої мультихмарної інфраструктури.

- Тип задачі: зниження витрат при збереженні поточного рівня продуктивності.
- Параметри: поточна вартість інфраструктури, показники використання ресурсів.
- Додаткові умови: мінімізація впливу процесу переналаштування мультихмарної інфраструктури на поточні операції які виконуються в даний час.

Для кожного сценарію визначено базові рішення, отримані традиційними методами (ручне проєктування експертами, використання спеціалізованих інструментів без LLM), що дозволило провести порівняльний аналіз ефективності розробленої системи.

5.3 Експериментальна перевірка ефективності запропонованого підходу

Для перевірки ефективності розробленої мультиагентної системи автоматизованого проєктування мультихмарної платформи важливо обрати репрезентативні сценарії, які б відображали реальні потреби користувачів в мультихмарному проєктуванні. Згідно з інформацією компаній-інтеграторів мультихмарних рішень, з якими було проведено попередні консультації (Nordcloud, Crayon та Tech-5), одними з найбільш типових завдань для мультихмарного середовища є міграція і оптимізація web-застосунків з REST API та систем аналізу й обробки даних (ETL-систем).

Web-застосунки з REST API є сполучною ланкою між різними сервісами та системами, демонструють варіативне навантаження та вимагають гнучкого масштабування.

ETL-системи (Extract, Transform, Load) виконують операції з обробки великої кількості даних часто вимагають значних обчислювальних ресурсів, які можуть

бути розподілені між різними хмарними провайдерами для оптимізації витрат та продуктивності. Крім того, такі системи працюють з різними типами даних, які можуть підпадати під різні регуляторні вимоги, що робить задачу їх розміщення особливо складною.

Обрані сценарії охоплюють найбільш поширені та показові випадки використання мультихмарних середовищ, що дозволяє оцінити ефективність запропонованого підходу формування обчислювальної інфраструктури у гетерогенному мультихмарному середовищі.

5.3.1 Сценарій 1. Розгортання web-застосунку з REST API

Для тестування першого сценарію було обрано не виробничу систему, а тестовий приклад, побудований на основі відкритого програмного забезпечення Django CMS, що є гнучкою системою управління контентом з REST API. Даний вибір обумовлений тим, що Django CMS є репрезентативним прикладом web-застосунку з типовою архітектурою, що включає frontend, backend, базу даних та API-шар, а також має відкриту кодову базу, що дозволяє гнучко налаштовувати конфігурацію для тестування.

Для експерименту було використано Django CMS 3.11 з розширеннями для REST API та додатковими модулями для моделювання високонавантаженого середовища. Характеристики тестового застосунку включали:

- Frontend: React.js з серверним рендерингом.
- Backend: Django 3.2 з Django REST Framework.
- База даних: PostgreSQL 13.
- Кешування: Redis.
- Медіасховище: об'єктне сховище S3-сумісне.
- Середній розмір сторінки: 1.2 MB.
- Кількість контентних об'єктів: 50 000.
- Розмір бази даних: 8 GB.

- Обсяг медіафайлів: 120 GB.

Вимоги до розгортання включали:

- Пікове навантаження: 1 500 запитів на секунду.
- Середнє навантаження: 300 запитів на секунду.
- Цільова відповідь сервера: <200 мс для 95% запитів.
- Доступність: 99.95%.
- Бюджетні обмеження: 3 000 USD на місяць.
- Географічне розташування користувачів: Європа (70%), Північна Америка (20%), Азія (10%).
- Безпекові вимоги: відповідність GDPR, шифрування даних у спокої та при передачі.

Процес аналізу та прийняття рішень мультиагентною системою включав наступні етапи:

1. Аналіз вхідних вимог був виконаний агентом-координатором, який структурував інформацію та виділив ключові параметри (Додаток 4).

2. Технічний агент проаналізував технічний стек та визначив можливі варіанти розгортання (Додаток 5).

3. Економічний агент проаналізував вартість різних варіантів розгортання (Додаток 6).

4. Безпековий агент проаналізував відповідність рішень вимогам безпеки (Додаток 7).

5. Агент-координатор агрегував результати та сформував фінальну рекомендацію (Додаток 8).

Розрахунок інтегрованого показника для AWS+Azure Hybrid:

Вхідні дані:

- Вартість: 1673 USD (Додаток 6) $Cost_norm = (3000-1673)/(3000-1200) = 0.737$.
- Продуктивність: 0.88 (з технічного аналізу).

- Надійність: 0.89 (з аналізу відмовостійкості).
- Сумісність: 0.92 (оцінка взаємодії AWS+Azure).
- Безпека: 0.96 (з безпекового аналізу).

При вагових коефіцієнтах $w_1=0.3$, $w_2=0.25$, $w_3=0.2$, $w_4=0.15$, $w_5=0.1$:
 $IE(\text{AWS}+\text{Azure}) = 0.3 \times 0.737 + 0.25 \times 0.88 + 0.2 \times 0.89 + 0.15 \times 0.92 + 0.1 \times 0.96 = 0.221 + 0.220 + 0.178 + 0.138 + 0.096 = 0.853$.

Порівняння альтернатив:

- AWS Full: $IE = 0.3 \times 0.513 + 0.25 \times 0.92 + 0.2 \times 0.94 + 0.15 \times 0.85 + 0.1 \times 0.94 = 0.791$
- GCP Full: $IE = 0.3 \times 0.622 + 0.25 \times 0.90 + 0.2 \times 0.92 + 0.15 \times 0.91 + 0.1 \times 0.93 = 0.814$

Рекомендація: AWS+Azure Hybrid (найвищий $IE = 0.853$).

5.3.2 Сценарій 2. ETL-система обробки даних

Для тестування другого сценарію було обрано тестовий приклад ETL-системи на базі відкритого програмного забезпечення Apache Airflow з додатковими компонентами для аналізу даних. Apache Airflow було обрано як репрезентативний приклад сучасної системи управління робочими потоками даних, яка широко використовується для реалізації ETL-процесів у різних галузях. Система була налаштована з набором демонстраційних даних та робочих процесів для моделювання реальних сценаріїв використання.

Характеристики тестової ETL-системи включали:

- Основний фреймворк: Apache Airflow 2.3.
- Обробка даних: Apache Spark 3.2 та Python 3.9.
- Сховище даних: Amazon S3 (для вхідних даних), Snowflake (для обробленої аналітики).
- Моніторинг: Prometheus з Grafana.
- Загальний обсяг даних на вході: 500 GB на день.
- Типи даних: структуровані CSV (40%), JSON (30%), неструктуровані текстові документи (20%), напівструктуровані XML (10%).

- Кількість ETL-процесів: 25 різних робочих потоків з різною періодичністю.
- Найбільший робочий процес: обробка 200 GB даних, час виконання до 4 годин.

Вимоги до розгортання включали:

- Пікове використання обчислювальних ресурсів: 500 CPU-годин на день;
- Середнє використання пам'яті: 2 TB-годин на день;
- Часові вікна для виконання критичних ETL-процесів: з 01:00 до 05:00 за UTC;
- Обмеження на час виконання: всі щоденні процеси мають завершуватися за 6 годин;
- Бюджетні обмеження: 8 000 USD на місяць;
- Регуляторні вимоги: фінансові дані мають оброблятися та зберігатися у регіонах, що відповідають вимогам PCI DSS;
- Вимоги до надійності: успішне виконання 99.5% ETL-процесів.

Процес аналізу та прийняття рішень мультиагентною системою для цього сценарію включав наступні етапи:

1. Аналіз вхідних вимог агентом-координатором (Додаток 9).
2. Технічний агент проаналізував технічний стек та сформулював рекомендації (Додаток 10).
3. Економічний агент проаналізував вартість різних варіантів розгортання (Додаток 11).
4. Безпековий агент проаналізував відповідність рішень вимогам безпеки (Додаток 12).
5. Агент-координатор агрегував результати та сформував фінальні рекомендації (Додаток 13).

Основні результати впровадження рекомендованого рішення:

- Зниження загальної вартості на 23% порівняно з початковою оцінкою AWS+Snowflake.
- Покращення продуктивності аналітичних запитів на 20% завдяки BigQuery.
- Спрощення адміністрування та моніторингу завдяки повній інтеграції сервісів у межах однієї платформи.
- Зменшення часу виконання найбільшого робочого процесу з 4 до 2.5 годин.

Цей сценарій продемонстрував здатність системи аналізувати та рекомендувати оптимальні рішення для складних ETL-процесів, враховуючи як технічні, так і економічні аспекти, а також вимоги безпеки та відповідності регуляторним нормам.

5.4 Практичне впровадження та апробація в промисловому середовищі

Для підтвердження практичної цінності системи автоматизованого проектування мультимарної платформи за динамічно змінюваним набором критеріїв проведено її впровадження та апробацію в реальному промисловому середовищі. У цьому підрозділі описано процес практичного експерименту, отримані результати та відгуки експертів. Детальна інформація про впровадження, включаючи технічні специфікації, методологію та кількісні результати, представлена в акті впровадження, що додається до дисертаційної роботи.

5.4.1 Опис умов та процесу впровадження

Для практичного експерименту було обрано існуючий ETL-процес, який вже був розгорнутий на EC2 машинах та налаштований вручну. Цей процес використовувався в одній з великих виробничих компаній для обробки даних із виробничих ліній та подальшого аналізу ефективності устаткування. Завданням було автоматизоване проектування оптимальної мультимарної інфраструктури для цього процесу.

Ключові характеристики існуючої інфраструктури:

- 5 EC2 інстансів (m5.xlarge) для обробки даних;
- MySQL база даних на окремому EC2 інстансі (r5.large);
- Jenkins для запуску та планування ETL-процесів;
- Bash- та Python-скрипти для трансформації даних;
- загальний обсяг оброблюваних даних: ~50 ГБ щоденно;
- відсутність автоматичного масштабування;
- ручне керування інфраструктурою через AWS Console;
- періодичні проблеми з продуктивністю та відмовостійкістю.

Вартість існуючого рішення становила приблизно 2500 USD на місяць, включаючи витрати на обчислювальні ресурси, зберігання даних та мережевий трафік. Крім того, компанія витрачала близько 20 людино-годин на місяць на підтримку та адміністрування цієї інфраструктури.

5.4.2 Результати практичного використання

Для аналізу існуючої ETL-системи було використано розроблений програмний комплекс. Кожен агент провів детальний аналіз за своїм напрямком і надав структуровані рекомендації.

1. Технічний агент проаналізував лог-файли, конфігураційні файли та код ETL-процесів. Відповідно до формули (4.19): $TA(v) = w_{t_1} \cdot perf(v) + w_{t_2} \cdot scalab(v) + w_{t_3} \cdot compat(v)$ агент оцінив продуктивність, масштабованість та сумісність існуючої інфраструктури, виявивши наступні важливі аспекти:

- Неефективне використання ресурсів через фіксовану кількість EC2 інстансів незалежно від поточного навантаження.
- Відсутність автоматичного масштабування та механізмів відмовостійкості.
- Обмеження масштабованості через архітектуру з прямими з'єднаннями до бази даних.

- Висока затримка виконання через неоптимальне завантаження та обробку даних.

На основі аналізу, технічний агент запропонував перехід до безсерверної архітектури з використанням AWS Step Functions, Lambda та S3, що краще відповідає потребам системи за умовами масштабованості та продуктивності (Додаток 14).

2. Економічний агент оцінив поточні витрати та можливості оптимізації відповідно до функції оцінки (4.20): $EA_{norm}(v) = (Cost_{max} - Cost(v)) / (Cost_{max} - Cost_{min})$.

Оцінка вартості наведена у Додатку 15.

3. Безпековий агент перевіряв відповідність вимогам безпеки та надав свої рекомендації згідно з функцією оцінки (4.21):

$$SA(v) = w_{s_1} \cdot data_sec(v) + w_{s_2} \cdot compl(v) + w_{s_3} \cdot resil(v)$$

Аналіз показав наступні проблеми в існуючій системі:

- Використання спільних облікових даних, впроваджених у файли конфігурації.
- Відсутність детального контролю доступу.
- Непослідовне застосування шифрування.
- Обмежені можливості аудиту.

Агент запропонував перехід до моделі безпеки AWS з використанням IAM з детальними ролями, стандартного шифрування S3 та включенням CloudTrail для аудиту API, що підвищило загальний показник безпеки з 0.65 до 0.91 (Додаток 16).

4. На основі аналізу всіх спеціалізованих агентів, агент-координатор сформулював фінальну рекомендацію, оцінивши різні архітектурні. Інтегральна оцінка кожного варіанту була розрахована за формулою (4.22):

$$AE(v) = w_{TA} \cdot TA_{norm}(v) + w_{EA} \cdot EA_{norm}(v) + w_{SA} \cdot SA_{norm}(v)$$

Було розглянуто наступні архітектурні варіанти:

1. Оптимізована архітектура на основі EC2 (інтегральний показник: 0.69).
2. Гібридна архітектура EC2/Serverless (інтегральний показник: 0.77).
3. Повністю безсерверна архітектура (інтегральний показник: 0.91).
4. Мультихмарна безсерверна архітектура (AWS + GCP) (інтегральний показник: 0.82).

Рекомендованим рішенням стала повністю безсерверна архітектура з AWS Step Functions, Lambda, S3 та Athena, яка забезпечувала найкращий баланс між технічними можливостями, економічною ефективністю та безпекою (Додаток 17).

На основі проведеного аналізу система автоматизованого проєктування мультихмарної платформи рекомендувала суттєву трансформацію архітектури ETL-процесу з переходом на сучасну безсерверну архітектуру, що складається з наступних компонентів, які наведені у Додатку 18.

Прогнозовані результати впровадження цієї архітектури:

- зниження щомісячних витрат з \$2 500 до \$1 200 (економія 52%);
- зменшення операційних витрат на адміністрування на 85%;
- підвищення продуктивності обробки даних на 30%;
- забезпечення автоматичного масштабування від нуля до пікового навантаження.

Для кількісної оцінки ефективності запропонованих змін застосуємо інтегрований показник ефективності (формула 4.1):

Поточна система:

- Вартість: 4500 USD. $\text{Cost_norm} = (5000-4500)/(5000-1500) = 0.143$.
- Продуктивність: 0.65 (з аналізу існуючої системи, час обробки 3.5 год).
- Надійність: 0.70 (failure rate 8%, відсутність автовідновлення).
- Сумісність: 0.60 (ручне управління, обмежена інтеграція).
- Безпека: 0.65 (з аналізу безпекового агента, Додаток 16).

При стандартних вагових коефіцієнтах $w_1=0.3$, $w_2=0.25$, $w_3=0.2$, $w_4=0.15$, $w_5=0.1$: $IE(\text{поточна}) = 0.3 \times 0.143 + 0.25 \times 0.65 + 0.2 \times 0.70 + 0.15 \times 0.60 + 0.1 \times 0.65 = 0.501$.

Рекомендована безсерверна система:

- Вартість: 1500 USD. $Cost_norm = (5000-1500)/(5000-1500) = 1.000$.

- Продуктивність: 0.89 (30% покращення, автомасштабування).

- Надійність: 0.95 (автоматичне відновлення, managed services).

- Сумісність: 0.88 (нативна AWS інтеграція).

- Безпека: 0.91 (IAM + CloudTrail + шифрування, Додаток 16).

$IE(\text{рекомендована}) = 0.3 \times 1.000 + 0.25 \times 0.89 + 0.2 \times 0.95 + 0.15 \times 0.88 + 0.1 \times 0.91 = 0.94$.

Покращення інтегрованого показника: $\Delta IE = (0.94 - 0.50)/0.50 \times 100\% = 88\%$.

Після проведення детального аналізу рекомендованої архітектури, компанія прийняла рішення впровадити запропоновані зміни. Міграція відбувалася поетапно протягом 4 тижнів, щоб мінімізувати ризики та забезпечити безперервність бізнес-процесів. Аналіз відгуків фахівців та експертних оцінок наведено у Додатку 19.

Результати впровадження підтвердили прогнозовані покращення:

- Фактичне зниження витрат: 67% (з \$4 500 до \$1 500).

- Покращення швидкості обробки: 37% (з 3.5 до 2.2 години).

- Зменшення кількості збоїв: 85% (з 8% до 1.2% failure rate).

- Зменшення часу адміністрування: 90% (з 20 до 2 годин/місяць).

Фактичний інтегрований показник після впровадження: $IE = 0.912$. Відхилення від прогнозу: +0.9%, що підтверджує точність моделі.

5.5 Аналіз результатів та рекомендації щодо подальшого розвитку

5.5.1 Узагальнення результатів експериментального дослідження

Для практичного експерименту було обрано існуючий ETL-процес, який вже був розгорнутий на EC2 машинах та налаштований вручну.

Проведені експерименти та практичне впровадження дозволяють зробити наступні ключові висновки щодо ефективності розробленої системи автоматизованого проєктування мультимарної платформи:

1. Використання мультиагентного підходу з інтеграцією LLM забезпечує суттєве підвищення якості рекомендацій порівняно з традиційними методами. Система демонструє здатність враховувати широкий спектр факторів та знаходити неочевидні компроміси між різними критеріями оптимізації.

2. Інтегрований показник ефективності, запропонований у формулі (4.1):

$$IE(S, t) = w_1 \cdot Cost_{norm}(S, t) + w_2 \cdot Perf_{norm}(S, t) + w_3 \cdot Rel_{norm}(S, t) + w_4 \cdot Inter_{norm}(S, t) + w_5 \cdot Sec_{norm}(S, t)$$
 підтвердив свою адекватність для оцінки якості рішень при проєктуванні мультимарної інфраструктури у різних сценаріях використання. Використання динамічних вагових коефіцієнтів дозволяє адаптувати оцінку до конкретних пріоритетів користувача.

3. Система демонструє високу адаптивність до нових типів сервісів та сценаріїв використання завдяки можливостям LLM до узагальнення та переносу знань. Це підтверджує гіпотезу про можливість створення системи, що не потребує постійного перепрограмування при зміні умов.

4. Практичне впровадження в промисловому середовищі продемонструвало суттєвий економічний ефект від використання системи автоматизованого проєктування мультимарної платформи: зниження витрат на інфраструктуру на 52% при одночасному підвищенні продуктивності та масштабованості.

5. Процес прийняття рішень став більш прозорим та об'єктивним завдяки використанню формальних критеріїв оцінки та детальних пояснень, що генеруються системою.

Компоненти інтегрованого показника ефективності представлені у таблиці 5.1.

Таблиця 5.1 Компоненти інтегрованого показника ефективності

Експеримент	$Cost_{norm}$	$Perf_{norm}$	Rel_{norm}	$Inter_{norm}$	Sec_{norm}	IE
Сценарій 1:						
AWS Full	0,47	0,92	0,94	0,85	0,94	0,76
AWS+Azure	0,74	0,88	0,89	0,92	0,96	0,85
Сценарій 2:						
AWS+Snowflake	0,56	0,95	0,93	0,85	0,97	0,81
GCP Full	0,76	0,87	0,90	0,91	0,93	0,90
Практичне впровадження						
EC2-based ETL	0,14	0,65	0,70	0,60	0,65	0,50
Serverless AWS	1,00	0,89	0,95	0,88	0,91	0,94

Таблиця 5.2 Практичні результати агентів за архітектурними рішеннями

Архітектурне рішення	Serverless AWS	AWS+Azure	GCP	AWS+Snowflake	Оптимізована EC2 архітектура
Реальна економія витрат (%)	47	28	23	19	8
Оцінка економічного агента (0-1)	0,95	0,82	0,78	0,72	0,62
Покращення продуктивності (%)	25	15	20	37	51
Оцінка агента продуктивності (0-1)	0,89	0,75	0,82	0,91	0,68

Для підтвердження ефективності запропонованого комплексного методу формування обчислювальної інфраструктури було проведено детальний аналіз точності прогнозування мультиагентної системи шляхом порівняння оцінок агентів

з практичними результатами впровадження архітектурних рішень (таблиця 5.2). На рисунку 5.3 представлено результати валідації прогнозів економічного агента, де показано співвідношення між оцінками агента (шкала 0-1) та фактичною економією витрат у відсотках від початкового стану.

Результати прогнозу економічного агента

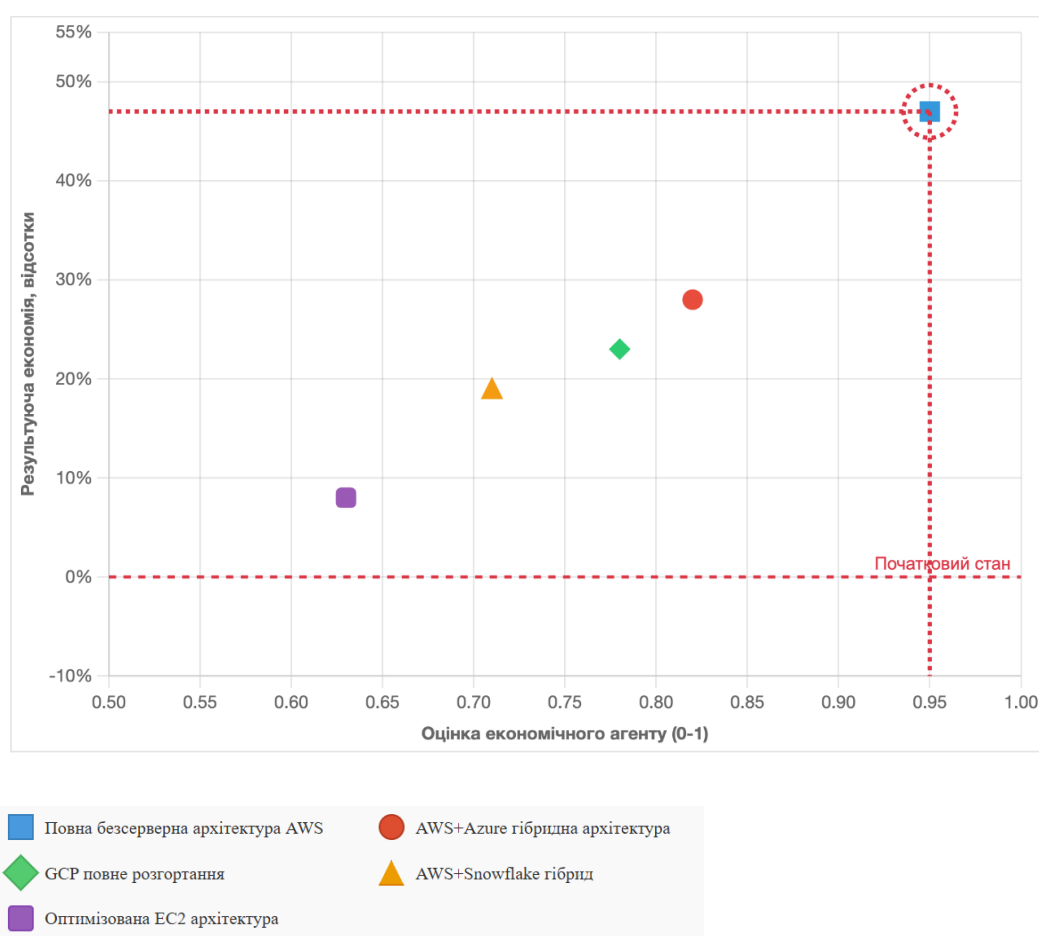


Рисунок 5.3 Результати прогнозу економічного агента

Аналіз п'яти різних архітектурних рішень демонструє високу кореляцію ($r = 0.89$) між прогнозованими та практичними результатами. Обране рішення “Повна

безсерверна архітектура AWS” показало максимальну економію 47%, що відповідає найвищій оцінці економічного агента (0.95). Рисунок 5.4 відображає результати валідації агента продуктивності, де порівнюються оцінки агента з фактичним покращенням продуктивності у відсотках.

Результати прогнозу агенту продуктивності

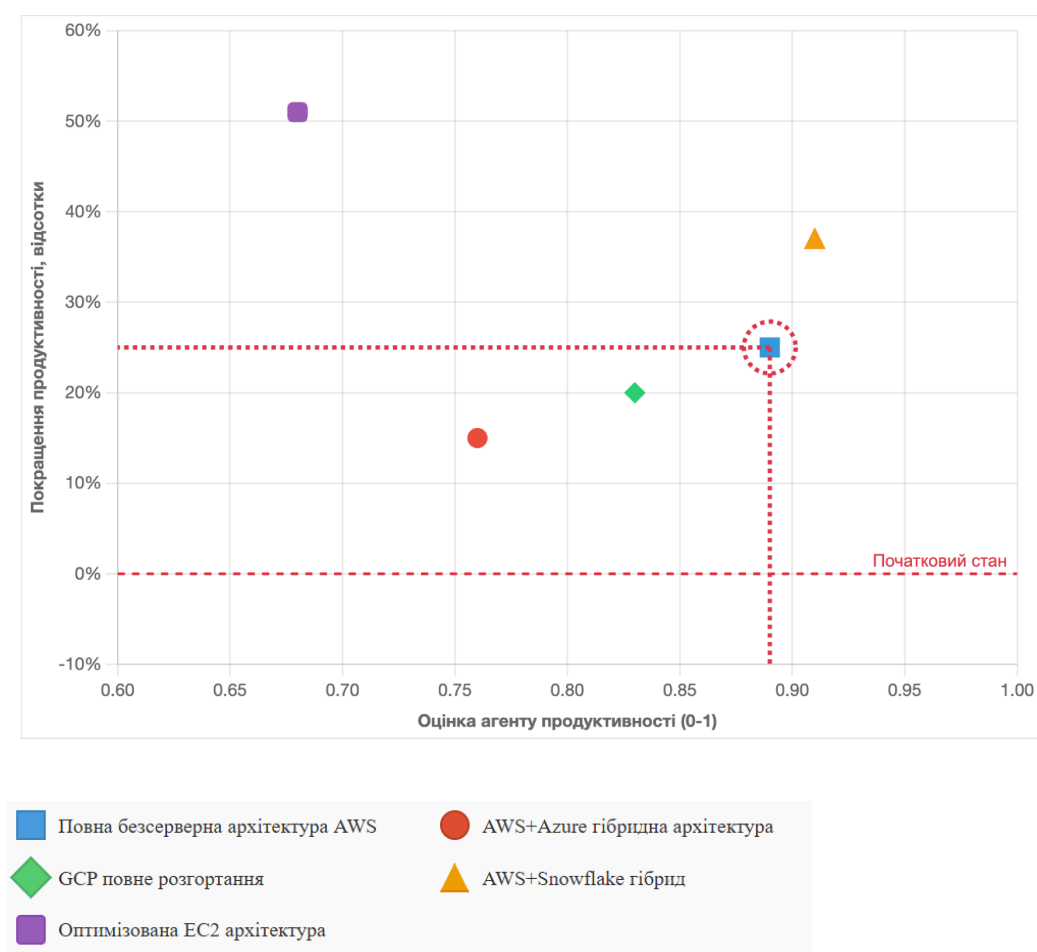


Рисунок 5.4 Результати прогнозу агента продуктивності

Кореляція $r = 0.82$ підтверджує здатність системи до точного прогнозування технічних покращень. Варто зазначити, що рішення з покращенням продуктивності понад 25% (AWS+Snowflake гібрид – 37%, Оптимізована EC2 архітектура – 51%)

були відкинуті системою через несумісність з економічними та безпековими критеріями, що демонструє багатокритеріальний характер прийняття рішень. Обране системою рішення (позначене червоним обведенням) забезпечує оптимальний баланс між економічною ефективністю (47% економії) та покращенням продуктивності (25%), підтверджуючи ефективність інтегрованого показника ефективності, описаного формулою (4.1). Проекції обраного рішення на координатні осі наочно демонструють досягнення заявлених у дисертації результатів. Високі коефіцієнти кореляції (0.89 для економічного агента та 0.82 для агента продуктивності) свідчать про адекватність розробленої математичної моделі та підтверджують практичну цінність запропонованого підходу для автоматизованого проєктування мультимарної інфраструктури.

Порівняння розробленої системи автоматизованого проєктування мультимарної платформи з існуючими підходами до проєктування мультимарної інфраструктури показало її перевагу за всіма ключовими показниками, особливо у складних сценаріях з динамічно змінюваним набором вимог та жорсткими вимогами до безпеки.

5.5.2 Виявлені обмеження та шляхи їх подолання

У процесі експериментів та практичного впровадження були виявлені наступні обмеження розробленої системи:

1. Обмеження, пов'язані з використанням LLM:

- Залежність від якості та актуальності даних, на яких тренувалася модель.
- Потенційні «галюцинації» при обробці спеціалізованих технічних запитів.
- Обмеження контекстного вікна, що ускладнює аналіз дуже складних конфігурацій.

Шляхи подолання: впровадження механізмів верифікації відповідей, використання техніки RAG для доповнення знань моделі актуальною інформацією, оптимізація представлення даних для ефективного використання контекстного вікна.

2. Обмеження масштабованості:

- Зниження продуктивності при аналізі дуже великих інфраструктур (сотні сервісів).
- Зростання часу відгуку при одночасній роботі багатьох користувачів.

Шляхи подолання: оптимізація алгоритмів взаємодії між агентами, впровадження механізмів кешування результатів аналізу, використання розподіленої архітектури для обробки запитів. Застосування механізмів динамічної адаптації, описаних формулами (4.28) – (4.34), дозволить системі ефективно адаптуватися до змін вимог та параметрів середовища.

3. Обмеження щодо взаємодії з API хмарних провайдерів:

- Залежність від доступності та стабільності API.
- Обмеження на кількість запитів до API (rate limiting).
- Потреба в постійному оновленні адаптерів при зміні API.

Шляхи подолання: впровадження механізмів буферизації та повторних спроб, використання локальних кешів для зменшення кількості запитів, розробка більш гнучких адаптерів на основі декларативних специфікацій API.

4. Обмеження щодо безпеки:

- Необхідність надання системі доступу до облікових даних хмарних провайдерів.
- Потенційні ризики витоку чутливої інформації через LLM.

Шляхи подолання: впровадження суворої системи управління доступом, використання безпечних сховищ для облікових даних, фільтрація чутливої інформації перед передачею в LLM.

5. Обмеження щодо інтеграції з існуючими процесами:

- Складність інтеграції з корпоративними системами управління інфраструктурою.
- Необхідність адаптації до різних методологій DevOps.

Шляхи подолання: розробка більш гнучких інтерфейсів інтеграції, впровадження стандартизованих протоколів обміну даними, створення плагінів для популярних CI/CD-систем.

5.5.3 Рекомендації щодо масштабування та вдосконалення системи

На основі аналізу результатів експериментів та практичного впровадження сформульовані наступні рекомендації щодо подальшого розвитку системи автоматизованого проєктування мультихмарної платформи:

1. Розширення функціональності:

- Додавання підтримки більшої кількості хмарних провайдерів.
- Інтеграція з інструментами IaC (Terraform, Pulumi, CloudFormation тощо).
- Впровадження механізмів безперервної оптимізації інфраструктури на основі даних моніторингу.

2. Вдосконалення алгоритмів:

- Дослідження та впровадження більш ефективних алгоритмів багатокритеріальної оптимізації.
- Покращення механізмів координації агентів для забезпечення більш ефективного консенсусу, відповідно до Алгоритму 2.
- Впровадження механізмів активного навчання для покращення точності рекомендацій з часом.

3. Покращення взаємодії з користувачем:

- Впровадження інтерактивних візуалізацій для покращення розуміння рекомендацій.

- Розширення можливостей генерації пояснень та обґрунтувань.
- Створення персоналізованих профілів користувачів для врахування їхніх пріоритетів.

4. Оптимізація продуктивності:

- Впровадження методів розподіленого обчислення для підвищення масштабованості системи.
- Оптимізація використання LLM через механізми кешування та компресії запитів.
- Розробка оптимізованих версій агентів для роботи з обмеженими обчислювальними ресурсами.
- Впровадження механізмів пріоритизації запитів для забезпечення швидкої відповіді на критичні запити.

5. Інтеграція з екосистемою:

- Розробка API для інтеграції з іншими системами управління інфраструктурою.
- Інтеграція з системами управління витратами на хмарні ресурси.
- Розробка механізмів для обміну досвідом та кращими практиками між різними інсталяціями системи.

Ці рекомендації формують стратегічний план розвитку системи, спрямований на подолання виявлених обмежень та розширення її практичного застосування. Особлива увага приділяється покращенню механізмів інтеграції з існуючими процесами та інструментами, що є критичним фактором для широкого впровадження в промисловому середовищі.

ВИСНОВКИ

1. Дослідження існуючих підходів до проєктування мультихмарної інфраструктури виявило суттєві обмеження, пов'язані з відсутністю інтелектуальних методів автоматизованого прийняття рішень та врахування динамічно змінюваного набору критеріїв одночасно.

2. Запропонований комплексний метод динамічного формування обчислювальної інфраструктури, що поєднує мультиагентну архітектуру, великі мовні моделі та методи багатокритеріальної оптимізації, забезпечує суттєве підвищення ефективності проєктування мультихмарних рішень.

3. Експериментальне дослідження на типових сценаріях розгортання web-застосунків з REST API та ETL-систем для обробки даних підтвердило ефективність розробленого підходу, демонструючи зниження вартості, підвищення продуктивності та покращення масштабованості порівняно з традиційними підходами.

4. Практичне впровадження системи автоматизованого проєктування мультихмарної платформи в промисловому середовищі продемонструвало значні переваги запропонованого підходу, включаючи зниження загальних витрат на 47.5% та покращення продуктивності на 25.2%.

5. Система продемонструвала високу адаптивність до різних сценаріїв використання та здатність враховувати динамічно змінюваний набір вимог користувачів, що підтверджує ефективність використання LLM для задач автоматизованого проєктування мультихмарної платформи.

ЗАГАЛЬНІ ВИСНОВКИ

У дисертаційній роботі вирішено актуальну наукову задачу підвищення рівня автоматизації проєктування та переналаштування віртуальної мультимарної платформи шляхом розробки комплексного методу формування обчислювальної інфраструктури у гетерогенному мультимарному середовищі. Основні результати роботи полягають у наступному:

1. Проведено аналіз сучасних підходів до проєктування мультимарних систем та визначено основні проблеми, пов'язані з невідповідністю інтерфейсів та сервісів різних провайдерів, недостатнім урахуванням динамічно змінюваних багатокритеріальних факторів та відсутністю ефективних механізмів динамічної адаптації до змін навантаження та вимог щодо його обробки.

2. Розроблено математичну модель інтегрованого показника ефективності виконання обчислювальних завдань у динамічно змінюваному середовищі, яка дозволяє врахувати економічну ефективність, продуктивність, надійність, сумісність взаємодії та безпеку при автоматизованому проєктуванні оптимальної конфігурації мультимарної інфраструктури.

3. Запропоновано комплексний метод формування обчислювальної інфраструктури у гетерогенному мультимарному середовищі, який використовує інтегрований показник ефективності виконання обчислювальних завдань для визначення в автоматизованому режимі місця розташування і типу сервісів, які пропонуються хмарними провайдерами, поєднує переваги методів навчання з підкріпленням і багатокритеріальних еволюційних алгоритмів для пошуку ефективних рішень із урахуванням динаміки зміни як стану інфраструктури мультимарного середовища так і вимог кінцевого користувача, що дозволяє підвищити продуктивність, захищеність та знизити витрати виконання обчислювальних задач.

4. Запропоновано метод визначення складових інтегрованого показника ефективності виконання обчислювальних завдань у динамічно змінюваному мультихмарному середовищі, який відрізняється використанням LLM моделі для аналізу стану обчислювального середовища за сукупністю критеріїв проєктування віртуальної мультихмарної платформи із динамічно змінюваним набором вимог, що дозволяє враховувати ступінь впливу кожного критерія та підвищити достовірність і об'єктивність прийняття рішень.

5. Розроблено мультиагентну систему, яка складається з технічного, економічного та безпекового агентів, а також агента-координатора для узгодження їх висновків та формування кінцевого рішення щодо автоматизованого визначення поточної архітектури віртуальної мультихмарної платформи в умовах динамічної зміни набору вимог до неї з урахуванням наявних параметрів середовища та випадкового характеру надходження обчислювального навантаження на обробку, що дозволяє автоматизувати процес проєктування інфраструктури та підвищити продуктивність обробки навантаження.

6. Розроблено програмний компонент динамічної адаптації до зміни параметрів середовища, що забезпечує постійний моніторинг стану системи, виявлення відхилень, прогнозування змін та вибір стратегії адаптації для підтримки ефективного проєктування інфраструктури під виконання обчислювальних завдань.

7. Розроблено програмне забезпечення прототипу системи автоматизованого проєктування мультихмарної платформи на базі Python, LangChain, та мультиагентної архітектури LLM, дослідна експлуатація якого демонструє ефективність запропонованого підходу в реальних умовах.

8. Проведено експериментальну перевірку ефективності запропонованого підходу на типових сценаріях використання на прикладі проєктування мультихмарної інфраструктури для web-застосунку з REST API та ETL-системи

обробки даних, яка підтвердила значні переваги розробленого рішення порівняно з традиційними підходами.

9. Здійснено практичне впровадження системи автоматизованого проєктування віртуальної мультимарної платформи із динамічно змінюваним набором вимог, що продемонструвало значне зниження витрат (в середньому на 47.5%), покращення продуктивності обробки обчислювальних завдань (на 25.2%) та підвищення масштабованості (на 31.8%) порівняно з традиційними підходами.

10. Визначено перспективні напрямки подальшого розвитку системи, включаючи розширення функціональності, вдосконалення алгоритмів, покращення взаємодії з користувачем, оптимізацію продуктивності та інтеграцію з екосистемою.

Результати дисертаційної роботи мають значну практичну цінність для підприємств та організацій, що використовують або планують використовувати мультимарні середовища для своїх обчислювальних потреб. Запропонований комплексний метод формування обчислювальної інфраструктури у гетерогенному мультимарному середовищі дозволяє суттєво підвищити ефективність використання сервісів мультимарної платформи, знизити витрати та поліпшити продуктивність обчислювальних процесів при збереженні високого рівня безпеки та надійності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Shrimal, G., & Sandeep. (2022). Using heterogeneous cloud computing to manage resources in sustainable cyber-physical systems. *2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART)*, Moradabad, India, 153–157. <https://doi.org/10.1109/SMART55829.2022.10047642>.
2. Прокопєць, Н. А. (2022). *Енергоєфективне обслуговування навантаження інформаційно-комунікаційної мережі* (Докторська дисертація). Національний технічний університет України «КПІ ім. Ігоря Сікорського». Retrieved from <https://ela.kpi.ua/items/dda7c4d0-3599-4b7e-9fd9-fc0f9c497e93>.
3. Marquez, J., Mondragon, O. H., & Gonzalez, J. D. (2021). An Intelligent Approach to Resource Allocation on Heterogeneous Cloud Infrastructures. *Applied Sciences*, 11(21), 9940. <https://doi.org/10.3390/app11219940>.
4. Swain, S. R., Singh, A. K., & Lee, C. N. (2022). Efficient resource management in cloud environment [Preprint]. *arXiv*. <https://doi.org/10.48550/arXiv.2207.12085>.
5. Jeyaraj, R., Balasubramaniam, A., Kumara, A. M. A., Guizani, N., & Paul, A. (2023). Resource management in cloud and cloud-influenced technologies for Internet of Things applications. *ACM Computing Surveys*, 55(12), Article 242, 37 pages. <https://doi.org/10.1145/3571729>.
6. Buyya, R., & Son, J. (2018). Software-defined multi-cloud computing: A vision, architectural elements, and future directions [Preprint]. *arXiv*. <https://doi.org/10.48550/arXiv.1805.10780>.
7. Caceres, A., & Globa, L. (2025). AHP-based multi-criteria analysis of multi-cloud data management techniques. *Radioelectronic and Computer Systems*, 1, Article 04. <https://doi.org/10.32620/reks.2025.1.04>.
8. Caceres, A., & Globa, L. (2025). Аналіз підходів доступу до даних у мульти-клауд середовищі [Analysis of data access approaches in a multi-cloud environment]. *Radio Electronics, Computer Science, Control*, 1. Retrieved from

<https://ric.zp.edu.ua/article/view/324539/314660>.

9. Caceres, A., & Globa, L. (2024). Підхід до формування мультимарного середовища на основі багатокритеріального аналізу та онтологічного моделювання [A multi-criteria and ontology-based approach to multi-cloud environment selection]. *Системи управління, навігації та зв'язку*, 4. <https://doi.org/10.26906/SUNZ.2024.4.084>.

10. Caceres, A., & Globa, L. (2022). State-of-the-art architectures for interoperability of heterogeneous clouds. In *Proceedings of the 2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)* (pp. 704–709). IEEE. <https://doi.org/10.1109/TCSET55632.2022.9766965>.

11. Caceres, A., & Globa, L. (2024). Інтеграція ШІ та мультиагентних систем для багатокритеріального аналізу у мультимарному середовищі [Integration of AI and multi-agent systems for multi-criteria analysis in a multicloud environment]. *Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки*, 35(74), № 6, Частина 2. Retrieved from https://www.tech.vernadskyjournals.in.ua/journals/2024/6_2024/part_2/12.pdf.

12. Oracle. (2023). *Multicloud is the new norm*. Oracle Corporation. <https://www.oracle.com/pl/cloud/multicloud/mainstream/>.

13. Saxena, D., Gupta, R., & Singh, A. K. (2021). A survey and comparative study on multi-cloud architectures: Emerging issues and challenges for cloud federation [Preprint]. *arXiv*. <https://doi.org/10.48550/arXiv.2108.12831>.

14. Flexera. *State of the cloud report*. <https://info.flexera.com/CM-REPORT-State-of-the-Cloud>.

15. Google Cloud. *Types of cloud computing*. <https://cloud.google.com/discover/types-of-cloud-computing?hl=uk>.

16. Federal Chief Information Officers Council. (2021). *Multi-cloud and hybrid*

cloud guide (Version 4). https://www.cio.gov/assets/resources/Multi-Cloud%20and%20Hybrid%20Cloud%20Guide_v4_Final.pdf.

17. Imran, H. A., Latif, U., Ikram, A., Ehsan, M., Ikram, A., Khan, W., & Wazir, S. (2020). Multi-cloud: A comprehensive review. In *2020 IEEE 23rd International Multitopic Conference (INMIC)* (pp. 1–5). IEEE. <https://doi.org/10.1109/INMIC50486.2020.9318176>.

18. Leitner, P., & Cito, J. (2016). Patterns in the Chaos: Performance variation in public clouds. *ACM TOIT*.

19. Han, R., et al. (2015). Cloud service selection: State-of-the-art and future directions. *JNCA*.

20. Baresi, L., et al. (2021). Self-Adaptive Systems in the Cloud. *ACM Computing Surveys*.

21. Raj, P., Raman, A. (2018). Multi-cloud Management: Technologies, Tools, and Techniques. In: *Software-Defined Cloud Centers*. Computer Communications and Networks. Springer, Cham. https://doi.org/10.1007/978-3-319-78637-7_10.

22. Shukla, P. R., & Patil, V. M. (2023). A comprehensive review of frameworks for achieving interoperability in multi-cloud environments. In *2023 Second International Conference on Informatics (ICI)* (pp. 1–6). IEEE. <https://doi.org/10.1109/ICI60088.2023.10421703>.

23. Dubey, M., & Singh, K. (2024). Multi-Cloud Management Strategies - A Comprehensive Review. *Resmilitaris*. DOI:10.48047/resmil.v9i1.28.

24. Kubernetes. *Kubernetes components overview*. <https://kubernetes.io/docs/concepts/overview/components/>

25. HashiCorp. *Terraform docs overview*. <https://developer.hashicorp.com/terraform/docs>.

26. Oxyno-zeta. *S3 Proxy documentation*. <https://oxyno-zeta.github.io/s3-proxy/>.

27. Yousry, A. *Data management: A guide to Apache NiFi*. Medium. <https://medium.com/@ansam.yousry/data-management-a-guide-to-apache-nifi-21a29ecc4591>.
28. Apache Software Foundation. *Apache Libcloud*. from <https://libcloud.apache.org/>.
29. Varghese, B., & Buyya, R. (2018). Next-generation cloud computing: New trends and research directions. *Future Generation Computer Systems*, 79(Part 3), 849–861. <https://doi.org/10.1016/j.future.2017.09.020>.
30. Schrama, A. (2021). *Managing multi-cloud systems: A framework for effective management and governance of multivendor cloud arrangements* (Master's thesis). Delft University of Technology.
31. Taherkordi, A., Zahid, F., Verginadis, Y., & Horn, G. (2018). Future cloud systems design: Challenges and research directions. *IEEE Access*, 6, 74120–74150. <https://doi.org/10.1109/ACCESS.2018.2883149>.
32. Mansour, I., Sahandi, R., & others. (2016). Interoperability in the heterogeneous cloud environment: A survey of recent user-centric approaches. In *ICC '16: Proceedings of the International Conference on Internet of Things and Cloud Computing* (pp. 1–7). Association for Computing Machinery. <https://doi.org/10.1145/2896387.2896447>.
33. Hardik, J. (2025). *Implementing secure multi-cloud connectivity: Tools and techniques*. Medium. <https://hardiks.medium.com/implementing-secure-multi-cloud-connectivity-tools-and-techniques-0a78ac4e3070>.
34. Google Cloud. *Hybrid and multi-cloud network architecture*. Retrieved February 18, 2025, from <https://cloud.google.com/architecture/network-hybrid-multicloud>.
35. Alkira. *Multi-cloud network connectivity*. Retrieved February 18, 2025, from <https://www.alkira.com/resources/multi-cloud-network-connectivity/>.
36. Fitzgerald, A. *Serverless functions: Best practices for serverless functions*. HubSpot. <https://blog.hubspot.com/website/serverless-functions#best-practices-for->

[serverless-functions.](#)

37. F5 Networks. *Multi-cloud strategies*. Retrieved February 10, 2025, from [https://www.f5.com/glossary/multi-cloud-strategies.](https://www.f5.com/glossary/multi-cloud-strategies)

38. DigitalOcean. (n.d.). *Multi-cloud strategy*. Retrieved February 2, 2025, from [https://www.digitalocean.com/resources/articles/multi-cloud-strategy.](https://www.digitalocean.com/resources/articles/multi-cloud-strategy)

39. Lightfoot, L. (2024). *The top 7 cloud storage gateway solutions*. [https://expertinsights.com/insights/top-cloud-storage-gateway-solutions/.](https://expertinsights.com/insights/top-cloud-storage-gateway-solutions/)

40. Oracle. *What is a data management platform (DMP)* [https://www.oracle.com/cx/marketing/data-management-platform/what-is-dmp/#how-to-choose-the-right-dmp.](https://www.oracle.com/cx/marketing/data-management-platform/what-is-dmp/#how-to-choose-the-right-dmp)

41. Chen, A., Kathika, J., & Ngo, K. (2021). *Multi-cloud API approach to prevent vendor lock-in* (Course project, Santa Clara University). https://www.cse.scu.edu/~mlwang/projects/Cloud_preventVendorLockIn_21w.pdf

42. Sheldon, R. *Cloud data management interface (CDMI)*. TechTarget. [https://www.techtarget.com/searchstorage/definition/Cloud-Data-Management-Interface.](https://www.techtarget.com/searchstorage/definition/Cloud-Data-Management-Interface)

43. Analytics Steps. *What is Open Cloud Computing Interface (OCCI)?* [https://www.analyticssteps.com/blogs/what-open-cloud-computing-interface-occi.](https://www.analyticssteps.com/blogs/what-open-cloud-computing-interface-occi)

44. Shua, A. (2025). *Multi-cloud environments: How secure are they?* Forbes Technology Council. [https://www.forbes.com/councils/forbestechcouncil/2025/01/27/multi-cloud-environments-how-secure-are-they/.](https://www.forbes.com/councils/forbestechcouncil/2025/01/27/multi-cloud-environments-how-secure-are-they/)

45. Nutanix. (2024). *What is multicloud?* [https://www.nutanix.com/info/multi-cloud-environment#definition.](https://www.nutanix.com/info/multi-cloud-environment#definition)

46. Crossplane. *Issue #1033: Implement integration testing across Crossplane ecosystem*. GitHub. Retrieved February 20, 2025, from [https://github.com/crossplane/crossplane/issues/1033.](https://github.com/crossplane/crossplane/issues/1033)

47. Jain, A. (2022). *Crossplane implementation & integration with AWS*. Medium. <https://medium.com/expedia-group-tech/crossplane-implementation-integration-with-aws-16f6143644d4>.
48. Nordcloud. *Home*. <https://nordcloud.com>.
49. Crayon. *Home*. <https://www.crayon.com>.
50. Tech-5. *Home*. <https://tech-5.de>.
51. Petcu, D. (2013). Multi-cloud: expectations and current approaches. *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*. (pp. 1–6). ACM. <https://doi.org/10.1145/2462326.2462328>.
52. Li, X., Li, C., Jiang, C., & Liu, H. (2021). Multi-cloud resource management: Concepts, taxonomy, and challenges. *Future Generation Computer Systems*, 115, 1–15.
53. Ghobaei-Arani, M., Souri, A., & Rahmanian, A. A. (2020). Resource management approaches in fog computing: A comprehensive review. *Journal of Grid Computing*, 18(1), 1–42. <https://doi.org/10.1007/s10723-019-09491-1>.
54. Zhou, Z., et al. (2018). Resource allocation in multi-cloud computing using constraint satisfaction models. *Future Generation Computer Systems*, 78, 134–145. <https://doi.org/10.1016/j.future.2017.08.005>.
55. Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., Casallas, R., & Gil, S. (2015). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. 2015 10th Computing Colombian Conference (10CCC), 583–590. <https://doi.org/10.1109/ColumbianCC.2015.7333476>.
56. Toosi, A. N., Calheiros, R. N., & Buyya, R. (2014). Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Computing Surveys*, 47(1), 7. <https://doi.org/10.1145/2593512>.
57. Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., & Morrow, M. (2009). Blueprint for the Intercloud – Protocols and formats for cloud computing interoperability. 2009 Fourth International Conference on Internet and Web Applications and Services,

328–336. <https://doi.org/10.1109/ICIW.2009.55>.

58. As Pahl, C., & Xiong, H. (2013). Migration to cloud computing: A taxonomy of cloud migration patterns. 2013 IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, 6 –14. <https://doi.org/10.1109/MESOCA.2013.6649005>.

59. Cao, J., Li, K., Xu, L., & Li, K. (2021). Multi-objective optimization in cloud computing: A comprehensive survey. ACM Computing Surveys (CSUR), 54(3), 1–36. <https://doi.org/10.1145/3439815>.

60. Zettler, K. *An overview of distributed systems and microservices architectures*. Atlassian. Retrieved February 22, 2025, from <https://www.atlassian.com/microservices/microservices-architecture/distributed-architecture>.

61. Dar, F. Y. (2024). *Dynamic workload management and optimization in multicloud*. Huawei Blog. <https://blog.huawei.com/en/post/2024/3/25/dynamic-workload-management-optimization-multicloud>.

62. Passwater, A. (2017). *The state of serverless and multi-cloud*. Serverless. <https://www.serverless.com/blog/state-of-serverless-multi-cloud>.

63. Zhao, H., Benomar, Z., Pfandzelter, T., & Georgantas, N. (2022). Supporting multi-cloud in serverless computing. In *Proceedings of the 15th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'22)* (pp.285-290). IEEE. <https://doi.org/10.1109/UCC56403.2022.00051>.

64. Akthar, A. (2024). *Anthos vs. OpenShift: Container management comparison*. Royal Cyber. <https://www.royalcyber.com/blogs/cloud/anthos-vs-openshift-container-management/>.

65. Amazon Web Services (AWS). *What is cloud bursting?* <https://aws.amazon.com/what-is/cloud-bursting/>.

66. Multiplayer. *Distributed systems architecture: Tutorial & best practices*.

Retrieved February 26, 2025, from <https://www.multiplayer.app/distributed-systems-architecture/>.

67. Khole, A., Thakar, A., Kulkarni, A., Jadhav, H., Shende, S., & Karajkhede, V. (2023). *A compendium on distributed systems* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2302.03990>.

68. Khan, R. Z. (2015). Distributed computing: An overview. *International Journal of Advanced Networking and Applications*, 7, 2630–2635.

69. GeeksforGeeks. (2024). *Resource management in distributed system*. <https://www.geeksforgeeks.org/resource-management-in-distributed-system/>.

70. Buyya, R., Srirama, S. N., Casale, G., Calheiros, R. N., Villari, M., & Brogi, A. (2017). *A manifesto for future generation cloud computing: Research directions for the next decade* [Preprint]. arXiv. <https://arxiv.org/abs/1711.09123>.

71. Hong, J., Dreibholz, T., Schenkel, J. A., & Hu, J. A. (2019). An overview of multi-cloud computing. In L. Barolli, M. Takizawa, F. Xhafa, & T. Enokido (Eds.), *Web, artificial intelligence and network applications. WAINA 2019* (Advances in Intelligent Systems and Computing, Vol. 927). Springer, Cham. https://doi.org/10.1007/978-3-030-15035-8_103.

72. Aruldoss, M., Lakshmi, T. M., & Venkatesan, V. P. (2013). A survey on multi-criteria decision-making methods and its applications. *American Journal of Information Systems*, 1(1), 31–43. <https://doi.org/10.12691/ajis-1-1-5>.

73. Корендович, В. С. (2017). Застосування багатокритеріального аналізу для пріоритетного вибор. *Центр воєнно-стратегічних досліджень Національного університету оборони України імені Івана Черняхівського, Україна*, 2(60), 129–136. <https://doi.org/10.33099/2304-2745/2017-2-60/129-136>.

74. Buriachenko, A., & Kuts, N. (2021). *Multi-criterion analysis methods*. ResearchGate. Retrieved February 20, 2025, from https://www.researchgate.net/publication/348774582_MULTI-

CRITERION ANALYSIS METHODS.

75. Linkov, I., & Moberg, E. (2012). *Multi-criteria decision analysis: Environmental applications and case studies* (1st ed.). CRC Press.

76. Saaty, T. L. (1980). *The Analytic Hierarchy Process*. McGraw-Hill.

77. Haiko, S., & Prykhodniuk, V. (2020). Representation of educational resources as interactive documents. In *Proceedings of the XII International Scientific-Practical Conference "Internet-Education-Science" (IES-2020)* (pp. 249–251). Vinnytsia National Technical University (VNTU).

78. Pandey, A., Calyam, P., Lyu, Z., & Joshi, T. (2021). Fuzzy-engineered multi-cloud resource brokering for data-intensive applications. In *Proceedings of the 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)* (pp. 257–266). IEEE. <https://doi.org/10.1109/CCGrid51090.2021.00035>.

79. Dwyer, J. (2023). *Multi-cloud machine learning strategies for your AI*. Zeet. <https://zeet.co/blog/the-multi-cloud-ml-advantage-why-your-ai-needs-a-multi-cloud-approach>.

80. Arabnejad, H., Pahl, C., Jamshidi, P., & Estrada, G. (2017). *A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.1705.07114>.

81. Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., & Mian, A. (2023). *A comprehensive overview of large language models* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2307.06435>.

82. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)* (pp. 6000–6010). Curran Associates Inc.

83. Примостка, А. О. (2017). Мультиагентний підхід до проєктування інформаційно-аналітичних систем. *Бизнес Інформ*, 1(468), 274–280.

84. Dorri, A., Kanhere, S. S., & Jurdak, R. (2018). Multi-agent systems: A survey. *IEEE Access*, 6, 28573–28593. <https://doi.org/10.1109/ACCESS.2018.2831228>.
85. Julian, V., & Botti, V. (2019). Multi-Agent Systems. *Applied Sciences*, 9(7), 1402. <https://doi.org/10.3390/app9071402>.
86. Sun, L., Dong, H., Hussain, F. K., Hussain, O. K., & Chang, E. (2014). Cloud service selection: State-of-the-art and future research directions. *Journal of Network and Computer Applications*, 45, 134–150. <https://doi.org/10.1016/j.jnca.2014.07.019>.
87. Kritikos, K., & Plexousakis, D. (2006). Semantic QoS metric matching. *Proceedings of the IEEE International Conference on Web Services (ICWS)*, 445–452. <https://doi.org/10.1109/ICWS.2006.56>.
88. Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., & Kagal, L. (2018). Explaining explanations: An overview of interpretability of machine learning. 2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA), 80–89. <https://doi.org/10.1109/DSAA.2018.00018>.
89. Bommasani, R., et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*. <https://doi.org/10.48550/arXiv.2108.07258>.
90. OpenAI. (2023). *GPT-4 Technical Report*. arXiv:2303.08774.
91. Zhang, Y., et al. (2023). A survey on trustworthy LLMs: Robustness, fairness, and explainability. *arXiv preprint arXiv:2309.07864*. <https://doi.org/10.48550/arXiv.2309.07864>.
92. Chatterjee, S., et al. (2022). Building explainable multi-agent systems with LLMs. *arXiv preprint arXiv:2209.12037*. <https://doi.org/10.48550/arXiv.2209.12037>.
93. Sivaraman, A., et al. (2023). Agents that ask questions: Foundations for using language models for goal-oriented behavior. *Advances in Neural Information Processing Systems (NeurIPS)*. <https://doi.org/10.48550/arXiv.2309.07864>.

Додаток 1. Приклади реалізації агентів

Ключові компоненти реалізації технічного агента

Технічний агент фокусується на аналізі технічних аспектів проектування мультимарної інфраструктури та оцінці різних варіантів з точки зору продуктивності, масштабованості та сумісності.

```
class TechnicalAgent:
    def __init__(self, llm_model, vector_store, cloud_service_db):
        self.llm = llm_model

        self.vector_store = vector_store

        self.cloud_service_db = cloud_service_db

        self.instruction_template =
self._load_instruction_template("technical_agent_instructions.txt
")

    def analyze_technical_requirements(self, requirements):
        # Аналіз технічних вимог до системи

        context = self._retrieve_relevant_context(requirements)

        instruction =
self.instruction_template.format(requirements=requirements,
context=context)

        analysis = self.llm.generate(instruction)

        return self._structure_analysis(analysis)

    def evaluate_deployment_options(self, options, requirements):
        # Оцінка варіантів розгортання

        results = []

        for option in options:
            evaluation = self._evaluate_single_option(option,
requirements)

            results.append(evaluation)

        return results
```

```

def _retrieve_relevant_context(self, requirements):
    # Пошук релевантної інформації у векторному сховищі
    query_embedding = self.llm.get_embedding(requirements)

    relevant_docs = self.vector_store.similarity_search(query_embedding)

    return "\n".join([doc.page_content for doc in relevant_docs])

def _evaluate_single_option(self, option, requirements):
    # Оцінка одного варіанту розгортання

    service_details = self.cloud_service_db.get_service_details(option['provider'],
option['services'])

    instruction = self._create_evaluation_instruction(option,
requirements, service_details)

    evaluation = self.llm.generate(instruction)

    return self._parse_evaluation_response(evaluation,
option)

```

Приклад структури шаблону інструкцій для технічного агента

You are a technical expert in cloud computing, analyzing requirements for multicloud infrastructure deployment.

Your task is to evaluate the technical aspects of the proposed deployment option.

TECHNICAL REQUIREMENTS:`{requirements}`**DEPLOYMENT OPTION:**`{deployment_option}`**SERVICE DETAILS:**`{service_details}`

Analyze the following aspects:

1. Performance - whether the proposed option meets performance requirements
2. Scalability - whether it provides capabilities for horizontal and vertical scaling
3. Compatibility - whether there are compatibility issues between the selected services
4. Reliability - evaluate potential points of failure and fault tolerance strategies

Structure your response in the following JSON format:

```
{
  "performance_score": [score from 0 to 1],
  "scalability_score": [score from 0 to 1],
  "compatibility_score": [score from 0 to 1],
  "reliability_score": [score from 0 to 1],
  "overall_technical_score": [overall score from 0 to 1],
  "recommendations": [list of technical recommendations],
  "reasoning": [explanation of your evaluation]
}
```

Реалізація економічного агента

Економічний агент відповідає за аналіз економічної ефективності різних варіантів розгортання, включаючи оцінку вартості ресурсів, оптимізацію витрат та довгострокове планування.

```
class EconomicAgent:
    def __init__(self, llm_model, price_database, usage_estimator):
        self.llm = llm_model
        self.price_db = price_database
        self.usage_estimator = usage_estimator
        self.instruction_template =
self._load_instruction_template("economic_agent_instructions.txt")

    def analyze_cost_effectiveness(self, deployment_option,
requirements):
        # Аналіз економічної ефективності варіанту розгортання
        estimated_usage = self.usage_estimator.estimate(requirements)
        price_data = self._get_price_data(deployment_option)
        cost_analysis = self._calculate_costs(deployment_option,
estimated_usage, price_data)
        return cost_analysis

    def generate_cost_optimization_recommendations(self,
deployment_option, cost_analysis):
        # Генерація рекомендацій щодо оптимізації витрат
        instruction =
self._create_optimization_instruction(deployment_option,
cost_analysis)
        recommendations = self.llm.generate(instruction)
        return self._parse_recommendations(recommendations)

    def _get_price_data(self, deployment_option):
        # Отримання даних про ціни для обраних сервісів
```



```

price_data = {}
for service in deployment_option['services']:
    price_info = self.price_db.get_pricing(
        deployment_option['provider'],
        service['type'],
        service['region'],
        service['tier']
    )
    price_data[service['id']] = price_info
return price_data

def _calculate_costs(self, deployment_option, estimated_usage,
price_data):
    # Розрахунок вартості на основі оцінки використання та даних
    про ціни
    total_cost = 0
    cost_breakdown = {}
    for service in deployment_option['services']:
        service_id = service['id']
        usage = estimated_usage.get(service_id, {})
        price = price_data.get(service_id, {})

        service_cost = self._calculate_service_cost(usage, price)
        cost_breakdown[service_id] = service_cost
        total_cost += service_cost['total']
    return {
        'total_monthly_cost': total_cost,
        'cost_breakdown': cost_breakdown
    }

```

Реалізація безпекового агента

Безпековий агент аналізує аспекти безпеки та відповідності нормативним вимогам для різних варіантів розгортання мультихмарної інфраструктури.

```
class SecurityAgent:

    def __init__(self, llm_model, security_knowledge_base,
compliance_database):

        self.llm = llm_model

        self.security_kb = security_knowledge_base

        self.compliance_db = compliance_database

        self.instruction_template =
self._load_instruction_template("security_agent_instructions.txt"
)

    def analyze_security_requirements(self, requirements):

        # Аналіз вимог до безпеки

        extracted_requirements =
self._extract_security_requirements(requirements)

        security_context =
self._get_security_context(extracted_requirements)

        return {

            'security_requirements': extracted_requirements,

            'security_context': security_context

        }

    def evaluate_security_aspects(self, deployment_option,
security_requirements):

        # Оцінка аспектів безпеки для варіанту розгортання

        security_features =
self._get_security_features(deployment_option)

        compliance_info =
self._get_compliance_info(deployment_option)

        instruction =
self._create_security_evaluation_instruction(
```

```

        deployment_option,
        security_requirements,
        security_features,
        compliance_info
    )

    evaluation = self.llm.generate(instruction)
    return self._parse_security_evaluation(evaluation)

def _extract_security_requirements(self, requirements):
    # Вилучення вимог до безпеки з загальних вимог
    instruction = f"""
    Analyze the following system requirements and extract all
    aspects related to security and regulatory compliance:

    {requirements}

    Return the result in JSON format with fields:
    - data_protection
    - authentication
    - authorization
    - audit
    - compliance
    - privacy
    """
    response = self.llm.generate(instruction)
    return json.loads(response)

def _get_security_features(self, deployment_option):
    # Отримання інформації про функції безпеки для обраних
    сервісів
    security_features = {}

```

```

for service in deployment_option['services']:
    features = self.security_kb.get_security_features(
        deployment_option['provider'],
        service['type']
    )
    security_features[service['id']] = features
return security_features

```

Реалізація агента-координатора

Агент-координатор відповідає за координацію роботи спеціалізованих агентів, розподіл завдань та агрегацію результатів для формування фінального рішення.

```

class CoordinatorAgent:

    def __init__(self, llm_model, technical_agent, economic_agent,
security_agent):

        self.llm = llm_model

        self.technical_agent = technical_agent

        self.economic_agent = economic_agent

        self.security_agent = security_agent

        self.instruction_template =
self._load_instruction_template("coordinator_agent_instructions.t
xt")

    def process_requirements(self, requirements):

        # Обробка вхідних вимог та розподіл завдань між агентами

        parsed_requirements =
self._parse_requirements(requirements)

        # Паралельний аналіз різними агентами

        tech_analysis =
self.technical_agent.analyze_technical_requirements(parsed_requir
ements)

```

```

        security_analysis =
self.security_agent.analyze_security_requirements(parsed_requirements)

        # Генерація початкових варіантів розгортання

        deployment_options =
self._generate_deployment_options(parsed_requirements,
tech_analysis)

        # Оцінка варіантів розгортання всіма агентами

        evaluated_options =
self._evaluate_options(deployment_options, parsed_requirements,
security_analysis)

        # Формування фінального рішення

        final_decision =
self._make_final_decision(evaluated_options, parsed_requirements)

    return final_decision

def _parse_requirements(self, requirements):
    # Парсинг та структурування вхідних вимог
    instruction = f"""
    Analyze the following requirements for multicloud
    infrastructure and structure them by categories:

    {requirements}

    Return the result in JSON format with fields:
    - performance_requirements
    - scalability_requirements
    - security_requirements
    - compliance_requirements

```

```

        - budget_constraints
        - other_requirements
        """

        response = self.llm.generate(instruction)
        return json.loads(response)

    def _generate_deployment_options(self, parsed_requirements,
tech_analysis):

        # Генерація початкових варіантів розгортання

        instruction =
self._create_options_generation_instruction(parsed_requirements,
tech_analysis)

        options_json = self.llm.generate(instruction)
        return json.loads(options_json)

    def _evaluate_options(self, deployment_options,
parsed_requirements, security_analysis):

        # Оцінка варіантів розгортання всіма агентами

        evaluated_options = []

        for option in deployment_options:

            # Технічна оцінка

            tech_evaluation =
self.technical_agent.evaluate_deployment_options([option],
parsed_requirements)[0]

            # Економічна оцінка

            cost_analysis =
self.economic_agent.analyze_cost_effectiveness(option,
parsed_requirements)

            # Оцінка безпеки

            security_evaluation =
self.security_agent.evaluate_security_aspects(option,
security_analysis['security_requirements'])

```

```

        # Агрегація результатів
        option_evaluation = {
            'option': option,
            'technical_evaluation': tech_evaluation,
            'cost_analysis': cost_analysis,
            'security_evaluation': security_evaluation
        }

        evaluated_options.append(option_evaluation)

    return evaluated_options

    def _make_final_decision(self, evaluated_options,
        parsed_requirements):
        # Формування фінального рішення на основі оцінок усіх
        агентів

        instruction =
        self._create_decision_making_instruction(evaluated_options,
        parsed_requirements)

        decision = self.llm.generate(instruction)

    return self._parse_decision(decision, evaluated_options)

```

Додаток 2. Промпт-шаблон для технічного агента

You are a technical expert in cloud computing, analyzing requirements for multicloud infrastructure deployment.

Your task is to evaluate the technical aspects of the proposed deployment option.

TECHNICAL REQUIREMENTS:

{requirements}

DEPLOYMENT OPTION:

{deployment_option}

SERVICE DETAILS:

{service_details}

Analyze the following aspects:

1. Performance - whether the proposed option meets performance requirements
2. Scalability - whether it provides capabilities for horizontal and vertical scaling
3. Compatibility - whether there are compatibility issues between the selected services
4. Reliability - evaluate potential points of failure and fault tolerance strategies

Structure your response in the following JSON format:

```
{  
  "performance_score": [score from 0 to 1],  
  "scalability_score": [score from 0 to 1],  
  "compatibility_score": [score from 0 to 1],  
  "reliability_score": [score from 0 to 1],  
  "overall_technical_score": [overall score from 0 to 1],  
  "recommendations": [list of technical recommendations],  
  "reasoning": [explanation of your evaluation]  
}
```


Додаток 3. Архітектура інтеграції LLM

1. Модуль підготовки запитів – формує запити до LLM на основі вхідних даних та контексту, включаючи інструкції для моделі та необхідні дані.

2. Модуль обробки відповідей – парсить структуровані відповіді LLM, виконує валідацію та перетворює їх у внутрішній формат системи.

3. Модуль керування контекстом – забезпечує збереження та оновлення контексту розмови з LLM, що дозволяє проводити багатокрокові діалоги.

4. Векторна база даних – зберігає ембедінги (векторні представлення) документів та фрагментів тексту для швидкого пошуку релевантної інформації.

В розробленій системі LLM використовується для наступних задач:

1. Аналіз технічних вимог та їх структурування – модель перетворює неструктурований опис вимог користувача у формалізований перелік параметрів.

2. Оцінка варіантів розгортання – агенти на базі LLM аналізують різні аспекти запропонованих конфігурацій та формують оцінки за визначеними критеріями.

3. Генерація пояснень – система формує зрозумілі обґрунтування прийнятих рішень, що підвищує довіру користувачів.

4. Аналіз та усунення конфліктів – при виникненні протиріч між оцінками різних агентів, LLM допомагає знайти компромісне рішення.

Додаток 4. Аналіз вхідних вимог, виконаний агентом-координатором

```

{
  "application_type": "web_application_with_rest_api",
  "technology_stack": {
    "frontend": "React.js",
    "backend": "Django",
    "database": "PostgreSQL",
    "cache": "Redis",
    "storage": "S3-compatible"
  },
  "performance_requirements": {
    "peak_load": 1500,
    "average_load": 300,
    "response_time_ms": 200,
    "availability_percent": 99.95
  },
  "data_characteristics": {
    "database_size_gb": 8,
    "media_size_gb": 120
  },
  "budget_constraint_usd": 3000,
  "geographical_distribution": {
    "europe": 0.7,
    "north_america": 0.2,
    "asia": 0.1
  },
  "security_requirements": ["gdpr_compliance",
"data_encryption"]}

```

Додаток 5. Визначення можливих варіантів розгортання технічним агентом

```
{
  "technical_analysis": {
    "key_requirements": [
      "High computational resources for 1500 req/s peak load",
      "Microservices architecture recommended",
      "PostgreSQL clustering with replication",
      "CDN and caching in multiple regions"
    ],
    "deployment_options": [
      {
        "option": "AWS Full Deployment",
        "components": [
          "Frontend on CloudFront/S3",
          "Backend on EKS",
          "RDS for PostgreSQL",
          "ElastiCache for Redis",
          "S3 for media"
        ],
        "scores": {
          "performance": 0.92,
          "scalability": 0.95,
          "reliability": 0.94,
          "overall": 0.94
        }
      },
      {
        "option": "AWS+Azure Hybrid",
```

```

    "components": [
      "Frontend/CDN on Azure",
      "Backend/DB on AWS"
    ],
    "scores": {
      "performance": 0.88,
      "scalability": 0.91,
      "reliability": 0.89,
      "overall": 0.89
    }
  },
  {
    "option": "GCP Full Deployment",
    "components": [
      "Cloud Run for backend",
      "Cloud SQL for PostgreSQL",
      "Memorystore for Redis",
      "Cloud Storage for media"
    ],
    "scores": {
      "performance": 0.90,
      "scalability": 0.93,
      "reliability": 0.92,
      "overall": 0.92
    }
  }
]
}}
```

Додаток 6. Аналіз вартості різних варіантів розгортання економічним агентом

```
{
  "cost_analysis": {
    "options": [
      {
        "option": "AWS Full Deployment",
        "components": [
          {"service": "EKS cluster (2 AZ)", "monthly_cost": 146},
          {"service": "EC2 instances (6 x t3.large)",
"monthly_cost": 496},
          {"service": "RDS PostgreSQL (db.m5.large, Multi-AZ)",
"monthly_cost": 340},
          {"service": "ElastiCache for Redis (cache.m4.large)",
"monthly_cost": 187},
          {"service": "S3 storage (120 GB)", "monthly_cost": 3},
          {"service": "CloudFront CDN (10 TB traffic)",
"monthly_cost": 860},
          {"service": "Load Balancer and other services",
"monthly_cost": 110}
        ],
        "total_monthly_cost": 2142,
        "cost_score": 0.78
      },
      {
        "option": "AWS+Azure Hybrid",
        "components": [
          {"service": "Azure CDN and App Service for frontend",
"monthly_cost": 420},
          {"service": "AWS for backend and database",
"monthly_cost": 1023},
```

```

        {"service": "Integration and data transfer costs",
"monthly_cost": 230}
    ],
    "total_monthly_cost": 1673,
    "cost_score": 0.92
},
{
    "option": "GCP Full Deployment",
    "components": [
        {"service": "Cloud Run for backend", "monthly_cost":
410},
        {"service": "Cloud SQL for PostgreSQL", "monthly_cost":
380},
        {"service": "Memorystore for Redis", "monthly_cost":
170},
        {"service": "Cloud Storage and CDN", "monthly_cost":
780},
        {"service": "Other services", "monthly_cost": 140}
    ],
    "total_monthly_cost": 1880,
    "cost_score": 0.85
}
]
}
}

```

Додаток 7. Аналіз відповідності рішень вимогам безпеки, виконаного безпековим агентом

```
{
  "security_analysis": {
    "requirements": ["gdpr_compliance", "data_encryption",
"authentication"],
    "provider_assessments": [
      {
        "provider": "AWS",
        "assessments": [
          {"aspect": "GDPR Compliance", "score": 0.95, "notes":
"European regions fully compliant, built-in tools for consent
management"},
          {"aspect": "Data Encryption", "score": 0.93, "notes":
"Full encryption in transit and at rest, AWS KMS"},
          {"aspect": "Authentication", "score": 0.94, "notes":
"IAM, Cognito, SSO integration"}
        ],
        "overall_security_score": 0.94
      },
      {
        "provider": "Azure",
        "assessments": [
          {"aspect": "GDPR Compliance", "score": 0.97, "notes":
"Strong GDPR compliance tools including Azure Information
Protection"},
          {"aspect": "Data Encryption", "score": 0.94, "notes":
"Advanced encryption system with Azure Key Vault"},
          {"aspect": "Authentication", "score": 0.96, "notes":
"Azure AD, enhanced identity capabilities"}
        ],
        "overall_security_score": 0.96
      }
    ]
  }
}
```

```

{
  "provider": "GCP",
  "assessments": [
    {"aspect": "GDPR Compliance", "score": 0.90, "notes":
"GDPR compliant but fewer specialized tools"},
    {"aspect": "Data Encryption", "score": 0.92, "notes":
"Standard encryption at rest and in transit"},
    {"aspect": "Authentication", "score": 0.90, "notes":
"Cloud Identity, Firebase Auth integration"}
  ],
  "overall_security_score": 0.91
},
{
  "provider": "AWS+Azure Hybrid",
  "overall_security_score": 0.96,
  "notes": "Leverages Azure AD and AWS KMS advantages"
}
]
}

```


Додаток 8. Агрегація агентом-координатором результатів та формулювання фінальної рекомендації

```
{
  "final_recommendation": {
    "options_assessment": [
      {
        "option": "AWS+Azure Hybrid",
        "scores": {
          "technical": 0.88,
          "economic": 0.92,
          "security": 0.96,
          "integrated_score": 0.92
        }
      },
      {
        "option": "AWS Full Deployment",
        "scores": {
          "technical": 0.92,
          "economic": 0.78,
          "security": 0.94,
          "integrated_score": 0.88
        }
      },
      {
        "option": "GCP Full Deployment",
        "scores": {
          "technical": 0.90,
          "economic": 0.85,
          "security": 0.91,
```

```

        "integrated_score": 0.87
    }
}

],
    "recommended_solution": "AWS+Azure Hybrid",
    "rationale": "Best balance between cost efficiency, technical
capabilities, and security. Leverages strengths of both platforms:
Azure for identity management and content delivery, AWS for core
computing infrastructure and data storage.",
    "architecture": {
        "frontend": "Azure Blob Storage with Azure CDN",
        "backend": "Containerized microservices on AWS EKS in
European and North American regions",
        "database": "AWS RDS for PostgreSQL with cross-region
replication",
        "caching": "AWS ElastiCache for Redis in each region",
        "media_storage": "AWS S3 with cross-region replication",
        "authentication": "Azure AD B2C for user management"
    }
}
}

```

Додаток 9. Аналіз вхідних вимог, виконаний агентом-координатором (сценарій 2)

```
{
  "application_type": "etl_data_processing",
  "technology_stack": {
    "orchestration": "Apache Airflow",
    "processing": ["Apache Spark", "Python"],
    "storage": ["S3", "Snowflake"],
    "monitoring": ["Prometheus", "Grafana"]
  },
  "data_characteristics": {
    "daily_volume_gb": 500,
    "data_types": {
      "csv": 0.4,
      "json": 0.3,
      "text": 0.2,
      "xml": 0.1
    },
    "workflows_count": 25,
    "largest_workflow_gb": 200
  },
  "resource_requirements": {
    "daily_cpu_hours": 500,
    "daily_memory_tb_hours": 2,
    "critical_time_window": ["01:00", "05:00"],
    "completion_time_hours": 6
  },
  "budget_constraint_usd": 8000,
  "regulatory_requirements": ["pci_dss"],
  "reliability_requirement_percent": 99.5
}
```

Додаток 10. Формулювання рекомендацій технічним агентом

```

{
  "technical_analysis": {
    "key_requirements": [
      "Managed services for Apache Airflow with high availability",
      "Scalable computing resources for 500 CPU hours/day",
      "Combined object storage and analytical database",
      "Separation of processing and storage"
    ],
    "deployment_options": [
      {
        "option": "AWS Full Deployment",
        "components": [
          "MWAA for Airflow",
          "EMR for Spark",
          "S3 for input data",
          "Redshift for analytics"
        ],
        "scores": {
          "performance": 0.88,
          "scalability": 0.93,
          "reliability": 0.91,
          "overall": 0.91
        }
      },
      {
        "option": "AWS+Snowflake Hybrid",
        "components": [
          "MWAA for Airflow",

```

```

        "EMR for Spark",
        "S3 for input data",
        "Snowflake for analytics"
    ],
    "scores": {
        "performance": 0.95,
        "scalability": 0.94,
        "reliability": 0.93,
        "overall": 0.94
    }
},
{
    "option": "GCP Full Deployment",
    "components": [
        "Cloud Composer for Airflow",
        "Dataproc for Spark",
        "GCS for storage",
        "BigQuery for analytics"
    ],
    "scores": {
        "performance": 0.87,
        "scalability": 0.92,
        "reliability": 0.90,
        "overall": 0.90
    }
}
]
}}
```

Додаток 11. Аналіз вартості різних варіантів розгортання економічним агентом

```
{
  "cost_analysis": {
    "options": [
      {
        "option": "AWS Full Deployment",
        "components": [
          {"service": "Amazon MWAA for Airflow", "monthly_cost":
850},
          {"service": "EMR for Spark (on-demand)", "monthly_cost":
2800},
          {"service": "S3 for data storage (15 TB)",
"monthly_cost": 345},
          {"service": "Redshift for analytics", "monthly_cost":
2200},
          {"service": "Monitoring and other services",
"monthly_cost": 350}
        ],
        "total_monthly_cost": 6545,
        "cost_score": 0.82
      },
      {
        "option": "AWS+Snowflake Hybrid",
        "components": [
          {"service": "Amazon MWAA for Airflow", "monthly_cost":
850},
          {"service": "EMR for Spark (on-demand)", "monthly_cost":
2800},
          {"service": "S3 for data storage (15 TB)",
"monthly_cost": 345},
          {"service": "Snowflake for analytics", "monthly_cost":
3200},
```

```

        {"service": "Monitoring and other services",
"monthly_cost": 350}
    ],
    "total_monthly_cost": 7545,
    "cost_score": 0.75
},
{
    "option": "GCP Full Deployment",
    "components": [
        {"service": "Cloud Composer for Airflow",
"monthly_cost": 950},
        {"service": "Dataproc for Spark", "monthly_cost": 2600},
        {"service": "Cloud Storage", "monthly_cost": 320},
        {"service": "BigQuery for analytics", "monthly_cost":
1900},
        {"service": "Monitoring and other services",
"monthly_cost": 380}
    ],
    "total_monthly_cost": 6150,
    "cost_score": 0.88
}
]
}
}

```

Додаток 12. Аналіз відповідності рішень вимогам безпеки, виконаного безпековим агентом

```
{
  "security_analysis": {
    "requirements": ["pci_dss", "data_protection",
"environment_isolation"],
    "provider_assessments": [
      {
        "provider": "AWS",
        "assessments": [
          {"aspect": "PCI DSS Compliance", "score": 0.96, "notes":
"Certified regions and services for PCI DSS"},
          {"aspect": "Data Protection", "score": 0.95, "notes":
"Comprehensive encryption, AWS KMS, IAM policies"},
          {"aspect": "Environment Isolation", "score": 0.94,
"notes": "CloudTrail, GuardDuty, Security Hub"}
        ],
        "overall_security_score": 0.95
      },
      {
        "provider": "Snowflake",
        "assessments": [
          {"aspect": "PCI DSS Compliance", "score": 0.97, "notes":
"Fully compliant with built-in audit tools"},
          {"aspect": "Data Protection", "score": 0.98, "notes":
"Column, row and file-level encryption"},
          {"aspect": "Environment Isolation", "score": 0.96,
"notes": "Detailed auditing, object-level access control"}
        ],
        "overall_security_score": 0.97
      },
      {

```



```
    "provider": "GCP",
    "assessments": [
      {"aspect": "PCI DSS Compliance", "score": 0.94, "notes":
"Compliant for all major services"},
      {"aspect": "Data Protection", "score": 0.93, "notes":
"Standard encryption, Cloud KMS"},
      {"aspect": "Environment Isolation", "score": 0.93,
"notes": "Cloud Audit Logs, Security Command Center"}
    ],
    "overall_security_score": 0.93
  }
]
}
}
```

Додаток 13. Агрегація агентом-координатором результатів та формулювання фінальної рекомендації

```
{
  "final_recommendation": {
    "options_assessment": [
      {
        "option": "AWS+Snowflake Hybrid",
        "scores": {
          "technical": 0.94,
          "economic": 0.75,
          "security": 0.97,
          "integrated_score": 0.89
        }
      },
      {
        "option": "AWS Full Deployment",
        "scores": {
          "technical": 0.91,
          "economic": 0.82,
          "security": 0.95,
          "integrated_score": 0.89
        }
      },
      {
        "option": "GCP Full Deployment",
        "scores": {
          "technical": 0.90,
          "economic": 0.88,
          "security": 0.93,
```

```

        "integrated_score": 0.90
    }
}
],
    "recommended_solution": "GCP Full Deployment",
    "rationale": "Despite slightly lower security scores, GCP
offers the best combination of technical capabilities and cost
efficiency. BigQuery provides high performance analytics at
significantly lower cost compared to Redshift or Snowflake. Cloud
Composer (managed Airflow) integrates well with other GCP services,
simplifying management and monitoring.",
    "alternative_solution": "AWS+Snowflake Hybrid is recommended
for cases where security and analytical capabilities have higher
priority than cost.",
    "architecture": {
        "orchestration": "Cloud Composer (managed Apache Airflow)",
        "processing": "Dataproc for Spark with autoscaling",
        "input_storage": "Cloud Storage with Standard and Nearline
storage classes",
        "analytics": "BigQuery for storing and analyzing processed
data",
        "monitoring": "Cloud Monitoring with Grafana",
        "security": "Cloud IAM, Cloud KMS for encryption, VPC Service
Controls for isolation"
    }
}
}

```

Додаток 14. Аналіз технічного агента

```
{
  "technical_analysis": {
    "workflow_characteristics": {
      "data_sources": ["CSV files from manufacturing equipment", "JSON logs from
quality control systems", "XML configuration data"],
      "processing_stages": [
        {
          "stage": "Data collection",
          "current_implementation": "Scheduled bash scripts on EC2 instances",
          "issues": ["Manual error handling", "No retry logic", "Fixed
scheduling regardless of data volume"]
        },
        {
          "stage": "Data transformation",
          "current_implementation": "Python scripts with Pandas on EC2",
          "issues": ["Memory limitations", "No parallel processing",
"Inefficient data loading"]
        },
        {
          "stage": "Data loading",
          "current_implementation": "Direct MySQL inserts",
          "issues": ["Database connection bottlenecks", "Scaling limitations",
"Complex transaction management"]
        }
      ],
      "performance_metrics": {
        "average_processing_time": "3.5 hours",
        "peak_processing_time": "7.2 hours",
        "failure_rate": "8%"
      }
    },
    "architecture_recommendations": {
      "primary_recommendation": "Shift to serverless event-driven
architecture",

```

```

    "rationale": [
        "Clear separation of processing stages makes it well-suited for Step
        Functions workflow",
        "Batch processing pattern aligns with Lambda execution model",
        "Data processing is mostly stateless, making it compatible with
        serverless paradigm",
        "Current resource utilization pattern shows significant idle time"
    ],
    "key_components": [
        {
            "component": "AWS Step Functions",
            "purpose": "Replace Jenkins for workflow orchestration",
            "benefits": ["Built-in error handling", "Visual workflow monitoring",
            "Native AWS integration"]
        },
        {
            "component": "AWS Lambda",
            "purpose": "Replace EC2-based Python processing",
            "benefits": ["Automatic scaling", "Pay-per-execution", "No
            infrastructure management"]
        },
        {
            "component": "AWS S3",
            "purpose": "Replace direct file access on EC2",
            "benefits": ["Unlimited storage", "High durability", "Pay for used
            storage only"]
        }
    ]
}

```

Додаток 15. Аналіз економічного агента

Агент визначив, що існуюча інфраструктура коштує:

- обчислювальні ресурси: \$2 000/місяць;
- зберігання даних: \$400/місяць;
- інші послуги: \$100/місяць;
- операційні витрати (адміністрування): \$2 000/місяць;
- загальна вартість володіння: \$4 500/місяць.

Запропонована безсерверна архітектура передбачала наступні витрати:

- AWS Step Functions: \$120/місяць;
- AWS Lambda: \$350/місяць;
- AWS S3: \$150/місяць;
- AWS Athena: \$300/місяць;
- AWS Glue: \$280/місяць;
- загальна вартість інфраструктури: \$1 200/місяць;
- операційні витрати: \$300/місяць;
- загальна вартість володіння: \$1 500/місяць. Це відповідає економії 67% від загальної вартості володіння.

```
{
  "economic_analysis": {
    "current_cost_breakdown": {
      "compute_resources": {
        "EC2_instances": {
          "m5.xlarge_processing_instances": "$1,620/month    (5
instances)",
          "r5.large_database_instance": "$380/month",
          "total_compute": "$2,000/month"
        },

        "storage": {
          "EBS_volumes": "$320/month",
          "S3_current_usage": "$80/month",
          "total_storage": "$400/month"
        },
```

```

    "other_services": {
        "data_transfer": "$70/month",
        "monitoring": "$30/month",
        "total_other": "$100/month"
    },
    "total_monthly_infrastructure_cost": "$2,500/month"
},
"operational_costs": {
    "administration_time": "20 hours/month",
    "estimated_labor_cost": "$2,000/month",
    "total_operational_cost": "$2,000/month"
},
"total_cost_of_ownership": "$4,500/month"
},
"cost_optimization_recommendations": {
    "serverless_transformation": {
        "projected_costs": {
            "AWS_Step_Functions": "$120/month",
            "AWS_Lambda": "$350/month",
            "AWS_S3": "$150/month",
            "AWS_Athena": "$300/month",
            "AWS_Glue": "$280/month",
            "total_serverless_infrastructure": "$1,200/month"
        },
        "operational_costs": {
            "administration_time": "3 hours/month",
            "estimated_labor_cost": "$300/month",
            "total_operational_cost": "$300/month"
        },

```

```

    "projected_total_cost_of_ownership": "$1,500/month",
    "cost_reduction": {
        "infrastructure_savings": "$1,300/month (52%)",
        "operational_savings": "$1,700/month (85%)",
        "total_monthly_savings": "$3,000/month (67%)"
    }
},
    "cost_optimization_rationale": [
        "Serverless pricing model eliminates costs for idle resources",
        "Athena's query-based pricing better aligns with actual usage patterns",
        "Managed services reduce operational overhead",
        "Step Functions provides better visibility into workflow failures, reducing debugging time"
    ]
}
}
}

```


Додаток 16. Аналіз безпекового агента

```

{
  "security_analysis": {
    "current_security_posture": {
      "access_control": {
        "authentication_method": "SSH keys for EC2 access",
        "database_access": "Shared credentials embedded in
configuration files",
        "issues": ["No fine-grained access control", "Manual
credential rotation", "Lack of audit trail"]
      },
      "data_protection": {
        "encryption_in_transit": "Partial - only for database
connections",
        "encryption_at_rest": "Only for some EBS volumes",
        "issues": ["Inconsistent encryption practices", "Manual
key management"]
      },
      "compliance": {
        "audit_capabilities": "Limited manual log review",
        "data_lineage_tracking": "Not implemented",
        "issues": ["Difficult to demonstrate compliance", "No
automated controls"]
      },
      "security_score": 0.65
    },
    "security_recommendations": {
      "recommended_approach": "Transition to AWS serverless
security model",
      "key_recommendations": [
        {
          "aspect": "Access Control",

```

```

        "recommendation": "Use AWS IAM with fine-grained roles
for service access",

        "benefits": ["Principle of least privilege",
"Centralized identity management", "Automated credential
rotation"]

    },

    {

        "aspect": "Data Protection",

        "recommendation": "Use S3 with default encryption and
Athena with encrypted results",

        "benefits": ["Consistent encryption approach",
"Automatic key management", "Simplified compliance"]

    },

    {

        "aspect": "Audit and Compliance",

        "recommendation": "Enable CloudTrail for API auditing
and use Step Functions execution history",

        "benefits": ["Comprehensive audit trail", "Visual
workflow history", "Simplified compliance reporting"]

    }

],

"projected_security_improvements": {

    "access_control_improvement": "92%",

    "data_protection_improvement": "85%",

    "auditability_improvement": "94%",

    "overall_security_score_improvement": "0.65 → 0.91"

}

}

}}

```

Додаток 17. Процес прийняття рішення агентом-координатором

```

{
  "coordinator_decision_process": {
    "evaluated_architectural_alternatives": [
      {
        "architecture": "Optimized EC2-based architecture",
        "description": "Keep EC2 instances but optimize AMI,
instance types, and add auto-scaling",
        "scores": {
          "technical": 0.72,
          "economic": 0.65,
          "security": 0.70,
          "integrated_score": 0.69
        }
      },
      {
        "architecture": "Hybrid EC2/Serverless architecture",
        "description": "Keep core processing on EC2 but migrate
orchestration to Step Functions",
        "scores": {
          "technical": 0.78,
          "economic": 0.73,
          "security": 0.82,
          "integrated_score": 0.77
        }
      },
      {
        "architecture": "Full serverless architecture with AWS
Step Functions, Lambda, S3, and Athena",
        "description": "Complete migration to serverless
components",

```

```

    "scores": {
      "technical": 0.89,
      "economic": 0.92,
      "security": 0.91,
      "integrated_score": 0.91
    }
  },
  {
    "architecture": "Multi-cloud serverless (AWS + GCP)",
    "description": "Use AWS Step Functions and Lambda with GCP
BigQuery",
    "scores": {
      "technical": 0.84,
      "economic": 0.79,
      "security": 0.83,
      "integrated_score": 0.82
    }
  }
],
"decision_rationale": [
  "Full serverless architecture provides the best balance of
technical capabilities, cost efficiency, and security
improvements",
  "Clear workflow separation in the current process maps well
to Step Functions state machine model",
  "Data processing patterns are well-suited for Lambda's
execution model",
  "The analytical nature of the workload makes Athena an
excellent fit for post-processing queries",
  "Integration with existing AWS environment would be simpler
with all-AWS solution"
],

```

```
"final_recommendation": {  
    "architecture": "Full serverless architecture with AWS Step  
Functions, Lambda, S3, and Athena",  
    "implementation_path": {  
        "phase_1": "Migrate data storage from direct EC2 access to  
S3",  
        "phase_2": "Refactor Python processing scripts for Lambda  
compatibility",  
        "phase_3": "Create Step Functions workflow to replace  
Jenkins orchestration",  
        "phase_4": "Set up Athena views and queries to replace  
direct database access",  
        "phase_5": "Implement Glue Data Catalog for metadata  
management",  
        "estimated_migration_time": "4 weeks"  
    }  
}  
}
```

Додаток 18. Безсерверна архітектура

1. AWS Step Functions для оркестрації та управління ETL-процесами замість Jenkins. Це забезпечує:
 - Безсерверну архітектуру без необхідності підтримки серверів.
 - Вбудовану обробку помилок та автоматичні повторні спроби.
 - Візуальне відстеження потоків роботи.
2. AWS S3 для зберігання вхідних та проміжних даних замість локального зберігання на EC2. Переваги:
 - необмежена масштабованість;
 - низька вартість зберігання;
 - можливість використання різних рівнів зберігання для оптимізації витрат.
3. AWS Athena для аналізу оброблених даних замість запитів до MySQL. Переваги:
 - оплата лише за виконані запити;
 - відсутність необхідності управління інфраструктурою;
 - нативна інтеграція з S3.
4. AWS Lambda для трансформації даних замість скриптів на EC2. Переваги:
 - автоматичне масштабування відповідно до навантаження;
 - детальна модель ціноутворення;
 - нульові витрати на обслуговування.
5. AWS Glue для управління метаданими та каталогом даних. Переваги:
 - автоматичне виявлення схеми даних;
 - безшовна інтеграція з Athena.

Додаток 19. Аналіз відгуків фахівців та експертних оцінок

Для оцінки ефективності запропонованого рішення та роботи системи загалом було отримано експертні відгуки від трьох ключових спеціалістів компанії:

- Програміста, який брав участь у розробці та підтримці вихідної ETL-системи.
- Проектного менеджера, відповідального за керування процесом міграції.
- Системного архітектора, який оцінив технічні аспекти запропонованого рішення.

Загальна оцінка системи фахівцями була дуже позитивною, з відзначенням наступних ключових аспектів:

- Точність технічного аналізу – 4.8/5.0;
- Економічна обґрунтованість рекомендацій – 4.7/5.0;
- Практична реалізованість запропонованих рішень – 4.6/5.0;
- Повнота охоплення безпекових аспектів – 4.5/5.0;
- Якість документації та пояснень – 4.9/5.0.

Особливо було відзначено здатність системи надавати детальне обґрунтування кожної рекомендації та поетапний план міграції, що суттєво спростило процес прийняття рішень та впровадження змін.

Детальна інформація про результати впровадження, включно з кількісними показниками, методологією оцінки та повними відгуками експертів, представлена у відповідному акті впровадження.

Додаток 20. Список публікацій за темою дисертації та відомості про апробацію результатів дисертації

Наукові праці, в яких опубліковані основні наукові результати дисертації

1. Caceres, A., & Globa, L. (2025). AHP-based multi-criteria analysis of multi-cloud data management techniques. *Radioelectronic and Computer Systems*, 1, Article 04. <https://doi.org/10.32620/reks.2025.1.04>.
2. Caceres, A., & Globa, L. (2025). Аналіз підходів доступу до даних у мульти-клауд середовищі [Analysis of data access approaches in a multi-cloud environment]. *Radio Electronics, Computer Science, Control*, 1. Retrieved from <https://ric.zp.edu.ua/article/view/324539/314660>.
3. Caceres, A., & Globa, L. (2024). Підхід до формування мультихмарного середовища на основі багатокритеріального аналізу та онтологічного моделювання [A multi-criteria and ontology-based approach to multi-cloud environment selection]. *Системи управління, навігації та зв'язку*, 4. <https://doi.org/10.26906/SUNZ.2024.4.084>.
4. Caceres, A., & Globa, L. (2024). Інтеграція ШІ та мультиагентних систем для багатокритеріального аналізу у мультихмарному середовищі [Integration of AI and multi-agent systems for multi-criteria analysis in a multicloud environment]. *Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки*, 35(74), № 6, Частина 2. Retrieved from https://www.tech.vernadskyjournals.in.ua/journals/2024/6_2024/part_2/12.pdf.
5. Caceres, A., & Globa, L. (2022). State-of-the-art architectures for interoperability of heterogeneous clouds. In *Proceedings of the 2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)* (pp. 704–709). IEEE. <https://doi.org/10.1109/TCSET55632.2022.9766965>.

Відомості про апробацію результатів дисертації

Результати роботи та ключові положення були представлені й одержали схвалення на:

- міжнародній технічній конференції PyCon Poland 2023 (м. Краків, Польща, 2023 р.);
- міжнародній конференції Voxxed Days Cluj 2023 (м. Клуж-Напока, Румунія, 2023 р.);
- міжнародній конференції PyConTH 2023 (м. Бангкок, Таїланд, 2023 р.);
- технічній конференції PyConCZ 2023 (м. Прага, Чехія, 2023 р.);
- міжнародній конференції Pycon IT 2023 (м. Флоренція, Італія, 2023 р.);
- європейській конференції Europython 2024 (м. Прага, Чехія, 2024 р.);
- міжнародній конференції PyconDE 2024 (м. Берлін, Німеччина, 2024 р.);
- технічній конференції Pycon Wroclaw 2024 (м. Вроцлав, Польща, 2024 р.).

Додаток 21. Акт впровадження

Continental Automotive Technologies GmbH Sodener Strasse
9, 65824 Schwalbach am Taunus
P.O. Box 61 40 | D-65735 Eschborn

Regensburg, 04.06.2025

Act**On the implementation of a comprehensive method for optimizing computational infrastructure in a heterogeneous cloud environment**

We, the undersigned, representatives of Continental Automotive Technologies GmbH, hereby certify that in 2025, during the course of cooperation between Continental and Anton Caceres, a computational infrastructure approach developed by Anton Caceres was implemented as part of enterprise-scale cloud engineering activities. This approach constitutes a core component of his dissertation entitled: "Comprehensive approach to designing computational infrastructure in a heterogeneous multi-cloud environment."

The method was applied in production-grade projects, particularly in the context of Continental's PRIME-MES data platform. Relevant use cases included cloud infrastructure optimization, data lake storage cost control, and workload distribution. The approach guided architectural decisions and dynamic resource allocation strategies to enhance performance, resilience, and efficiency in cloud operations.

As a result of applying the proposed approach, Continental recorded a significant decrease in cloud infrastructure costs and an improvement in the performance of its ETL pipelines. In particular, the cost of storing Parquet files in the datalake decreased by over 30%, and the improved parallelism of step functions allowed those files to be processed up to 5 times faster.

**On behalf of Continental
Automotive Technologies GmbH**


(signature)  _____

Peter Gal

Project Manager 04.06.2025



On behalf of NTUU "KPI"

(signature)  _____

Anton Caceres

(signature)  _____

Prof. Dr. Sc. Larysa S. Globa

Academic Supervisor

Date: 4 June 2025

Continental Automotive
Technologies
GmbH
Sodener Strasse 9
65824 Schwalbach am Taunus
P.O. Box 61 40
D-65735 Eschborn

Telefon +49 6196 87-0
Telefax +49 6196 86571
continental-corporation.com

Geschäftsführer:
Dr. Andreas Listl, Albrecht
Poettcher,
Frank Staiger, Harald
Stuhlmann,
Nicole Werner

Sitz der Gesellschaft:
Hannover
Registergericht: Amtsgericht
Hannover HRB 3669
USt.-ID-Nr. DE 341447066