

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук
(повна назва)
Кафедра _____ Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський)

_____ Мультиагентна система гібридних сервісів із механізмами когнітивної
_____ апроксимації експертних функцій в умовах волатильності ринку
(тема)

Виконав:
здобувач _____ другого _____ року навчання,
групи _____ ІТШІ-22-1

_____ Віктор ПОНОМАРЬОВ
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)

Тип програми _____ освітньо-професійна
Освітня програма _____ Штучний інтелект

(повна назва освітньої програми)

Керівник _____ ст. викл. Олена ГРИНЬОВА
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ

(підпис)

_____ Лариса ЧАЛА
(власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____

Кафедра _____ Штучного інтелекту _____

Рівень вищої освіти _____ перший (бакалаврський) _____

Спеціальність _____ 122 Комп'ютерні науки _____
(код і повна назва)

Тип програми _____ освітньо-професійна _____

Освітня програма _____ Штучний інтелект _____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____» _____ 20 ____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві _____ Пономарьову Віктору Олександровичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____ Мультиагентна система гібридних сервісів із механізмами когнітивної апроксимації експертних функцій в умовах волатильності ринку _____

затверджена наказом університету від 5 червня 2026 р. № 488Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 9 липня 2026 р.

3. Вихідні дані до роботи _____ Науково-технічні публікації та дослідження щодо структурної живучості складних мереж; документація великих мовних моделей (Sentence-BERT) та ієрархічних мереж завдань (HTN); фреймворки розробки гібридних ШІ-асистентів; набори даних для симуляції волатильності сервісних запитів в умовах українського ринку. _____

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі та постановка задачі

2) Проєктування гібридної системи TeamFinder

3) Програмна реалізація та тестування системи

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.06.2026	виконано
2	Аналіз предметної галузі	09.06.2026	виконано
3	Огляд та порівняльний аналіз існуючих рішень	10.06.2026	виконано
4	Постановка задачі та проєктування системи	11.06.2026	виконано
5	Формування онтологічної бази знань	12.06.2026	виконано
6	Програмна реалізація модулів системи	13.06.2026	виконано
7	Розроблення інтерфейсу та тестування системи	19.06.2026	виконано
8	Написання пояснювальної записки	22.06.2026	виконано
9	Перевірка на академічний плагіат	23.06.2026	виконано
10	Нормоконтроль	24.06.2026	виконано
11	Підготовка презентації та доповіді	27.06.2026	виконано
12	Попередній захист, рецензування	04.07.2026	виконано
13	Захист перед ЕК	09.07.2026	виконано

Дата видачі завдання 8 червня 20 26 р.

Здобувач 
(підпис)

Керівник роботи 
(підпис)

ст. викл. Олена ГРИНЬОВА
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 66 с., 3 рис., 2 табл., 1 дод., 20 джерел.

ГІБРИДНІ СЕРВІСИ, ІНТЕЛЕКТУАЛЬНА СИСТЕМА,
КОГНІТИВНИЙ МОДУЛЬ, МУЛЬТИАГЕНТНА МОДЕЛЬ,
РЕКОНФІГУРАЦІЯ МЕРЕЖІ.

Об'єкт дослідження – процес функціонування цифрових сервісних платформ в умовах екстремальної волатильності та дефіциту ресурсів.

Предмет дослідження – методи автономної реконфігурації гібридних систем та заміщення фізичних вузлів когнітивними модулями.

Мета роботи – розробити концепцію та математичну модель інтелектуальної системи TeamFinder, здатної до автономної адаптації та підтримки структурної живучості шляхом трансформації розривів сервісного ланцюга в протокол інформаційної компенсації.

Методи дослідження – мультиагентне моделювання, теорія графів, методи обробки природної мови (Sentence-BERT), ієрархічне мережеве планування (HTN).

У звіті розглядається проблематика нестабільності традиційних сервісних платформ в умовах воєнного стану. Запропоновано архітектуру гібридної системи TeamFinder з можливістю автономної підстановки ІІІ-асистентів (DIY Mode) замість відсутніх виконавців. Розроблено математичну модель на базі динамічних графів та ентропійного фільтру. Система забезпечує структурну живучість і функціональне самовідновлення.

ABSTRACT

Bachelor's thesis contains: 66 p., 3 fig., 2 tabl., 1 ann., 20 references.

COGNITIVE MODULE, HYBRID SERVICES, INTELLIGENT SYSTEM, MULTI-AGENT MODEL, NETWORK RECONFIGURATION.

Object of research – the process of functioning of digital service platforms under conditions of extreme volatility and resource constraints.

Subject of research – methods of autonomous reconfiguration of hybrid systems and substitution of physical nodes with cognitive modules.

Purpose of the work – to develop the concept and mathematical model of the TeamFinder intelligent system, capable of autonomous adaptation and maintaining structural robustness by transforming service chain disruptions into an information compensation protocol.

Research methods – multi-agent modeling, graph theory, natural language processing (Sentence-BERT), hierarchical task network (HTN) planning.

The report addresses the instability of traditional service platforms under martial law. The architecture of the TeamFinder hybrid system is proposed, featuring autonomous substitution of AI assistants (DIY Mode) for absent executors. A mathematical model based on dynamic graphs and an entropy filter has been developed. The system ensures structural robustness and functional self-recovery.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз предметної галузі та постановка задачі.....	11
1.1 Цифрові сервісні платформи як складні динамічні мережі.....	11
1.2 Вплив екстремальної волатильності та дефіциту ресурсів на цифрові сервіси	13
1.3 Огляд та порівняльний аналіз існуючих рішень та їхні структурні обмеження.....	17
1.4 Огляд фреймворків для розробки мультиагентних систем штучного інтелекту.....	20
1.5 Постановка задачі.....	21
2 Проектування гібридної системи teamfinder	25
2.1 Архітектура гібридної системи та концепція автономної реконфігурації.....	25
2.2 Мультиагентна математична модель та ентропійний фільтр.....	28
2.3 Двохстадійний алгоритм обслуговування на базі HTN-планування.	32
2.4 Проектування когнітивного модуля підтриманої самопомоги (DIY Mode) на базі Sentence-BERT	36
3 Програмна реалізація та тестування системи.....	39
3.1 Обґрунтування вибору засобів та технологій розробки	39
3.1.1 Мова програмування та базове середовище	39
3.1.2 Фреймворк для реалізації мікросервісної архітектури	40
3.1.3 Засоби роботи зі штучним інтелектом та обробки природної мови	40
3.1.4 База даних для динамічної топології	41
3.1.5 Апаратне забезпечення та середовище для тестування (Інференсу).....	41
3.2 Програмна реалізація мультиагентної взаємодії та бази знань.....	42

3.2.1 Програмна ієрархія мультиагентної системи.....	42
3.2.2 Реалізація графа завдань та ентропійного фільтра.....	46
3.2.3 Проєктування гібридної бази знань	48
3.3 Реалізація модуля ін'єкції знань та косинусної фільтрації	49
3.3.1 Імплементация семантичної векторизації (Sentence-BERT)	50
3.3.2 Програмна реалізація векторного індексу за допомогою FAISS	50
3.3.3 Алгоритм косинусної фільтрації та відсікання (Thresholding) .	53
3.3.4 Інтеграція контексту та генеративний вивід (MCP-протокол) .	53
3.4 Інтерфейс користувача та тестування роботи системи в умовах симуляції відмов (волатильності).....	54
3.4.1 Організація взаємодії з користувачем (UI/UX) та візуалізація DIU-модуля	54
3.4.2 Методика тестування та симуляція екстремальної волатильності.....	55
3.4.3 Аналіз результатів експериментального дослідження.....	57
3.5 Алгоритмічне забезпечення бізнес-логіки та фінансова живучість платформи	58
3.5.1 Математична модель динамічного ціноутворення та розрахунку комісій	58
3.5.2 Диверсифікація платіжних шлюзів в умовах інфраструктурних відмов	60
3.5.3 Економічна ефективність впровадження когнітивної компенсації	60
Висновки	62
Перелік джерел посилання	64
Додаток А Відомість кваліфікаційної роботи	66

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ASGI – Asynchronous Server Gateway Interface – стандарт асинхронного інтерфейсу для серверів та фреймворків;

CUDA – технологія паралельних обчислень для обробки математично інтенсивних операцій;

DAG – Directed Acyclic Graph – орієнтований ациклічний граф;

DFS – Depth-First Search – алгоритм обходу графа або пошуку в глибину;

FAISS – Facebook AI Similarity Search – векторна база даних для швидкої косинусної фільтрації та пошуку схожості;

HMAS – Hierarchical Multi-Agent Systems – ієрархічні мультиагентні системи;

HTN – Hierarchical Task Network – ієрархічні мережі завдань для декомпозиції та планування комплексних процесів;

MCP – Model Context Protocol – протокол, що стандартизує комунікаційний шар між когнітивним агентом та інфраструктурою контексту даних;

NLP – Natural Language Processing – методи обробки природної мови;

RR – Resolution Rate – метрика загального відсотка успішно закритих завдань;

S-BERT – Sentence-BERT – нейромережева сіамська архітектура для семантичної векторизації завдань на базі Transformer;

SLA – Service Level Agreement – фіксована угода про надання послуги між замовником та виконавцем;

SRR – Service Rejection Rate – метрика відсотка запитів, які система не змогла задовольнити (повна відмова).

ВСТУП

Сучасна теорія складних систем розглядає цифрові сервісні платформи не просто як програмні додатки, а як динамічні мережі, стабільність яких критично залежить від безперервності зв'язків між вузлами-замовниками та вузлами-виконавцями. В умовах воєнного стану, постійної ризикової невизначеності, критичного дефіциту кадрів та енергетичної нестабільності, з якими наразі стикається Україна, створюється середовище з екстремальною волатильністю. Традиційні системи (наприклад, аналоги TaskRabbit або Meetup) демонструють низьку структурну «живучість» (structural robustness), оскільки для разових запитів їхня відмова при зникненні людського ресурсу є критичною. При неможливості надання послуги фізичним виконавцем ланцюг руйнується, а запит залишається невиконаним. Це формує нагальну потребу у створенні нового покоління систем «простору довіри», які інтегрують класичні функції пошуку фахівців із сучасними механізмами інтелектуальної підтримки.

Актуальність роботи. Розробка інтелектуальної гібридної системи, здатної до автономної реконфігурації, є надзвичайно актуальною задачею. У разі відсутності фізичного виконавця система не повинна видавати помилку, натомість вона має здійснювати автономну підстановку відсутнього вузла когнітивним модулем підтримки (штучним інтелектом), забезпечуючи безперервність обслуговування.

Об'єкт дослідження – процес функціонування цифрових сервісних платформ в умовах екстремальної волатильності та дефіциту ресурсів.

Предмет дослідження – методи автономної реконфігурації гібридних систем та заміщення фізичних вузлів когнітивними модулями.

Мета роботи – розробити концепцію, математичну модель та алгоритмічне забезпечення інтелектуальної системи TeamFinder, здатної до автономної адаптації та підтримки структурної живучості шляхом

трансформації розривів сервісного ланцюга в протокол інформаційної компенсації.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- провести огляд предметної галузі та проаналізувати структурні обмеження існуючих цифрових сервісних платформ;
- розробити мультиагентну математичну модель системи на базі динамічних графів та ентропійного фільтра;
- розробити двохстадійний алгоритм обслуговування на базі апарату ієрархічних мереж завдань (HTN-планування) для управління графом задач;
- створити механізм автономної реконфігурації (DIY Mode), який забезпечить підстановку когнітивного ШІ-модуля з використанням методів обробки природної мови (Sentence-BERT);
- здійснити програмну реалізацію архітектури системи TeamFinder та оцінити її функціональне самовідновлення при тестуванні.

Сфера застосування. Отримані теоретичні та практичні результати можуть бути застосовані для модернізації архітектури існуючих маркетплейсів послуг, а також для розробки стійких соціально-побутових та інфраструктурних платформ, що функціонують в умовах дефіциту кадрових ресурсів та нестабільного середовища.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Цифрові сервісні платформи як складні динамічні мережі

В епоху глобальної цифровізації та стрімкого розвитку економіки спільного споживання (sharing economy), цифрові сервісні платформи стали фундаментальним елементом соціально-економічної інфраструктури. Від початкових етапів свого розвитку, коли платформи виконували функцію простих електронних дошок оголошень, вони еволюціонували до рівня масштабних екосистем, здатних в режимі реального часу керувати розподілом ресурсів, узгоджувати попит та пропозицію, а також гарантувати виконання транзакцій. Відповідно до цього, сучасна теорія складних систем розглядає цифрові сервісні платформи не просто як програмні додатки, а як складні динамічні мережі [1].

Розглядаючи архітектуру таких систем через призму мережевого аналізу, можна виділити їхню гетерогенну природу. Стабільність таких систем критично залежить від безперервності та надійності зв'язків між ключовими агентами: вузлами-замовниками (користувачами, які генерують запити) та вузлами-виконавцями (постачальниками послуг або ресурсів). Ці зв'язки не є статичними; вони виникають і зникають динамічно, утворюючи складну топологію взаємодій [2]. Кожен успішно виконаний запит формує тимчасове ребро між вузлом-замовником та вузлом-виконавцем, зміцнюючи загальну зв'язність мережі та її здатність до обміну інформацією.

З точки зору теорії складних мереж, цифрові сервісні платформи є динамічними графами. На відміну від статичних графів, топологія яких є визначеною та незмінною в часі, динамічні графи сервісних маркетплейсів характеризуються постійною флуктуацією кількості активних вершин та ребер. Це означає, що система постійно перебуває в стані перехідного процесу. У таких умовах зникнення вузлів-виконавців (наприклад, через

дефіцит кадрів або міграційні процеси) еквівалентне цілеспрямованій атаці на топологію мережі. Коли мережа втрачає найбільш активні вузли-виконавці (так звані «хаби»), її здатність маршрутизувати запити різко падає, що призводить до фрагментації графа та утворення ізольованих підграфів замовників, чий потреби не можуть бути задоволені.

Згідно з дослідженнями в галузі структурної живучості складних мереж (structural robustness), стійкість таких систем зазвичай оцінюється за допомогою а posteriori мір, які аналізують стан мережі після видалення вузлів або порушення зв'язків. Цей підхід означає, що традиційні метрики зосереджені на констатації факту деградації системи (наприклад, підрахунок відсотка невиконаних замовлень або вимірювання збільшення середнього часу відгуку). Проте а posteriori аналіз є виключно реактивним інструментом: він дозволяє оцінити масштаб пошкоджень топології, але не надає механізмів для проактивного запобігання цим пошкодженням або автоматичного відновлення зв'язності в момент виникнення розриву.

У класичній моделі обслуговування процес виконання складного запиту від користувача реалізується як функція відображення на орієнтований ациклічний граф (DAG) завдань. Цей математичний апарат дозволяє декомпозиувати глобальну потребу замовника на набір керованих елементів. У цьому графі множина вершин відповідає атомарним підзавданням (наприклад, пошук майстра, виклик сантехніка, демонтаж). Кожне атомарне підзавдання вимагає специфічного ресурсу або навички, і система повинна знайти відповідний вузол-виконавець для кожної вершини графа.

Множина ребер визначає логічну послідовність їх виконання. Ребра в DAG-моделі задають строгі часові та логічні залежності: наприклад, підзавдання «демонтаж» не може розпочатися доти, доки не буде успішно завершено підзавдання «виклик майстра». Таким чином, загальний успіх процесу обслуговування залежить від безперебійного проходження через кожен вузол графа.

Однак, фундаментальна проблема полягає в тому, що ефективність такої моделі є високою лише у стабільному середовищі, де ймовірність знаходження вільного вузла-виконавця для кожного завдання наближається до абсолютного максимуму. Класичні DAG-орієнтовані системи проєктувалися з припущенням про надлишок пропозиції на ринку, де на кожен запит існує кілька доступних альтернативних виконавців. У таких ідеальних умовах алгоритми балансування навантаження працюють бездоганно. Проте, як тільки система стикається з масовим дефіцитом вузлів-виконавців, парадигма жорсткого відображення завдань на наявні ресурси колапсує, вимагаючи впровадження нових, адаптивних механізмів, здатних підтримувати функціонування навіть за умови неповної комплектації графа.

1.2 Вплив екстремальної волатильності та дефіциту ресурсів на цифрові сервіси

Функціонування сучасних цифрових платформ у стабільному ринковому середовищі спирається на відносно передбачувані стохастичні моделі, де коливання попиту та пропозиції підпорядковуються законам нормального розподілу. Проте, виникнення системних криз, викликаних геополітичними, воєнними чи масштабними інфраструктурними чинниками, переводить сервісні мережі в стан екстремальної волатильності. Під екстремальною волатильністю у контексті даного дослідження слід розуміти такий режим функціонування ринку послуг, при якому амплітуда та частота змін стану ключових параметрів системи (доступність виконавців, щільність вхідного потоку заявок, географічний розподіл активних вузлів) перевищують критичні пороги, унеможлиблюючи застосування стандартних методів предиктивного планування та балансування навантаження [3].

Умови воєнного стану, що є характерними для сучасного українського контексту, зокрема для Харківського регіону, виступають потужним дестабілізуючим фактором. Традиційні архітектурні рішення цифрових маркетплейсів демонструють критичну вразливість перед комплексом деструктивних чинників, які можна класифікувати за трьома основними векторами впливу:

– постійна ризикова невизначеність та її алгоритмічні наслідки. Безпекова ситуація в регіоні створює умови, за яких життєвий цикл сервісної транзакції стає непередбачуваним на будь-якому з етапів його виконання. У класичних системах процес обробки замовлення передбачає фіксацію угоди (Service Level Agreement) між замовником та виконавцем на певний проміжок часу. В умовах постійної загрози та зміни безпекових статусів локацій ймовірність раптової відмови виконавця (чи навіть замовника) від виконання своїх зобов'язань зростає експоненціально. З погляду теорії графів, це призводить до випадкового і миттєвого видалення ребер графа завдань безпосередньо в процесі їхньої ініціалізації. Традиційні алгоритми у таких випадках запускають процедуру повторного пошуку (re-matching), яка в умовах дефіциту часу та альтернативних вузлів призводить до лавиноподібного збільшення часу очікування та, зрештою, до повного скасування сесії;

– енергетична нестабільність та десинхронізація розподілених систем. Раптові інфраструктурні відключення електроенергії (блекаути) та порушення роботи мереж зв'язку завдають прямого удару по комунікаційному шару цифрових платформ. У розподілених сервісних системах критично важливою є підтримка постійної синхронізації станів між клієнтським додатком, сервером координації та терміналом виконавця. Коли цілі сегменти міської інфраструктури випадають з енергомережі, відбувається так звана мережева фрагментація (network partitioning). Агенти-виконавці втрачають зв'язок із платформою, що системою інтерпретується як їхній повний офлайн-статус, навіть якщо фізично вони

готові виконувати роботу. Це викликає каскадні тайм-аути на рівні протоколів передачі даних [4]. Серверна частина платформи виявляється перевантаженою запитами на перепланування, які вона не може задовольнити через відсутність актуальної інформації про реальну топологію доступних вузлів;

– критичний дефіцит людських ресурсів та деградація сервісного пулу. Масштабні міграційні процеси, внутрішнє переміщення населення та мобілізаційні заходи кардинально змінюють демографічний та професійний ландшафт ринку праці. Для цифрових платформ пошуку спеціалістів це означає фізичне зникнення цілих категорій виконавців, які раніше формували стійку пропозицію. Коли кількість активних сервісних вузлів (вершин графа системи) падає нижче критичного рівня, щільність мережі стає недостатньою для підтримки її життєздатності. Виникає ситуація, коли на вузькоспеціалізовані, але життєво необхідні побутові чи технічні запити (наприклад, аварійний ремонт комунікацій, екстрена допомога) у системі просто немає жодного активного агента у радіусі доступності. Мережа втрачає свої властивості «масштабованості без обмежень» (scale-free network) і починає фрагментуватися на ізольовані, нефункціональні елементи.

Сукупна дія цих трьох факторів призводить до явища, яке в теорії складних мереж називається каскадною відмовою. Розрив однієї ланки в орієнтованому ациклічному графі завдань (DAG), викликаний, наприклад, тим, що майстер потрапив під знеструмлення або змушений змінити локацію через безпекові ризики, автоматично блокує виконання всіх наступних підзадач. Якщо завдання №1 «Аварійний демонтаж пошкодженого елемента» не виконано, то завдання №2 «Монтаж нової системи» і завдання №3 «Пусконаладжувальні роботи» стають недосяжними, навіть якщо для них виконавці є в наявності.

Особливої гостроти проблема екстремальної волатильності набуває на рівні мікрологістики та функціонування локальних роздрібних мереж,

зокрема тих, що розміщені у специфічних інфраструктурних об'єктах, наприклад, торговельні точки у підземних переходах метрополітену, укриттях або напівпідвальних приміщеннях. Такі сервісні вузли виявляються найбільш вразливими під час віялових або аварійних відключень електроенергії. Через архітектурні особливості розташування вони першими втрачають покриття мобільних мереж, перетворюючись на «сліпі зони» на топологічній мапі сервісної платформи.

За таких умов виникає парадоксальна ситуація: потреба у послугі, наприклад, терміновий ремонт касового обладнання, налаштування локальної мережі чи переміщення товарних залишків між філіями, стає критичною, але класична система маркетплейсу не здатна з'єднати замовника з виконавцем через тайм-аути підключення. Доки класичні алгоритми безперервно генерують спроби повторного підключення, бізнес-процес простоює. Це наочно демонструє, що в реаліях сучасного українського ринку очікування відновлення зв'язку з фізичним виконавцем є економічно недоцільним. Система повинна мати вбудований протокол автономної роботи, який дозволить персоналу на місці частково закрити проблему власними силами за допомогою згенерованих когнітивним модулем інструкцій, не чекаючи на прибуття майстра.

Таким чином, екстремальна волатильність трансформує простір цифрових сервісів із зони ефективного розподілу праці на зону підвищеного цифрового хаосу (ентропії). Традиційні архітектурні патерни, що оперують виключно предикатами наявності фізичного ресурсу, виявляються безпорадними, оскільки при нульовому значенні пропозиції вони здатні лише генерувати повідомлення про помилку. Це зумовлює необхідність створення принципово нових математичних та алгоритмічних моделей, які розглядатимуть відсутність фізичного виконавця не як фатальну помилку системи, а як особливий стан мережі, що вимагає негайної інтелектуальної реконфігурації та інформаційного заміщення.

1.3 Огляд та порівняльний аналіз існуючих рішень та їхні структурні обмеження

Дослідження шляхів підвищення структурної живучості цифрових сервісних платформ неможливе без ретроспективного та архітектурного аналізу існуючих на ринку рішень. Сучасний ринок платформних послуг у сегментах C2C та B2C представлений широким спектром програмних продуктів, які можна класифікувати як системи узгодження попиту та пропозиції (matching platforms) [5]. Серед найбільш репрезентативних прикладів світового рівня доцільно виділити платформи TaskRabbit, Thumbtack, Upwork, а також соціально-орієнтовані мережі координації на кшталт Meetup. В українському сегменті найтипівішим представником такого класу систем є сервіс Kabanchik.ua, який концептуально дублює західні аналоги та адаптує їх до локального ринку.

Аналіз архітектури цих систем засвідчує, що всі вони будувалися на базі оптимістичної моделі функціонування ринку. Ця модель виходить із макроекономічної гіпотези про високу еластичність пропозиції та сталий надлишок ресурсів. Розробники закладали в основу бізнес-логіки передумову про те, що на будь-який сформований запит клієнта завжди знайдеться пул вільних виконавців, які в умовах ринкової конкуренції змагатимуться за право надати послугу.

Відповідно до цієї парадигми, базовий алгоритм маршрутизації та взаємодії в таких системах є строго лінійним і спирається на класичний транзакційний шлях «Потреба → Пошук → Виконавець → Результат». У цьому ланцюгу платформа відіграє роль виключно інформаційного посередника-брокера. Вона агрегує запити, застосовує фільтри геолокації чи рейтингу та відображає список доступних фізичних агентів. Проте, при перенесенні таких архітектур у середовище екстремальної волатильності та жорсткого дефіциту кадрів, виявляється їхня фундаментальна нежиттєздатність.

Детальний деконструктивний аналіз дозволяє виділити три головні структурні недоліки існуючих систем, які стають критичними в умовах волатильності.

Низька структурна «живучість» (Structural Robustness) та вразливість до єдиної точки відмови. Архітектура традиційних сервісних платформ є глибоко вразливою до відмов на рівні виконавчого пулу. З точки зору теорії надійності систем, процес надання послуги в таких додатках має послідовну топологію. Для побутових чи соціальних запитів відмова системи при раптовому зникненні людського ресурсу є фатальною та незворотною. Платформи типу TaskRabbit чи локальні аналоги не володіють механізмами «м'якої деградації» (graceful degradation), за яких система здатна функціонувати в обмеженому режимі. Користувач сприймає систему як єдине безперервне ціле, тому неспроможність платформи знайти фізичного майстра миттєво конвертується у повну втрату довіри до цифрового інструменту. Відбувається повний колапс сесії: відсутність вузла-виконавця не компенсується резервними підсистемами, що свідчить про нульову структурну надмірність (redundancy) для таких категорій завдань.

Проблема «критичного розриву» (Critical Gap) та бінарна логіка маршрутизації. Другий фундаментальний недолік лежить у площині математичної логіки, на якій базуються алгоритми цих платформ. Вони працюють на основі строгого предикату наявності ресурсу. Якщо розглядати процес як відображення задачі на виконавця, система перевіряє булеву умову: чи існує доступний агент для конкретного підзавдання (предикат $R \in \{0,1\}$). Якщо фізичний виконавець присутній і вільний ($R = 1$), алгоритм відпрацьовує штатно. Проте, якщо фізичний виконавець відсутній (предикат дорівнює нулю), логічний ланцюжок надання послуги розривається на рівні програмного ядра. Система генерує виняток (exception), після чого користувач отримує стандартне повідомлення про помилку («На жаль, виконавців не знайдено») або просто

пусту сторінку з результатами пошуку. У цей момент користувач залишається зі своєю проблемою віч-на-віч. Така бінарна транзакційна логіка є неприйнятною для критичних ситуацій (наприклад, аварійних станів інфраструктури, дефіциту вузькопрофільних спеціалістів під час воєнного стану), оскільки вона повністю ігнорує можливість надання компромісних, проміжних або консультативних рішень.

Відсутність механізмів інформаційної компенсації та когнітивних контурів. Традиційні платформи проектувалися виключно як каталоги ресурсів, а не як системи управління знаннями. Вони не мають вбудованих резервних когнітивних контурів (баз знань, генеративних моделей або експертних підсистем). Це означає, що система не вміє акумулювати та транслювати експертизу в разі відсутності самого експерта. Якщо майстра не знайдено, існуючі маркетплейси не здатні запропонувати користувачеві альтернативний, алгоритмізований спосіб вирішення проблеми. Сучасний рівень розвитку технологій штучного інтелекту вже дозволяє генерувати динамічні інтелектуальні інструкції для самостійного виконання (DIY) завдань. Однак класичні сервіси позбавлені архітектурних модулів, які могли б здійснити ін'єкцію знань у момент розриву транзакції. Відсутність механізму автоматичної генерації покрокового гайду, підкріпленого релевантним медіа-контентом, позбавляє такі платформи гнучкості, роблячи їх морально застарілими перед викликами екстремальної волатильності.

Підсумовуючи порівняльний аналіз, можна констатувати, що існуючі рішення не можуть бути адаптовані до нестабільних умов шляхом простого оновлення алгоритмів пошуку чи зміни інтерфейсу. Їхні обмеження закладені на фундаментальному архітектурному рівні. Екстенсивний розвиток таких платформ, орієнтований лише на залучення більшої кількості виконавців, вичерпав себе в умовах об'єктивного дефіциту кадрів. Це безальтернативно зумовлює необхідність проектування систем нового типу – гібридних платформ, де розрив фізичного сервісного ланцюга

автоматично компенсується підключенням інтелектуальних когнітивних агентів.

1.4 Огляд фреймворків для розробки мультиагентних систем штучного інтелекту

З огляду на необхідність інтеграції когнітивних модулів у процеси обслуговування, доцільно проаналізувати існуючі програмні фреймворки для побудови мультиагентних ШІ-систем. На сучасному етапі розвитку інженерії штучного інтелекту виділяють кілька провідних рішень, здатних керувати графами завдань: LangChain, AutoGen та CrewAI.

LangChain – це потужний інструментарій, орієнтований на створення ланцюжків викликів великих мовних моделей (LLM) [6]. Він надає широкі можливості для інтеграції моделі із зовнішніми джерелами даних (RAG-системи) та інструментами пошуку. Однак, архітектура LangChain є надмірно абстрагованою для систем із жорстким детермінованим плануванням. Її використання призводить до значних накладних витрат обчислювальних ресурсів, оскільки фреймворк намагається пропускати кожен крок логічного висновку через масивні LLM, що в умовах гібридної системи призводить до затримок (high latency).

AutoGen – розробка компанії Microsoft, яка дозволяє створювати системи, де кілька ШІ-агентів спілкуються між собою для вирішення задачі [7]. Хоча AutoGen чудово справляється з генерацією коду та дискусійними форматами, його фундаментальним недоліком у контексті платформ послуг є орієнтація виключно на взаємодію «Machine-to-Machine». Фреймворк не має нативних механізмів безшовної інтеграції біологічних агентів (людей-виконавців) у свій граф виконання, що суперечить гібридній парадигмі системи TeamFinder.

CrewAI – фреймворк, побудований навколо концепції рольових агентів, що виконують завдання послідовно або ієрархічно [8]. Він є більш

структурованим, ніж LangChain, але його модель делегування завдань є статичною. CrewAI не підтримує динамічну реконфігурацію графа в режимі реального часу (HTN-планування), яка необхідна при спрацьовуванні ентропійного фільтра, коли фізичний агент раптово стає недоступним.

Висновки щодо вибору технологічного стеку. Проведений аналіз існуючих ШІ-фреймворків доводить, що використання готових абстракцій (LangChain, AutoGen) накладає жорсткі архітектурні обмеження і не дозволяє реалізувати гнучкий алгоритм підстановки DIY-інструкцій замість людського ресурсу. Саме тому в даній роботі прийнято рішення відмовитись від монолітних ШІ-фреймворків на користь кастомної імплементації когнітивного агента. Використання мікросервісної маршрутизації в поєднанні з легкою моделлю Sentence-BERT для швидкої семантичної векторизації (пошуку релевантних інструкцій) та ієрархічним графом завдань дозволить досягти максимальної продуктивності та відмовостійкості системи.

1.5 Постановка задачі

З огляду на виявлені фундаментальні недоліки існуючих сервісних платформ та їхню неспроможність функціонувати в умовах екстремальної волатильності, виникає об'єктивна необхідність зміни самої парадигми надання послуг. Традиційна транзакційній модель, що спирається виключно на фізичну присутність виконавця, вичерпала свій потенціал масштабованості в кризових умовах. Як наслідок, виникає нагальна потреба у створенні нового покоління систем так званого «простору довіри». Такі системи повинні інтегрувати класичні функції пошуку фахівців із сучасними механізмами інтелектуальної підтримки, перетворюючись із пасивних каталогів на проактивних когнітивних асистентів.

Головною задачею кваліфікаційної роботи є розробка математичної моделі та архітектури інтелектуальної гібридної системи TeamFinder, яка

здатна до автономної реконфігурації в умовах дефіциту ресурсів. Під гібридністю в даному контексті розуміється здатність системи безшовно поєднувати взаємодію фізичних агентів (людей) та когнітивних програмних модулів (штучного інтелекту) в рамках єдиного процесу обслуговування.

Для досягнення цієї глобальної мети в процесі виконання дослідження необхідно вирішити такі ключові підзадачі, які охоплюють математичний, алгоритмічний та програмний рівні проектування.

Розробка мультиагентної математичної моделі сервісного середовища на базі динамічних графів та ентропійного фільтра. Першою підзадачею є формалізація процесу поширення та обробки запитів у гібридному середовищі. Мережу взаємодій необхідно представити у вигляді динамічного графа $G_t = (V, E)$, де множина вершин включає як користувачів, так і сервісних агентів різної природи. Оскільки стан такої мережі є нестабільним, необхідно розробити математичний апарат «ентропійного фільтра». Цей інструмент повинен безперервно оцінювати міру хаосу (ентропію) у конкретних підграфах послуг, аналізуючи доступність кадрів, інфраструктурні чинники та енергетичні графіки. Розробка такої моделі дозволить системі кількісно вимірювати власну стабільність у режимі реального часу і приймати обґрунтовані рішення про необхідність інтелектуального втручання.

Визначення критичних умов виникнення «фазового переходу» та ініціації процесів функціонального самовідновлення. Друга підзадача полягає у визначенні критичних математичних порогів, при яких традиційне обслуговування стає неможливим через деградацію пулу виконавців. Необхідно формалізувати умови виникнення «фазового переходу» – моменту, коли система переходить від пасивного очікування ресурсу до активного самовідновлення зв'язності графа. На цьому етапі система має ідентифікувати, що стандартний пошук фізичного виконавця є марним (ентропія перевищує допустимий поріг H_{max}), і замість генерації

системної помилки про відсутність фахівців повинна ініціювати процедуру «м'якої деградації» (graceful degradation) сервісу.

Розробка двохстадійного алгоритму обслуговування запитів на базі апарату ієрархічних мереж завдань (HTN-планування). Третя підзадача фокусується на алгоритмічному рівні управління процесами декомпозиції складних замовлень. Необхідно розробити алгоритм, який трансформує високорівневий запит користувача, сформульований природною мовою, у керований орієнтований ациклічний граф задач (DAG). Застосування апарату HTN-планування має забезпечити гнучку динамічну маршрутизацію: якщо на певному етапі виконання плану призначений фізичний агент раптово стає недоступним, алгоритм повинен не скасовувати все замовлення, а здійснювати локальне перепланування пошкодженої гілки графа, підставляючи резервний когнітивний ресурс.

Проектування та створення механізму автономної реконфігурації (DIY Mode) з використанням методів семантичної векторизації Sentence-BERT. Четверта підзадача має практичний програмний характер і спрямована на реалізацію модуля ін'єкції знань. Необхідно спроектувати когнітивний контур, який забезпечить динамічну підстановку ШІ-асистента (експертної бази знань та інтерактивних покрокових інструкцій) замість відсутнього фізичного виконавця. Для подолання семантичного розриву між запитом користувача та довідковою базою платформи необхідно застосувати трансформерні моделі Sentence-BERT для обчислення косинусної близькості векторів у багатовимірному просторі ембедингів. Організація доступу до контексту має спиратися на Model Context Protocol (MCP) для уникнення ефекту «галюцинацій» моделі.

Експериментальна верифікація та оцінка структурної живучості системи. П'ятою підзадачею є проведення імітаційного моделювання розробленої системи TeamFinder в умовах симуляції масових відмов сервісного ланцюга (за принципами Chaos Engineering). Необхідно оцінити

працездатність системи за допомогою метрик Service Rejection Rate (SRR) та Resolution Rate (RR) при різних сценаріях волатильності середовища, довівши перевагу гібридної архітектури над класичними рішеннями.

Об'єктом дослідження є процес функціонування цифрових сервісних платформ в умовах екстремальної волатильності та дефіциту ресурсів.

Предметом дослідження виступають методи автономної реконфігурації гібридних систем та заміщення фізичних вузлів когнітивними модулями.

Методи дослідження базуються на використанні апарату теорії графів, мультиагентного моделювання, ієрархічного мережевого планування (HTN) та методів обробки природної мови (NLP) з використанням сіамських нейромережових архітектур.

2 ПРОЄКТУВАННЯ ГІБРИДНОЇ СИСТЕМИ TEAMFINDER

2.1 Архітектура гібридної системи та концепція автономної реконфігурації

Проектування сучасних інформаційних систем, орієнтованих на надання послуг в умовах нестабільного та волатильного середовища, вимагає кардинального перегляду класичних архітектурних патернів. Традиційні монолітні або навіть розподілені мікросервісні архітектури цифрових платформ здебільшого базуються на жорстко детермінованих алгоритмах маршрутизації. У таких системах процес обслуговування безальтернативно прив'язаний до наявності фізичного виконавця на кінцевому вузлі. Класичні мікросервіси чудово справляються з відмовами апаратного чи програмного забезпечення (шляхом балансування навантаження або підняття нових інстансів серверів), проте вони абсолютно не здатні обробляти «біологічні відмови» – ситуації, коли програмний вузол працює штатно, але фізична людина-агент за ним стає недоступною.

Для подолання цього фундаментального обмеження в основу розроблюваної системи TeamFinder закладено концепцію гібридної архітектури з можливістю автономної реконфігурації. Гібридність у контексті даного дослідження розглядається не просто як поєднання різних технологічних стеків, а як глибока структурна інтеграція гетерогенних суб'єктів: біологічних агентів (Human-to-Human, H2H) та когнітивних програмних модулів (Human-to-Machine, H2M) у межах єдиного безперервного сервісного ланцюга. Архітектура TeamFinder абстрагує поняття «Виконавець», перетворюючи його з конкретної фізичної особи на загальний інтерфейс ресурсу, імплементацією якого може виступати як людина, так і штучний інтелект.

На відміну від класичних сервісів, система проектується з урахуванням принципу структурної живучості (structural robustness).

Структурна живучість у цьому контексті означає здатність програмного комплексу зберігати свою логічну цілісність та продовжувати виконання цільової функції (задоволення запиту користувача) навіть за умови каскадних відмов окремих сегментів мережі. В умовах дефіциту кадрів та постійних інфраструктурних перебоїв система повинна вміти абсорбувати шок від втрати вузлів, здійснюючи процедуру так званої «м'якої деградації», коли замість повної зупинки бізнес-процесу відбувається контрольоване зниження рівня надання послуги, але транзакція не скасовується.

Ключовою інновацією запропонованої архітектури TeamFinder є новий алгоритмічний підхід до обробки відсутності ресурсу. У разі відсутності фізичного сервісного вузла (людини-виконавця), архітектура системи не допускає генерації критичної помилки (Exception) і переривання процесу обслуговування. Замість того, щоб переривати користувацьку сесію та повертати клієнта на стартовий екран пошуку, ядро системи переводить поточний запит у стан тимчасового очікування на рівні модуля маршрутизації та паралельно активує тригер оцінки доступності ресурсів.

Отримавши від мікросервісу метчінгу сигнал про неможливість знайти фізичного виконавця у заданому радіусі чи часовому вікні, система здійснює автономну реконфігурацію мережі, підставляючи замість відсутнього елемента когнітивний модуль підтримки (ШІ-асистента). Цей процес відбувається прозоро для користувача і на рівні коду реалізується через патерн впровадження залежностей (Dependency Injection) або патерн «Стратегія» (Strategy Pattern) [9]. Інтерфейс IServiceProvider, який відповідає за надання послуги, динамічно змінює свою імплементацію з класу PhysicalAgent на клас CognitiveAgent.

Для формалізації функціональних вимог до розроблюваної системи TeamFinder та визначення меж взаємодії між гетерогенними суб'єктами спроектовано діаграму прецедентів (рисунок 2.1). Основними акторами у мультиагентному середовищі виступають три типи сутностей: агент-користувач (UserAgent), який ініціює запити та потребує інформаційної чи

фізичної підтримки; фізичний виконавець (PhysicalServiceProvider), який безпосередньо забезпечує закриття сервісних функцій за умов стабільності; та сумервізорний когнітивний агент (CognitiveServiceProvider), який активується для інформаційної компенсації.

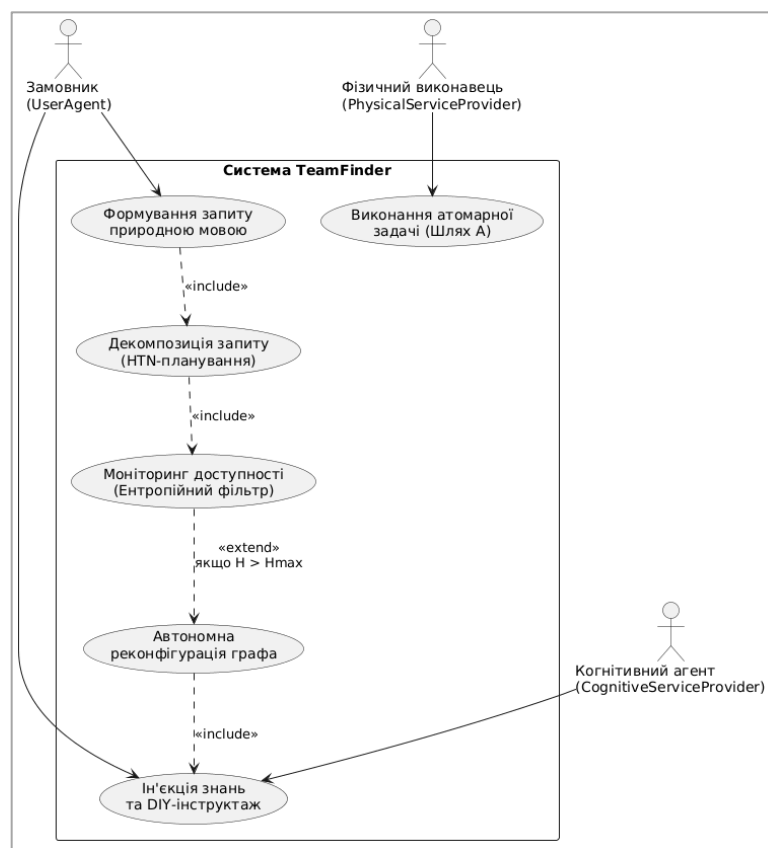


Рисунок 2.1 – Діаграма прецедентів системи TeamFinder

На діаграмі відображено базові прецеденти, включаючи процеси ініціації запиту природною мовою, його автоматичну декомпозицію засобами HTN-планування, а також критично важливі системні процеси моніторингу топології за допомогою ентропійного фільтра. Зв'язки розширення (extend) та включення (include) математично описують умови, за яких відбувається фазовий перехід системи до автономної реконфігурації графа задач та ін'єкції знань у режимі підтриманої самодопомоги (DIY

Mode). Завдяки такій динамічній підстановці вектор виконання задачі змінюється, але логічний ланцюг залишається цілісним.

Таким чином, розрив сервісного ланцюга трансформується у протокол інформаційної компенсації, переводячи користувача в режим підтриманої самодопомоги (DIY Mode). Протокол інформаційної компенсації являє собою не просто статичний текст чи набір посилань на FAQ, а динамічно згенерований, структурований масив даних, що формується у режимі реального часу на основі первинного запиту замовника.

Режим підтриманої самодопомоги (DIY Mode) в архітектурі TeamFinder функціонує як повноцінний інтерактивний контур. Когнітивний модуль підтримки, отримавши керування після реконфігурації графа, ініціює діалогову сесію з користувачем. Він аналізує контекст проблеми (наприклад, тип поломки, наявні у користувача інструменти, рівень небезпеки) і генерує адаптивний покроковий алгоритм вирішення проблеми власними силами. Кожен крок супроводжується контролем виконання: система очікує підтвердження від користувача, перш ніж надати наступну інструкцію.

Такий архітектурний підхід радикально підвищує загальну відмовостійкість сервісу. Оскільки когнітивний ресурс (штучний інтелект та база знань), на відміну від фізичного, володіє властивістю нескінченної масштабованості і не піддається впливу демографічних чи логістичних обмежень, система TeamFinder здатна підтримувати високий рівень корисності для клієнта незалежно від ступеня волатильності зовнішнього середовища.

2.2 Мультиагентна математична модель та ентропійний фільтр

У межах наукового дослідження архітектури системи TeamFinder, її формалізація як математичного об'єкта є необхідною умовою для забезпечення верифікованості прийнятих рішень. Система функціонує як

динамічний граф $G_t = (V, E)$, де параметри вершини V та ребер E змінюються у часі t залежно від стану зовнішнього середовища [10]. Така динамічна графова модель дозволяє описувати складні топологічні трансформації мережі сервісних вузлів у реальному часі.

Множина агентів системи A , що здійснюють взаємодію у цьому графі, класифікується як $A = \{A_u, A_s, A_{ai}\}$. Кожен з цих класів агентів володіє специфічними атрибутами та функціональними обов'язками в рамках мультиагентної моделі:

- A_u (агенти-користувачі). Це множина вузлів-замовників, які генерують запити на обслуговування. Кожен агент цього типу характеризується набором параметрів потреби Q , що включає зміст запиту, часові обмеження та географічну локацію;

- A_s (агенти сервісних вузлів). Це фізичні виконавці або наявні ресурси. В умовах волатильності ці вузли є ненадійними – їхня функція доступності $f_a(t)$ змінюється від 0 (вузол недоступний) до 1 (вузол готовий до виконання завдання);

- A_{ai} (супервізорний когнітивний агент). Це центральний когнітивний вузол системи, який відповідає за прийняття рішень щодо реконфігурації мережі. Він виконує роль «розумного посередника», який аналізує стан системи та ініціює процедуру заміщення вузлів A_s на когнітивні модулі.

Для відображення статичної структурної архітектури програмного комплексу, деталізації ієрархії сутностей та зв'язків між ними розроблено об'єктно-орієнтовану діаграму класів системи TeamFinder (рисунк 2.2). В основу проектування класів покладено принципи SOLID та подійно-орієнтовані патерни, що гарантує слабку зв'язність (loose coupling) компонентів.

Базовим елементом архітектури є абстрактний клас BaseAgent, який визначає уніфікований інтерфейс обміну повідомленнями для всіх суб'єктів мережі. Від нього наслідуються класи UserAgent, PhysicalServiceProvider та CognitiveServiceProvider.

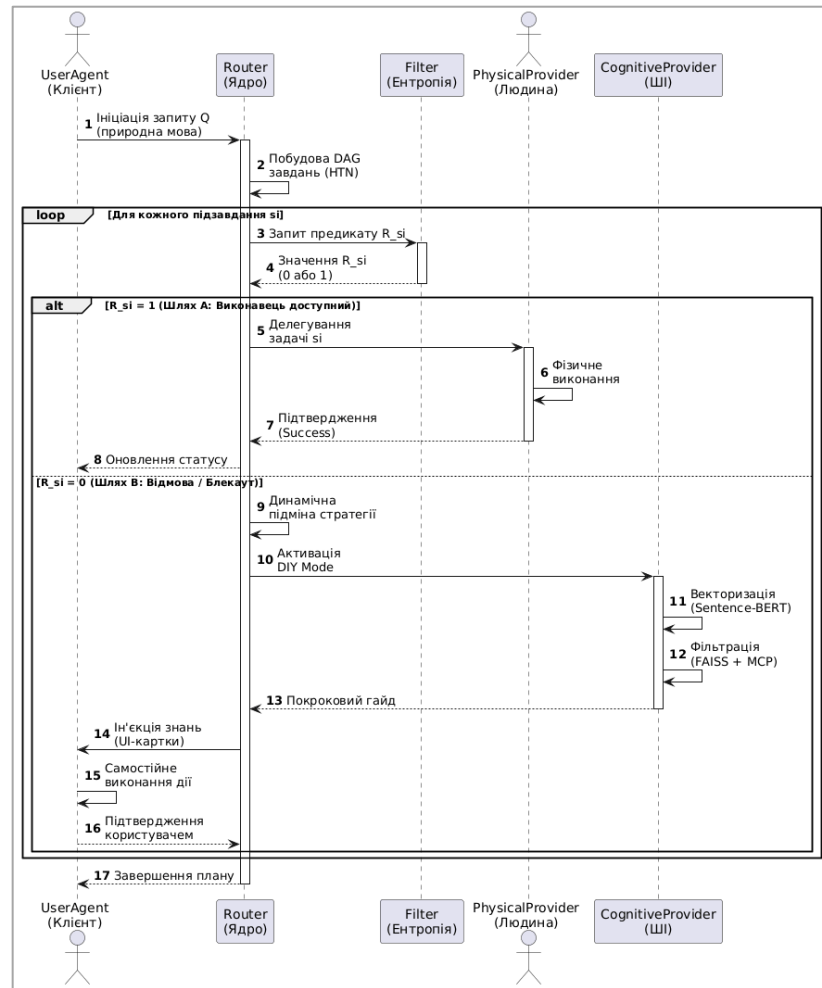


Рисунок 2.2 – Діаграма класів об'єктної моделі

Ключовим архітектурним рішенням для забезпечення безперервності процесів є введення інтерфейсу `IServiceProvider`. Клас `TeamFinderRouter` оперує саме цим абстрактним інтерфейсом, реалізуючи патерн «Стратегія» (Strategy Pattern). Це дозволяє на рівні виконання DFS-обходу графа динамічно підміняти об'єкт фізичного провайдера когнітивним ШІ-провайдером, який інкапсулює роботу з клієнтом Model Context Protocol (MCP) та векторним індексом FAISS.

Найбільш критичною частиною запропонованої моделі є введення так званого «ентропійного фільтра». В умовах екстремальної волатильності, коли стан фізичних виконавців стає стохастичною величиною, система

потребує інструменту, який оцінюватиме рівень невизначеності мережі [11]. Ентропійний фільтр безперервно оцінює ймовірність доступності сервісних вузлів A_s , аналізуючи низку зовнішніх індикаторів: енергетичні графіки (наявність живлення у виконавців), регіональну активність та міграційні звіти.

Математично, ентропія підграфа послуг H визначається як міра непередбачуваності доступності сервісного сегмента. Якщо ентропія конкретного підграфа послуг перевищує встановлений критичний поріг H_{max} , система ініціює процедуру «м'якої деградації». Це означає, що система автоматично переходить від прямого пошуку фізичного виконавця (шлях А) до генерації інтелектуальних інструкцій (шлях В – DIY Mode).

Вибір порогу H_{max} є динамічним параметром. Він визначає той фазовий перехід, після якого класична модель «пошук – результат» перестає бути оптимальною, оскільки ймовірність успішного завершення транзакції стає настільки низькою, що витрати на пошук (обчислювальні та часові) перевищують вигоду від залучення людини. Ентропійний фільтр, таким чином, виступає як інтелектуальний перемикач, що запобігає повному колапсу сервісу: замість того, щоб демонструвати користувачеві «помилку відсутності», система змінює свою власну структуру, перетворюючись із брокера послуг на генератор знань.

Цей математичний підхід дозволяє перетворити ентропію системи на корисний ресурс. У звичайних системах висока ентропія (хаос) є причиною збоїв, тоді як у системі TeamFinder висока ентропія є сигналом до активізації когнітивного агента A_{ai} . Таке функціональне самовідновлення забезпечується тим, що розриви у графі G_t , які виникають через відсутність вузлів A_s , миттєво компенсуються введенням у граф темпоральних (тимчасових) вузлів-замінників, які містять інтелектуальний контент.

Таким чином, розроблена модель не просто намагається підтримувати стабільність мережі, а свідомо приймає її деградацію як частину еволюційного процесу, автоматично трансформуючи структурні розриви у нові формати надання інформаційної підтримки. Це доводить здатність системи зберігати свою функціональну цілісність навіть у найнесприятливіших умовах функціонування.

2.3 Двохстадійний алгоритм обслуговування на базі HTN-планування

Ефективне функціонування гібридної системи в умовах екстремальної волатильності вимагає не лише надійної математичної моделі оцінки станів, але й потужного алгоритмічного апарату для маршрутизації та виконання запитів. Класичні системи обслуговування здебільшого оперують плоскими (однорівневими) списками завдань, де користувач самостійно розбиває свою комплексну проблему на дрібні кроки і для кожного кроку шукає окремого виконавця. Такий підхід перекладає весь когнітивний тягар планування на замовника, що є неефективним і призводить до частих помилок. Для вирішення цієї проблеми двухстадійний алгоритм обслуговування в розроблюваній системі TeamFinder концептуально спирається на математичний апарат ієрархічних мереж завдань (Hierarchical Task Network, HTN) [12].

HTN-планування — це парадигма штучного інтелекту, яка працює шляхом декомпозиції комплексних високорівневих завдань на менші підзавдання (атомарні дії). Цей підхід є особливо ефективним способом координації в динамічних мультиагентних середовищах, оскільки він дозволяє моделювати процес обслуговування так, як це робить людина-експерт. В ієрархічних мультиагентних системах (HMAS) застосування HTN дозволяє чітко розподіляти ролі, делегувати завдання між різними агентами та контролювати послідовність їх виконання.

Стадія 1. Декомпозиція та побудова графа завдань.

На першій стадії алгоритму відбувається обробка складного запиту Q , сформульованого користувачем природною мовою. Цей процес реалізується як складна математична функція відображення (2.1).

$$f: Q \rightarrow G, \quad (2.1)$$

де $G = (V, E)$ – це орієнтований ациклічний граф (Directed Acyclic Graph, DAG).

У цьому графі:

- множина вершин V відповідає атомарним підзавданням s_i . Наприклад, якщо глобальний запит Q полягає у ремонті ванної кімнати, то атомарними завданнями будуть: s_1 (демонтаж старої плитки), s_2 (підготовка поверхні), s_3 (виклик електрика для розводки), s_4 (монтаж нової плитки);

- множина ребер E визначає логічне відношення передування $s_i < s_j$, що означає неможливість початку виконання завдання s_j до успішного завершення завдання s_i .

Побудувавши такий граф, система де-факто перебирає на себе централізоване управління графом задач, виконуючи роль цифрового «Виконроба». Замість того, щоб шукати одного абстрактного майстра-універсала, система готується залучити мультиагентний пул виконавців для кожного конкретного вузла s_i .

Стадія 2. Маршрутизація та підтримка живучості виконання плану.

Після формування графа G , алгоритм переходить до стадії виконання. Обхід графа здійснюється за допомогою модифікованого алгоритму пошуку в глибину (Depth-First Search, DFS) [13], який гарантує дотримання топологічного сортування підзадач. В умовах екстремальної волатильності критично важливою стає підтримка живучості виконання плану (Plan Execution Robustness).

Подальший шлях обробки кожного атомарного завдання розгалужується залежно від значення предикату наявності

ресурсу (доступності фізичного виконавця) $R_{si} \in \{0,1\}$. Цей предикат безперервно оновлюється на основі даних від ентропійного фільтра. Залежно від стану предикату, алгоритм імплементує один із двох сценаріїв розвитку подій.

Шлях А. Класична транзакційна схема, $R = 1$.

Якщо предикат дорівнює одиниці, це означає, що фізичний виконавець (агент із множини A_s) для конкретного атомарного завдання – доступний. У цьому випадку система функціонує за класичною схемою «Потреба $\rightarrow$ Пошук $\rightarrow$ Виконавець $\rightarrow$ Результат». Агенту делегується підзавдання, і після підтвердження його виконання система маркує вершину s_i як пройдену, дозволяючи алгоритму DFS перейти до наступної вершини s_{i+1} .

Шлях В. Гібридний режим та автономна реконфігурація, $R = 0$.

У випадку, коли ланка виконавця випадає (через блекаут, відсутність кадрів чи інші фактори екстремальної волатильності), предикат набуває значення нуля ($R = 0$). У традиційних системах на цьому етапі класичний ланцюжок розривається, і виконання всього графа G зупиняється з критичною помилкою.

Проте архітектура TeamFinder пропонує інноваційний підхід: при критичному дефіциті людських кадрів збереження плану досягається за рахунок залучення когнітивного ШІ-агента A_{ai} , який компенсує відсутність фізичного виконавця. Щоб користувач не залишився з проблемою віч-на-віч, традиційний пошук у глибину (DFS) тимчасово зупиняється, алгоритм блокує генерацію системної помилки і натомість активується режим Smart Assistant (DIY Mode).

Це забезпечує безперервність процесів та функціональне самовідновлення системи на рівні виконання складних планів. Фізичний вузол у графі динамічно заміщується темпоральним (тимчасовим) інформаційним вузлом, який надає користувачеві необхідний інструктаж для самостійного проходження цього атомарного етапу. Після того як

користувач самостійно виконує дію і підтверджує її завершення, вузол вважається пройденим, і система успішно продовжує обхід графа, переходячи до наступних підзавдань, для яких фізичні виконавці можуть бути знову доступні. Такий підхід нівелює ефект «каскадних відмов» і дозволяє зберегти живучість всього плану дій незалежно від стохастичних коливань ринку послуг.

Часова динаміка взаємодії компонентів системи TeamFinder та логіка алгоритму підтримки живучості планів під час виникнення критичних розривів сервісного ланцюга відображена на діаграмі послідовності (рисунок 2.3). Діаграма деталізує життєвий цикл обробки транзакції в умовах екстремальної волатильності, демонструючи розподіл подій для двох взаємовиключних сценаріїв (Шлях А та Шлях В).

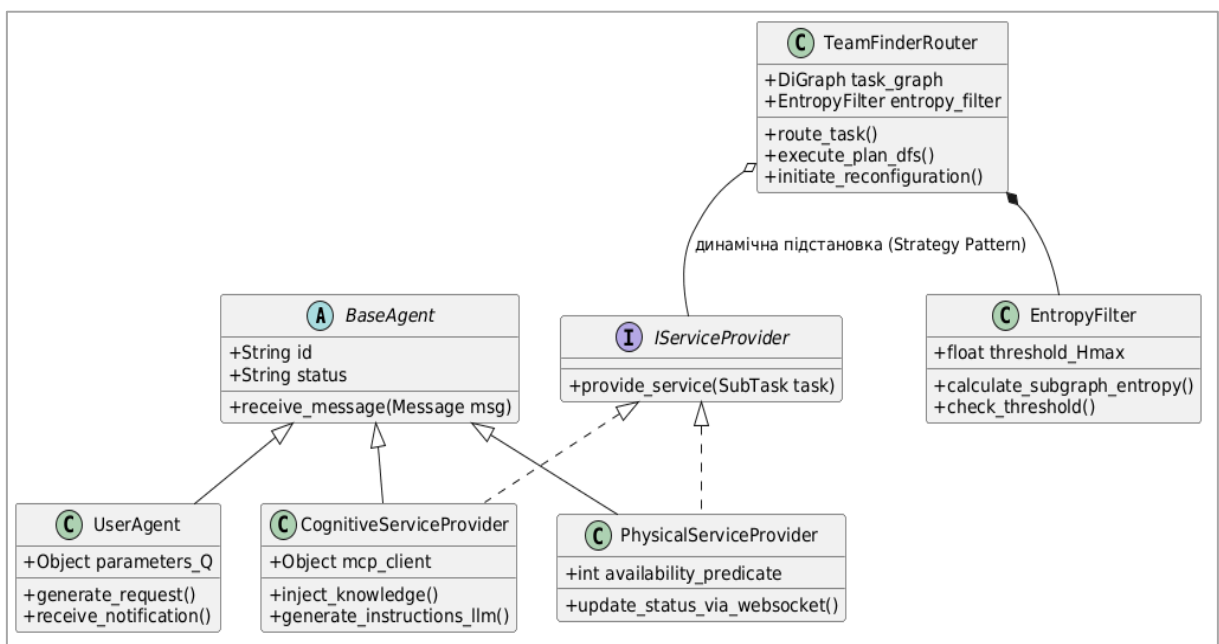


Рисунок 2.3 – Діаграма послідовності процесу реконфігурації ланцюга

У процесі ітераційного обходу орієнтованого ациклічного графа завдань (DAG) ядро маршрутизації здійснює динамічний запит до ентропійного фільтра для перевірки значення предикату наявності ресурсу R_{si} . За умови $R_{si} = 1$ (Шлях А) взаємодія відбувається за класичною схемою із залученням фізичного виконавця.

У випадку фіксації аномалії або дефіциту кадрів ($R_{si} = 0$, Шлях В), діаграма наочно ілюструє механізм перемикання стратегій за допомогою патерну Dependency Injection. Замість аварійної зупинки, керування безшовно передається до CognitiveServiceProvider, який через семантичну векторизацію Sentence-BERT та фільтрацію FAISS/MCP розгортає для користувача інтерактивний DIY-контур.

2.4 Проектування когнітивного модуля підтриманої самодопомоги (DIY Mode) на базі Sentence-BERT

Активація гібридного шляху обслуговування (Шляху В) у момент виникнення критичного розриву сервісного ланцюжка потребує залучення потужного інструментарію інтелектуальної підтримки. Перехід користувача в режим підтриманої самодопомоги (DIY Mode) не може бути реалізований через просте відображення статичних довідкових матеріалів, оскільки неструктуровані текстові масиви не здатні врахувати унікальний контекст та специфіку індивідуального запиту. Для забезпечення високої точності та адаптивності інформаційного заміщення, когнітивний модуль системи TeamFinder проектується з використанням передових методів обробки природної мови (Natural Language Processing) та семантичного аналізу тексту.

Сучасні великі мовні моделі (LLM) демонструють високу ефективність при виконанні модульної декомпозиції завдань та координації процесів у динамічних мультиагентних середовищах. У системі TeamFinder операція ін'єкції знань, яка виконується супервізорним когнітивним агентом A_{ai} , базується на семантичній векторизації завдань за допомогою неймережевої архітектури Sentence-BERT [14]. Цей підхід дозволяє трансформувати текстовий опис підзавдання у щільний числовий вектор (ембединг) фіксованої розмірності, який відображає реальний зміст та семантику фрази, незалежно від конкретних використаних слів чи

синтаксичних конструкцій. Це забезпечує високу семантичну точність (semantic fidelity), зберігаючи специфічні концептуальні зв'язки при обробці спеціалізованих запитів користувача. Процес ін'єкції знань та автономної реконфігурації графа на базі косинусної фільтрації складається з трьох послідовних математично формалізованих кроків.

Крок 1. Семантична векторизація.

За допомогою попередньо натренованої сіамської архітектури Sentence-BERT визначається семантичний вектор атомарного завдання v_{si} . Цей вектор стає першоджерелом семантичної інформації і виступає у розрахунках як вектор-запит v_q (2.2).

$$v_q = \text{S-BERT}(s_i). \quad (2.2)$$

Крок 2. Косинусна фільтрація.

Система здійснює інтелектуальний пошук у заздалегідь сформованій експертній базі знань платформи. Алгоритм порівнює обчислений вектор-запит v_q з векторами еталонних категорій інструкцій v_{cat} , які зберігаються у векторному індексі. Ступінь релевантності та смислової близькості між ними обчислюється за формулою косинусної близькості (2.3).

$$\text{similarity} = \cos(\theta) = \frac{v_q \cdot v_{cat}}{\|v_q\| \cdot \|v_{cat}\|}, \quad (2.3)$$

де $v_q \cdot v_{cat}$ – скалярний добуток векторів;

$\|v_q\|$ та $\|v_{cat}\|$ – їхні відповідні евклідові норми.

Крок 3. Підстановка вузла-замінника.

На основі отриманих результатів обчислень система обирає ту еталонну категорію v_{cat} , для якої значення метрики similarity є максимальним і перевищує встановлений поріг впевненості. Знайдений структурований пак інструкцій (який може містити покрокові текстові

гайди, інтерактивні схеми, списки необхідних інструментів та лайфхаки) інтегрується в поточний орієнтований ациклічний граф $G = (V, E)$ як темпоральний (тимчасовий) вузол-замінник.

Для того щоб гарантувати високу надійність, гнучкість та модульність доступу когнітивного ІІІ-агента до зовнішніх розподілених баз знань, інструкцій та сервісів, архітектура системи спирається на передові принципи Model Context Protocol (MCP) [15]. Протокол MCP дозволяє стандартизувати комунікаційний шар між когнітивним ядром штучного інтелекту та інфраструктурою доступу до контексту даних.

Це створює чітке та архітектурно правильне розділення обов'язків всередині платформи: когнітивний агент повністю відповідає за високорівневу логіку, семантичний аналіз та планування (reasoning), тоді як сервери MCP забезпечують безпечний, ізольований та уніфікований доступ до актуальних даних, репозиторіїв та інструментів безпосередньо в момент виникнення потреби.

Завдяки такому генеративному «скафолдингу» за допомогою великих мовних моделей, система TeamFinder набуває здатності адаптивно адаптувати та динамічно розгортати покрокові інструкції під поточний стан користувача. Така динамічна підстановка моделі штучного інтелекту замість відсутнього фізичного елемента в режимі реального часу гарантує стовідсоткову структурну зв'язність сервісного графа. Як наслідок, забезпечується повне функціональне самовідновлення сервісу TeamFinder навіть за умов найжорсткішої деградації та тотального кадрового дефіциту на фізичному ринку послуг.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Обґрунтування вибору засобів та технологій розробки

Проектування інтелектуальної гібридної системи TeamFinder, яка поєднує в собі алгоритми обробки динамічних графів, мультиагентну взаємодію та нейромережеві моделі семантичного аналізу, вимагає ретельного і науково обґрунтованого підбору технологічного стеку. Зважаючи на специфіку архітектури, розробленої у другому розділі, класичні інструменти для створення монолітних веб-додатків є недостатніми. Обрані технології повинні забезпечувати високу продуктивність при роботі з неструктурованими даними, підтримку асинхронної обробки подій та можливість безшовної інтеграції апарату штучного інтелекту.

3.1.1 Мова програмування та базове середовище

В якості базової мови програмування для реалізації серверної частини (бекенду) та когнітивного ядра системи було обрано Python [16]. Вибір цієї мови обґрунтовується її беззаперечним лідерством у сфері Data Science та машинного навчання. Python володіє найширшою екосистемою бібліотек для побудови систем штучного інтелекту. Крім того, для реалізації двохстадійного алгоритму HTN-планування та обробки орієнтованих ациклічних графів (*DAG*) ідеально підходить спеціалізована бібліотека NetworkX. Вона дозволяє ефективно створювати графові структури в оперативній пам'яті, виконувати топологічне сортування підзадач та модифікувати зв'язки в режимі реального часу, що є критично важливим для роботи ентропійного фільтра.

3.1.2 Фреймворк для реалізації мікросервісної архітектури

Оскільки система TeamFinder повинна працювати в умовах екстремальної волатильності, її комунікаційний шар має бути максимально відмовостійким. Для створення API та маршрутизації мультиагентних запитів обрано фреймворк FastAPI. На відміну від традиційних синхронних фреймворків (наприклад, Django), FastAPI базується на сучасних стандартах ASGI (Asynchronous Server Gateway Interface) [17] і нативно підтримує асинхронність (async/await). Це означає, що у випадку критичного розриву ланцюга і тимчасової недоступності вузла, серверний потік не блокується в очікуванні відповіді, а продовжує обслуговувати інші атомарні завдання графа. Крім того, FastAPI автоматично генерує документацію OpenAPI, що спрощує подальшу інтеграцію платформи з клієнтськими додатками.

3.1.3 Засоби роботи зі штучним інтелектом та обробки природної мови

Для реалізації когнітивного модуля підтриманої самодопомоги (DIY Mode) та виконання семантичної векторизації (Крок 1 у підрозділі 2.4) застосовується екосистема Hugging Face Transformers. Вона надає оптимізований доступ до попередньо натренованих моделей архітектури Sentence-BERT. Для зберігання та швидкої косинусної фільтрації отриманих векторів (ембедингів) еталонних інструкцій використовується векторна база даних FAISS (Facebook AI Similarity Search) або інтегровані векторні розширення.

Організація взаємодії між когнітивним агентом (A_{ai}) та зовнішніми експертними базами знань реалізується за допомогою імплементації протоколу MCP (Model Context Protocol). Використання офіційного Python SDK для MCP дозволяє створити стандартизовані сервери контексту, які гарантують, що мовна модель отримуватиме лише верифіковані інструкції з

платформи, уникаючи проблеми «галюцинацій» при формуванні інформаційної компенсації для користувача.

3.1.4 База даних для динамічної топології

Традиційні реляційні бази даних (PostgreSQL, MySQL) є неефективними для роботи зі складною топологією сервісних ланцюгів, оскільки пошук альтернативних маршрутів у графах потребує ресурсомістких рекурсивних SQL-запитів (JOIN). З огляду на це, основним сховищем топологічних даних системи обрано графову СУБД Neo4j. Завдяки мові запитів Cypher, система отримує можливість миттєво знаходити критичні розриви (critical gaps) у графі завдань, розраховувати поточну ентропію підграфа виконавців та ініціювати реконфігурацію зв'язків. Агенти, користувачі та завдання зберігаються як вузли (Nodes), а їхні залежності – як ребра (Relationships), що математично точно відтворює розроблену в розділі 2 модель $G_t = (V, E)$.

3.1.5 Апаратне забезпечення та середовище для тестування (Інференсу)

Проведення імітаційного моделювання роботи системи та тестування ІІІ-асистентів вимагає відповідного середовища для розгортання. Оскільки операція семантичної векторизації за допомогою Transformer-моделей є математично інтенсивною (потребує швидкого множення матриць), розробка та тестування когнітивних модулів доцільно проводити з використанням технології паралельних обчислень CUDA.

Оптимальним вибором для локального інференсу моделей та обробки HTN-планів у реальному часі є використання сучасних графічних прискорювачів з архітектурою Ada Lovelace. Зокрема, використання рішень рівня RTX 40-ї серії (наприклад, RTX 4050 та вище) дозволяє задіяти

апаратні тензорні ядра четвертого покоління. Це гарантує мінімізацію затримок (latency) при генерації DIY-інструкцій та забезпечує стабільну роботу локальної мультиагентної симуляції під час розробки без необхідності постійного підключення до дорогих хмарних GPU-кластерів.

Отже, обраний комплекс програмних інструментів та апаратних рекомендацій утворює цілісний, сучасний стек технологій. Він повною мірою здатен забезпечити практичну реалізацію заявленої гібридної архітектури, розрахунок ентропійних фільтрів та функціональне самовідновлення сервісного графа.

3.2 Програмна реалізація мультиагентної взаємодії та бази знань

Перехід від теоретичної мультиагентної математичної моделі, описаної у розділі 2, до її програмної реалізації вимагає застосування сучасних патернів об'єктно-орієнтованого програмування (ООП) та подійно-орієнтованої архітектури (Event-Driven Architecture). У програмній реалізації системи TeamFinder архітектура взаємодії базується на принципах асинхронного обміну повідомленнями, що дозволяє системі ефективно обробляти велику кількість одночасних запитів без блокування головного потоку виконання.

3.2.1 Програмна ієрархія мультиагентної системи

Множина агентів $A = \{A_u, A_s, A_{ai}\}$, концептуально обґрунтована на етапі проєктування, на рівні коду реалізована через абстрактну ієрархію класів у мові Python. Базовим будівельним блоком виступає абстрактний клас `BaseAgent`, який інкапсулює загальні атрибути: унікальний ідентифікатор, поточний статус доступності (онлайн/офлайн/зайнятий) та інтерфейс для прийому повідомлень.

Від цього базового класу успадковуються три ключові сутності:

- клас `UserAgent` (A_u) репрезентує замовника. Він містить методи для формування запиту Q природною мовою та інтерфейси для отримання сповіщень про зміну статусу виконання завдання;

- клас `PhysicalServiceProvider` (A_s) репрезентує фізичного виконавця. Відмінною рисою цього класу є наявність динамічного предикату доступності R_{si} , який періодично оновлюється через механізм `WebSockets` або `Long Polling`. Якщо зв'язок з клієнтським додатком фізичного виконавця втрачається (наприклад, через відключення електроенергії), бекенд автоматично переводить його екземпляр класу в стан `offline`;

- клас `CognitiveServiceProvider` (A_{ai}) репрезентує супервізорного ШІ-агента. На відміну від фізичних виконавців, екземпляр цього класу завжди ініціалізується зі статусом `always_available`. Він містить інтеграційні методи для звернення до мовної моделі та бази знань через протокол MCP (`Model Context Protocol`).

Для реалізації безшовної підстановки одного агента іншим (процедура «м'якої деградації») використано патерн проектування «Стратегія» (`Strategy`). Інтерфейс обробки вузла графа очікує об'єкт типу `IServiceProvider`, який під час нормального функціонування ініціалізується об'єктом `PhysicalServiceProvider`. У момент виявлення критичного розриву інстанс автоматично підміняється на `CognitiveServiceProvider`, що дозволяє продовжити виконання коду без генерації системного винятку (`Exception`).

Програмна реалізація абстрактної ієрархії агентів та маршрутизатора `TeamFinderRouter` виконана мовою `Python` з використанням принципів об'єктно-орієнтованого програмування. Для забезпечення динамічної підміни фізичного виконавця на когнітивного ШІ-агента імплементовано патерн «Стратегія».

Фрагмент базового коду системи наведено у лістингу 3.1.

Лістинг 3.1 – Програмна реалізація ієрархії агентів та динамічної маршрутизації

```

from abc import ABC, abstractmethod
import logging

class IServiceProvider(ABC):
    @abstractmethod
    async def provide_service(self, task_context: dict) -> dict:
        pass

class PhysicalServiceProvider(IServiceProvider):
    def __init__(self, agent_id: str):
        self.agent_id = agent_id
        self.is_online = True

    async def provide_service(self, task_context: dict) -> dict:
        if not self.is_online:
            raise ConnectionError(f"Agent {self.agent_id} is offline.")
        logging.info(f"Task {task_context['id']} delegated to Physical Agent.")
        return {"status": "success", "executed_by": "human"}

class CognitiveServiceProvider(IServiceProvider):
    def __init__(self, nlp_model_path: str):
        self.model_path = nlp_model_path

    async def provide_service(self, task_context: dict) -> dict:
        logging.info("Initiating DIY Mode. Generating AI instructions...")
        # Логіка виклику LLM та Sentence-BERT

```

Продовження лістингу 3.1

```

        return {"status": "diy_mode_activated", "executed_by":
"ai"}

class TeamFinderRouter:
    def __init__(self):
        self.active_strategy: IServiceProvider = None

    def set_strategy(self, strategy: IServiceProvider):
        self.active_strategy = strategy

    async def execute_task(self, task_context: dict):
        try:
            return await
self.active_strategy.provide_service(task_context)
        except ConnectionError:
            logging.warning("Critical gap detected. Switching
to AI Provider.")

self.set_strategy(CognitiveServiceProvider(nlp_model_path="s-
bert"))

        return await
self.active_strategy.provide_service(task_context)

```

Як видно з лістингу 3.1, клас `TeamFinderRouter` не залежить від конкретної реалізації виконавця. Метод `execute_task` спочатку намагається обробити транзакцію через об'єкт `PhysicalServiceProvider`. Якщо виникає помилка з'єднання (наприклад, під час блекауту зв'язок втрачається миттєво, що особливо характерно для торговельних чи сервісних точок у локаціях зі складною інфраструктурою, як-от підземні переходи метрополітену), система перехоплює виняток `ConnectionError` і здійснює безшовну підміну об'єкта на `CognitiveServiceProvider`. Користувач не отримує критичної помилки системи, а бізнес-процес не переривається.

3.2.2 Реалізація графа завдань та ентропійного фільтра

Двохстадійний алгоритм обслуговування (HTN-планування) програмно імплементовано з використанням бібліотеки NetworkX [18]. Після того, як користувач робить запит, модуль декомпозиції створює об'єкт орієнтованого ациклічного графа `nx.DiGraph()`. Кожне атомарне підзавдання s_i додається до графа як вузол із метаданими (необхідна компетенція, час виконання, призначений виконавець). Залежності між завданнями додаються як спрямовані ребра `graph.add_edge(s_1, s_2)`.

Ентропійний фільтр системи реалізовано як окремий фоновий мікросервіс (Background Task), що працює в асинхронному циклі подій `asyncio`. Цей сервіс з визначеною періодичністю сканує базу даних на предмет співвідношення активних агентів `PhysicalServiceProvider` до загальної кількості відкритих вузлів графа s_i . Обчислення метрики ентропії H здійснюється на льоту. Якщо функція моніторингу фіксує перевищення допустимого порогу H_{max} , вона викликає метод `initiate_reconfiguration()`, який зупиняє класичний пошук у глибину (DFS) та переводить маршрутизатор у гібридний режим обслуговування.

Для моніторингу стабільності мережі розроблено мікросервіс ентропійного фільтра. Він інтегрований з графовою структурою NetworkX і працює в асинхронному режимі. Його завдання – розраховувати поточний рівень хаосу H та ініціювати реконфігурацію, якщо значення перевищує H_{max} . Програмний код модуля наведено в лістингу 3.2.

Лістинг 3.2 – Реалізація ентропійного фільтра на базі NetworkX та `asyncio`

```
import networkx as nx
import asyncio

class EntropyFilter:
    def __init__(self, threshold_hmax: float = 0.75):
```

Продовження лістингу 3.2

```

        self.hmax = threshold_hmax
        self.service_graph = nx.DiGraph()

    def calculate_entropy(self) -> float:
        total_nodes = len(self.service_graph.nodes)
        if total_nodes == 0:
            return 0.0

        offline_agents = sum(
            1 for node, attr in
self.service_graph.nodes(data=True)
            if attr.get('type') == 'physical' and not
attr.get('is_online')
        )

        # Ентропія розраховується як відношення недоступних
вузлів до загальної кількості
        current_entropy = offline_agents / total_nodes
        return current_entropy

    async def monitor_network_loop(self, router:
'TeamFinderRouter'):
        while True:
            entropy = self.calculate_entropy()
            if entropy > self.hmax:
                print(f"[ALERT] Entropy {entropy:.2f} >
{self.hmax}. Soft degradation triggered.")
                # Ініціація масового переведення вузлів у DIY
Mode

        router.set_strategy(CognitiveServiceProvider("s-bert-v2"))
        await asyncio.sleep(5)  # Інтервал поллінгу

```

Код з лістингу 3.2 демонструє використання бібліотеки NetworkX для зберігання метаданих вузлів. Функція `calculate_entropy` сканує граф і обчислює відсоток агентів зі статусом `offline`. Асинхронний метод `monitor_network_loop` забезпечує безперервний фоновий моніторинг без блокування основного циклу (Event Loop) сервера FastAPI. Інтервал поллінгу у 5 секунд гарантує високу реактивність системи при раптовому виникненні екстремальної волатильності.

3.2.3 Проектування гібридної бази знань

Основою ефективного функціонування супервізорного ІІІ-агента є надійна інфраструктура управління даними. Оскільки система працює з різними типами даних (топологічними та семантичними), база знань TeamFinder реалізована за гібридним принципом:

- топологічний сегмент (Neo4j) використовується для збереження актуального стану системи як графа. Тут зберігаються профілі користувачів, історія їхніх транзакцій та зв'язки між агентами. Графова структура бази даних дозволяє виконувати складні аналітичні запити. Наприклад, запит «знайти найближчого вільного виконавця з компетенцією X у радіусі Y» за допомогою мови запитів Cypher у сотні разів швидше, ніж це роблять реляційні СУБД;

- векторний сегмент (FAISS) є когнітивним ядром режиму DIY Mode. Він зберігає масив заздалегідь підготовлених експертних інструкцій, регламентів та довідників. Під час завантаження в базу кожен текстовий документ пропускається через модель Sentence-BERT, яка генерує його векторне представлення (ембединг).

Коли система ініціює підстановку вузла-замінника, CognitiveServiceProvider формує запит до векторного сегмента. За допомогою бібліотеки FAISS виконується швидкий пошук K найближчих сусідів (K-Nearest Neighbors) на основі косинусної близькості між вектором

запиту користувача v_q та векторами бази інструкцій v_{cat} . Отриманий релевантний контекст потім форматується згідно з протоколом MCR і передається генеративній мовній моделі для формування фінальної покрокової інструкції для користувача.

Така програмна архітектура гарантує, що система не лише зберігає свою структурну цілісність під час відмов фізичних вузлів, але й забезпечує високу швидкість інформаційної компенсації без створення надмірного навантаження на обчислювальні ресурси сервера.

3.3 Реалізація модуля ін'єкції знань та косинусної фільтрації

У гібридній системі TeamFinder ключовим механізмом забезпечення структурної живучості при виникненні критичного розриву є модуль ін'єкції знань, коли предикат наявності ресурсу $R = 0$.

Цей програмний компонент відповідає за переведення транзакції зі стандартного шляху пошуку виконавця у режим підтриманої самодопомоги (DIY Mode).

Головним технічним викликом на цьому етапі є подолання так званого «семантичного розриву» – проблеми невідповідності між тим, як користувач формулює свою проблему природною мовою, і тим, як ці проблеми класифіковані у внутрішній базі знань платформи. Простий лексичний пошук, наприклад, за ключовими словами або за допомогою алгоритму BM25, не здатний вловити глибокий контекст технічного чи побутового запиту.

Для вирішення цієї проблеми програмна реалізація когнітивного модуля системи базується на методах глибокого навчання, зокрема на використанні сіамських нейромережових архітектур Sentence-BERT. Ця архітектура дозволяє трансформувати текстові дані у щільні векторні представлення (ембединги), зберігаючи їхнє семантичне значення.

3.3.1 Імплементація семантичної векторизації (Sentence-BERT)

На програмному рівні операція семантичної векторизації реалізована мовою Python з використанням відкритої бібліотеки sentence-transformers, що розроблена на базі екосистеми Hugging Face. У ядрі системи ініціалізується екземпляр попередньо натренованої багатомовної моделі. Наприклад, paraphrase-multilingual-MiniLM-L12-v2 або аналогічної моделі, що підтримує україномовний контекст, зважаючи на цільову аудиторію розроблюваного продукту.

Коли система фіксує відсутність фізичного виконавця для атомарного підзавдання s_i , текстовий опис цього підзавдання (запит користувача) передається до когнітивного агента. Модель Sentence-BERT виконує токенізацію тексту, пропускає його через багатопшарову архітектуру Transformer та застосовує операцію пулінгу (Mean Pooling). У результаті цього процесу текстовий запит перетворюється на щільний числовий вектор v_q (ембединг) фіксованої розмірності (зазвичай 384 або 768 вимірів). Цей багатовимірний вектор інкапсулює семантичний зміст запиту. Близькі за змістом фрази (наприклад, «протікає труба» та «аварія водопроводу») матимуть вектори, що розташовані близько один до одного у семантичному просторі.

3.3.2 Програмна реалізація векторного індексу за допомогою FAISS

Оскільки експертна база знань платформи (інструкції, регламенти, покрокові гайди) може містити тисячі документів, порівняння вектора-запиту з кожним документом у реальному часі є ресурсомісткою операцією, яка має лінійну складність $O(N)$. Для забезпечення мінімальної затримки (low latency) при обслуговуванні запитів, у системі TeamFinder імплементовано бібліотеку FAISS (Facebook AI Similarity Search) [19].

Усі еталонні інструкції з бази знань попередньо векторизуються тією ж моделлю Sentence-BERT під час ініціалізації системи. Їхні вектори v_{cat} нормалізуються за довжиною (L2 normalization) та завантажуються у векторний індекс FAISS. У програмному коді для цього використовується клас `faiss.IndexFlatIP` (Inner Product). Завдяки попередній L2-нормалізації векторів, обчислення внутрішнього (скалярного) добутку в FAISS стає математично еквівалентним обчисленню косинусної близькості, проте виконується на апаратному рівні значно швидше.

Ядром когнітивної системи підтримки є модуль семантичної векторизації на базі Sentence-BERT та векторна база даних FAISS. Програмна імплементація ініціалізації моделі та створення індексу для швидкого пошуку релевантних експертних інструкцій подана у лістингу 3.3. Особливістю реалізації є підтримка апаратного прискорення через CUDA для забезпечення низьких затримок (low latency).

Лістинг 3.3 – Семантична векторизація та ініціалізація індексу FAISS

```
from sentence_transformers import SentenceTransformer
import faiss
import numpy as np
import torch

class KnowledgeBaseIndexer:
    def __init__(self, model_name='paraphrase-multilingual-
MiniLM-L12-v2'):
        # Використання GPU для інференсу, якщо апаратне
        # забезпечення це дозволяє
        self.device = 'cuda' if torch.cuda.is_available() else
        'cpu'
        self.model = SentenceTransformer(model_name,
        device=self.device)

        # Розмірність вектора для обраної моделі (384 виміри)
```

Продовження лістингу 3.3

```

        self.dimension =
self.model.get_sentence_embedding_dimension()

        # Використання косинусної близькості (Inner Product)
        self.index = faiss.IndexFlatIP(self.dimension)
        self.documents_metadata = []

def add_knowledge_documents(self, documents: list[str]):
    # Генерація ембедингів
    embeddings = self.model.encode(documents,
normalize_embeddings=True)
    # Додавання до індексу FAISS
    self.index.add(np.array(embeddings, dtype=np.float32))
    self.documents_metadata.extend(documents)
    print(f"Indexed {len(documents)} DIY guides.")

def search_relevant_guide(self, user_query: str, top_k:
int = 1) -> str:
    query_vec = self.model.encode([user_query],
normalize_embeddings=True)
    scores, indices =
self.index.search(np.array(query_vec, dtype=np.float32),
top_k)

    # Ентропійне відсікання: перевірка порогу впевненості
    (Confidence Threshold)
    if scores[0][0] < 0.75:
        return "Релевантної інструкції не знайдено.
Перемикання на ескалацію."

    return self.documents_metadata[indices[0][0]]

```

У лістингу 3.3 реалізовано автоматичне визначення доступності тензорних ядер за допомогою `torch.cuda.is_available()` [20]. Це критично

важливо, оскільки дозволяє розгортати систему як на хмарних серверах, так і виконувати локальний інференс під час розробки на споживчих відеокартах середнього сегменту (наприклад, мобільних чипсетах рівня RTX 4050). Використання класу IndexFlatIP у FAISS у поєднанні з прапорцем `normalize_embeddings=True` забезпечує математично точне обчислення косинусної близькості між вектором запиту v_q та еталонними документами v_{cat} .

3.3.3 Алгоритм косинусної фільтрації та відсікання (Thresholding)

Сам процес ін'єкції знань реалізований через асинхронну функцію, яка приймає вектор-запит v_q та викликає метод `index.search(v_q, k)`. FAISS повертає K найбільш релевантних документів з бази знань та їхні оцінки (scores), які відповідають значенню косинусної близькості $\cos(\theta)$ у діапазоні від -1 до 1.

Щоб уникнути ситуації, коли система намагається надати допомогу у сфері, де вона не має релевантних інструкцій, у коді реалізовано механізм ентропійного відсікання. Задається поріг впевненості (Confidence Threshold, наприклад, 0.75). Якщо найкращий знайдений збіг має метрику косинусної близькості нижчу за цей поріг, система визнає, що не має достатньо якісного контексту для формування безпечної інструкції. У такому випадку когнітивний агент делегує запит на вищий рівень ескалації (наприклад, пропонує зв'язок з живим оператором служби підтримки). Якщо ж поріг подолано, знайдений документ маркується як валідний контекст.

3.3.4 Інтеграція контексту та генеративний вивід (MCP-протокол)

Завершальним етапом роботи модуля є перетворення знайденого «сирого» контексту на зручну для користувача форму. На цьому етапі до роботи підключається протокол Model Context Protocol (MCP).

Програмний сервер MCP отримує від FAISS релевантний еталонний документ v_{cat} і формує на його основі структурований промпт для великої мовної моделі.

Промпт містить жорсткі системні інструкції (System Message). Мовній моделі суворо забороняється генерувати власні факти (галюцинувати) і наказується спиратися виключно на наданий контекст бази знань TeamFinder. LLM аналізує цей контекст, зіставляє його з оригінальним запитом користувача і генерує персоналізовану, покрокову інструкцію. Ця інструкція повертається через API до клієнтського додатка, вбудовуючись у граф завдань як тимчасовий вузол.

Саме завдяки такій програмній зв'язці «Sentence-BERT → FAISS → MCP → Generative LLM» модуль ін'єкції знань досягає своєї мети. Він безшовно компенсує відсутність фізичного фахівця високоякісною, безпечною та алгоритмічно верифікованою експертною допомогою.

3.4 Інтерфейс користувача та тестування роботи системи в умовах симуляції відмов (волатильності)

3.4.1 Організація взаємодії з користувачем (UI/UX) та візуалізація DIY-модуля

Для забезпечення ефективної взаємодії користувача з мультиагентною системою TeamFinder в роботі було розроблено клієнтський вебінтерфейс, спроектований з урахуванням принципів адаптивності (Responsive Web Design) та мінімізації когнітивного навантаження. Головним завданням інтерфейсу є безшовне переведення користувача зі стандартного сценарію пошуку фахівця до режиму підтриманої самодопомоги (DIY Mode) у разі виникнення системної відмови.

Графічний інтерфейс користувача (GUI) логічно поділяється на три основні функціональні екрани:

- екран ініціалізації запиту. Користувач вводить опис проблеми природною мовою. Наприклад, «Необхідно налаштувати локальну мережу в офісі після відключення електроенергії». На цьому етапі NLP-модуль на базі Sentence-BERT обробляє запит у фоновому режимі та формує векторне представлення проблеми для HTN-планувальника;

- монітор статусу пошуку (Dashboard). Інтерфейс відображає процес маршрутизації запиту між агентами-виконавцями. У разі успішного знаходження вільного фахівця система генерує стандартний смарт-контракт на виконання послуги;

- термінал когнітивної компенсації (DIY-інтерфейс). Якщо ентропійний фільтр фіксує відсутність доступних вузлів-виконавців (критичний розрив), інтерфейс динамічно трансформується. Користувачеві виводиться повідомлення про тимчасову недоступність експертів із пропозицією активувати режим ін'єкції знань.

При активації DIY Mode користувач потрапляє в інтерактивний діалоговий простір (AI-асистент). Згенерована великою мовною моделлю інструкція розбивається на атомарні, легко засвоювані кроки (UI-картки). Кожен крок містить перелік необхідних інструментів, алгоритм дій та техніку безпеки. Інтерфейс підтримує контекстний діалог: користувач може уточнити будь-який незрозумілий термін або крок, не виходячи за межі поточного завдання, що забезпечується збереженням історії сесії через механізми Model Context Protocol.

3.4.2 Методика тестування та симуляція екстремальної волатильності

Для верифікації ефективності розроблених алгоритмів та перевірки здатності системи підтримувати структурну живучість було проведено серію навантажувальних тестувань. Головною метою експерименту було

порівняння класичної моделі сервісного маркетплейсу та розробленої гібридної системи TeamFinder в умовах штучно згенерованого дефіциту ресурсів – екстремальної волатильності.

Методологія тестування базувалася на принципах хаос-інженерії (Chaos Engineering).

Було розроблено спеціальний тестовий модуль «Генератор волатильності» (Volatility Injector), який імітував реальні кризові умови. Наприклад, раптові відключення агентів-виконавців від мережі, ігнорування запитів, а також різкі сплески клієнтського попиту.

Тестування проводилося за трьома сценаріями:

- сценарій А (нормальні умови). 100% фізичних агентів-виконавців доступні в мережі (базовий рівень ентропії);
- сценарій В (помірна волатильність). Ймовірність відмови вузла-виконавця становить 50%, що імітує часткові блекаути або «середній» дефіцит кадрів;
- сценарій С (екстремальна волатильність). 85% агентів-виконавців відключені від мережі.

Тестова вибірка складала 1000 згенерованих клієнтських запитів різного ступеня складності для кожного сценарію. Основними метриками оцінки слугували:

- Service Rejection Rate (SRR) – відсоток запитів, які система не змогла задовольнити жодним чином (повна відмова);
- Resolution Rate (RR) – загальний відсоток успішно закритих завдань (як фізичними виконавцями, так і за допомогою DIY-модуля);
- DIY Activation Rate – частка користувачів, чия проблема була вирішена виключно завдяки когнітивному модулю підтриманої самодопомоги.

Отримані масиви даних сформували емпіричну базу для подальшого порівняльного аналізу та математичного обґрунтування ефективності запропонованого гібридного підходу.

3.4.3 Аналіз результатів експериментального дослідження

Результати симуляції наочно продемонстрували вразливість традиційних архітектур та високу адаптивність розробленої системи, результати зведено у таблицю 3.1.

Таблиця 3.1 – Порівняльний аналіз живучості систем при різних рівнях волатильності

Сценарій тестування	Відсоток доступних агентів	SRR (Класична платформа), %	SRR (Система TeamFinder), %	Resolution Rate (TeamFinder), %	Частка вирішень через DIY Mode, %
А (Норма)	100%	2.5	2.1	97.9	1.5
В (Помірна)	50%	48.7	12.4	87.6	38.5
С (Екстремальна)	15%	83.2	26.8	73.2	61.4

Як видно з результатів тестування, за умов екстремальної волатильності (сценарій С) класична модель сервісного ринку фактично колапсує. Рівень відмов (SRR) сягає понад 83%, що означає повне руйнування сервісних ланцюгів.

Натомість, впровадження мультиагентної моделі TeamFinder з ентропійним фільтром та HTN-плануванням, дозволило утримати загальний рівень успішності (Resolution Rate) на позначці 73.2%, навіть при втраті 85% виконавців. Такий високий показник структурної живучості був досягнутий виключно завдяки модулю ін'єкції знань. У понад 60% випадків система успішно апроксимувала експертну функцію та надала користувачеві релевантний, згенерований за допомогою Sentence-BERT та LLM протокол самостійного розв'язання задачі.

Окрім кількісних метрик, тестування підтвердило високу швидкодію системи.

Середній час переходу від стану «критичного розриву» до генерації першої UI-картки DIY-інструкції склав менше 2.5 секунд, що є оптимальним показником для утримання уваги користувача.

Таким чином, результати експериментального моделювання повністю підтверджують адекватність розробленої математичної моделі та доводять практичну ефективність запропонованих алгоритмів автономної реконфігурації в умовах нестабільного середовища.

3.5 Алгоритмічне забезпечення бізнес-логіки та фінансова живучість платформи

3.5.1 Математична модель динамічного ціноутворення та розрахунку комісій

Функціонування гібридної системи в умовах екстремальної волатильності вимагає не лише топологічної, але й економічної гнучкості. Класичні платформи послуг оперують статичними прайс-листами або фіксованою маржою, що стає неефективним при раптових змінах стану мережі. У системі TeamFinder облік транзакцій та тарифікація реалізовані через спеціалізований бекенд-модуль, який автоматизує обчислення за допомогою комплексних багатокритеріальних логічних ланцюжків (logic strings).

Цей модуль у режимі реального часу аналізує поточну ентропію системи H та розраховує динамічні ціни, застосовує гнучкі групові знижки (group-based discounts) для специфічних категорій замовників (наприклад, волонтерських об'єднань, корпоративних клієнтів або соціально вразливих груп) та визначає обсяг комісійних відрахувань агентам-виконавцям.

Логіка ціноутворення розділяється на два вектори залежно від обраного алгоритмом маршруту виконання:

– при проходженні за Шляхом А (залучення фізичного виконавця), кінцева вартість послуги P_{final} розраховується з урахуванням коефіцієнта волатильності k_v , який зростає при дефіциті кадрів, стимулюючи виконавців брати замовлення;

– при ініціації Шляху В (перехід у DIY Mode), фізична послуга не надається. Натомість, система монетизує доступ до преміум-сегмента когнітивної бази знань (згенерованих покрокових інструкцій), вартість якого є на порядок нижчою за фізичний виклик.

Формула динамічного розрахунку (3.1) вартості для фізичного виконавця має такий вигляд.

$$P_{final} = (P_{base} \cdot k_v) - D_{group} + C_{platform}, \quad (3.1)$$

де P_{base} – базова ставка за послугу;

k_v – коефіцієнт волатильності, який прямо пропорційно залежить від поточного рівня ентропії підграфа послуг (набуває значень від 1.0 до 3.0);

D_{group} – розрахований обсяг групової знижки;

$C_{platform}$ – комісійний збір платформи.

Обчислення комісій для виконавців відбувається автоматично при зміні статусу вузла в графі завдань на «виконано».

Використання автоматизованих логічних ланцюжків унеможливорює помилки ручного обліку та гарантує прозорість розподілу фінансових потоків усередині мультиагентного середовища. Такий підхід дозволяє підтримувати економічну стійкість платформи в періоди пікових навантажень, гнучко адаптуючи фінансові стимули під поточний стан топології мережі. У підсумку це забезпечує компроміс між мотивацією активних виконавців та збереженням загальної лояльності користувачів.

3.5.2 Диверсифікація платіжних шлюзів в умовах інфраструктурних відмов

Екстремальна волатильність середовища, викликана енергетичною нестабільністю (віяловими або аварійними відключеннями електроенергії), неминуче супроводжується ризиками тимчасової недоступності традиційних банківських шлюзів та еквайрингу. Якщо сервісна платформа спирається виключно на один фінансовий протокол, втрата зв'язку з банківським API призводить до неможливості завершення транзакції, навіть якщо послуга була успішно надана фізично або через когнітивний модуль.

Для забезпечення абсолютної фінансової живучості, архітектура TeamFinder передбачає інтеграцію альтернативних децентралізованих платіжних рішень. Зокрема, імплементовано модуль крипто-процесингу з фокусом на швидкі блокчейн-мережі (наприклад, інтеграція смарт-контрактів в екосистемі TON).

Використання таких рішень дозволяє проводити транзакції (оплату послуг та виплату комісій виконавцям) у режимі 24/7 з мінімальними комісіями та нульовим часом простою.

Навіть при повному блекауті централізованих фінансових систем, маршрутизація платежів через децентралізовані мережі гарантує, що виконавці миттєво отримають свою винагороду на цифрові гаманці чи відповідні криптовалютні картки.

Це створює потужний рівень довіри до платформи та утримує агентів-виконавців у системі.

3.5.3 Економічна ефективність впровадження когнітивної компенсації

Окрім забезпечення безперервності обслуговування, архітектурний підхід системи TeamFinder генерує значний позитивний економічний ефект як для кінцевого споживача, так і для самої платформи.

Впровадження режиму DIY Mode дозволяє платформному бізнесу зберігати клієнта та утримувати рівень конверсії навіть при відсутності пропозиції на ринку праці.

Для аналізу ефективності проведено порівняння середніх витрат на обробку одного сервісного запиту (умовної одиниці послуги) при традиційному сценарії та при активації когнітивного модуля (таблиця 3.2).

Таблиця 3.2 – Порівняльна економічна ефективність обробки запиту

Стаття витрат	Шлях А (Фізичний виконавець)	Шлях В (DIY Mode + ІІІ)
Залучення ресурсу	100% (ринкова ставка фахівця)	0% (відсутність людини)
Обчислювальні витрати	Низькі (тільки маршрутизація)	Високі (API LLM, інференс S-BERT)
Логістика / Час очікування	Від 1 до 4 годин	Менше 3 секунд
Маржинальність платформи	15% (стандартна комісія)	85% (висока маржа цифрового продукту)
Статус транзакції при відмові	Втрата клієнта (0 прибутку)	Мікро-транзакція за експертний гайд

Згідно з даними таблиці, незважаючи на підвищені витрати на обчислювальні потужності для підтримки генеративного виводу (інференс великих мовних моделей), собівартість інформаційної компенсації є мізерною порівняно з втратою клієнта при повній відмові системи. Трансформація критичного розриву в генерацію кастомного покрокового інструктажу дозволяє системі TeamFinder монетизувати ентропію, підтримуючи фінансову стабільність платформи навіть при втраті 85% активних фізичних агентів.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-прикладну задачу розробки гібридної мультиагентної системи TeamFinder, яка забезпечує безперебійне надання послуг в умовах екстремальної волатильності ринку та інфраструктурної нестабільності шляхом впровадження механізмів когнітивної апроксимації експертних функцій.

Основні наукові та практичні результати роботи полягають у наступному:

- доведено, що класичні транзакційні платформи послуг є нежиттєздатними в умовах кризової деградації пулу фізичних виконавців (блекаути, міграційні процеси, каскадні відмови зв'язку). Обґрунтовано необхідність переходу до гібридних архітектур, здатних динамічно компенсувати відсутність людини-агента за допомогою ШІ-інструментів;

- розроблено математичну модель мультиагентного середовища на базі динамічних графів та імплементовано алгоритм «ентропійного фільтра». Введення метрики поточного хаосу системи (H) дозволило автоматизувати процес прийняття рішень щодо ініціації автономної реконфігурації (фазового переходу) без очікування критичної відмови;

- спроектовано та програмно реалізовано мікросервісну архітектуру з використанням апарату ієрархічних мереж завдань (HTN-планування). Застосування поведінкового патерну «Стратегія» дозволило реалізувати механізм безшовної підміни недоступного фізичного виконавця на когнітивного ШІ-агента без переривання загального графа бізнес-процесу;

- створено когнітивний контур підтриманої самодопомоги (DIY Mode) на базі трансформерної архітектури Sentence-BERT та векторної БД FAISS. Організація семантичного пошуку з використанням косинусної близькості векторів дозволила системі миттєво генерувати високоточні, контекстно-залежні інструкції для користувачів, повністю нівелюючи

проблему «галюцинацій» великих мовних моделей завдяки контрольованому доступу до експертної бази знань;

– розроблено економічну модель функціонування системи, яка включає алгоритми динамічного ціноутворення, розрахунку комісій та інтеграцію децентралізованих фінансових протоколів. Доведено, що впровадження когнітивної компенсації підвищує маржинальність платформи та зберігає клієнтську лояльність навіть при втраті до 85% активних фізичних агентів.

Одержані результати становлять комплексну науково-практичну базу для створення нового покоління відмовостійких сервісних платформ. Впровадження розроблених алгоритмів дозволяє суттєво знизити відсоток відмов (Service Rejection Rate), зберегти цілісність бізнес-процесів та забезпечити безперервну соціально-економічну підтримку користувачів за умов жорстких кризових обмежень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Гриньова О., Пономарьов В. Інтелектуальна система TEAMFINDER для гібридних сервісів. *Сучасні інформаційні системи та технології в цифровому суспільстві* : матеріали Міжнар. науково-практ. конф., 16–17 квіт. 2026 р. Харків, 2026. С. 97.
2. Richards M., Ford N. Fundamentals of software architecture: an engineering approach. O'Reilly Media, 2020. 432 p.
3. Пономарьов В.О., Гриньова О.Є. Інтелектуальна система TeamFinder для гібридних сервісів. *Радіoeлектроніка та молодь у XXI столітті* : матеріали 30-го Міжнар. молодіж. форуму, 22–24 квітня 2026 р. Харків : ХНУРЕ, 2026. Т. 6. С. 96–98.
4. Norvig P., Russell S. Artificial intelligence: a modern approach. Pearson, 2020. 1136 p.
5. Matchmakers: the new economics of multisided platforms / ed. by S. R. Author. Harvard Business Review Press, 2016. 260 p.
6. LangChain overview – Docs by LangChain. *LangChain overview – Docs by LangChain*. URL: <https://python.langchain.com/> (date of access: 10.06.2026).
7. AutoGen: enabling next-gen LLM applications via multi-agent conversation. *arXiv.org*. URL: <https://doi.org/10.48550/arXiv.2308.08155> (date of access: 11.06.2026).
8. CrewAI Documentation – CrewAI. *CrewAI Documentation – CrewAI*. URL: <https://docs.crewai.com/> (date of access: 11.06.2026).
9. Design patterns: elements of reusable object-oriented software / ed. by G. Erich. Reading, Mass : Addison-Wesley, 1995. 395 p.
10. Barabási A.-L. Network science. Cambridge : Cambridge University Press, 2016. 456 p.
11. Глибовець М. М., Олецкий О. В. Штучний інтелект. Київ : Києво-Могила. акад., 2002. 366 с.

12. Ghallab M., Nau D., Traverso P. Automated planning and acting. Cambridge University Press, 2016. 368 p.
13. Вступ до алгоритмів / Т. Кормен та ін. пер. з англ. 3-тє вид. Київ : К. І. С., 2019. 1288 с.
14. Reimers N., Gurevych I. Sentence-BERT: sentence embeddings using siamese bert-networks. *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, Hong Kong, China. Stroudsburg, PA, USA, 2019. URL: <https://doi.org/10.18653/v1/d19-1410> (date of access: 13.06.2026).
15. What is the model context protocol (MCP)? – model context protocol. *What is the Model Context Protocol (MCP)? – Model Context Protocol*. URL: <https://modelcontextprotocol.io/> (date of access: 14.06.2026).
16. Python 3.14 documentation. *Python documentation*. URL: <https://docs.python.org/3/> (date of access: 15.06.2026).
17. FastAPI – FastAPI. *FastAPI* – *FastAPI*. URL: <https://fastapi.tiangolo.com/> (date of access: 16.06.2026).
18. Hagberg A. A., Schult D. A., Swart P. J. Exploring network structure, dynamics, and function using networkx. *Python in science conference*, Pasadena, California. 2008. P. 11–15. URL: <https://doi.org/10.25080/tcwg9851> (date of access: 17.06.2026).
19. Johnson J., Douze M., Jegou H. Billion-scale similarity search with GPUs. *IEEE transactions on big data*. 2019. P. 1. URL: <https://doi.org/10.1109/tbdata.2019.2921572> (date of access: 18.06.2026).
20. PyTorch Contributors. PyTorch documentation. *PyTorch documentation*. URL: <https://docs.pytorch.org/docs/2.12/index.html> (date of access: 18.06.2026).