МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н. Каразіна

Факультет: **ННІ Каразінський банківський інститут**

Кафедра: **Інформаційних технологій та математичного моделювання**

Спеціальність: **122 Комп'ютерні науки**

Освітня програма: **Комп'ютерні науки**

Група: АК-21М **денна форма навчання**

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

ДОСЛІДЖЕННЯ ТА ОЦІНКА АРХІТЕКТУРИ AI AGENT DESIGN PATTERNS У MICROSOFT AZURE

ЗА НАКАЗОМ № 4601-5 3262 ВІД 15 вересня 2025 РОКУ

Здобувача вищої освіти **Соколова Артема Валерійовича**

Робота допущена до захисту в ЕК
протокол кафедри ІТММ № 2 від 02.12.2025р.

В.о. завідувача кафедри ІТММ
PhD
_____ Д.М. Ковальчук

Науковий керівник
наук.ст., звання
_____ ПІБ

м. Харків 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет навчально-науковий інститут "Каразінський банківський інститут"
Кафедра інформаційних технологій та математичного моделювання
Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ **Н. І. Стяглик**
Підпис ініціали, прізвище

"15" вересня 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЄКТ)

Соколову Артему Валерійовичу
(прізвище, ім'я, по батькові студента)

1. Тема роботи: Дослідження та оцінка архітектури AI Agent Design Patterns у Microsoft Azure.

керівник роботи Рогов Андрій Володимирович, к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від "15" вересня 2025року № 4601-5 3262

2. Строк подання студентом роботи 24 листопада 2025 року

3. Перелік питань, які потрібно розробити:

У розділі 1: Проаналізувати сучасні підходи до RAG та багатоагентних систем підтримки клієнтів в Azure.

У розділі 2: Розробити архітектуру й прототип мультиагентного сервісу підтримки на базі Azure AI Foundry та Azure AI Search.

У розділі 3: Оцінити та порівняти патерни оркестрації агентів за якістю, безпекою й придатністю для сценарію Taurbull.

4.План роботи

№ з/п	Назви етапів роботи
1	Вибір здобувачем теми кваліфікаційної магістерської роботи
2	Затвердження плану і завдання кваліфікаційної магістерської роботи
3	Здача кваліфікаційної магістерської роботи керівнику
4	Підпис кваліфікаційної магістерської роботи керівника
5	Підпис кваліфікаційної магістерської роботи у нормоконтролера
6	Допуск завідувачем кафедри до захисту кваліфікаційної магістерської роботи
7	Захист кваліфікаційної магістерської роботи

Дата видачі завдання 15 вересня 2025 року

Студент _____ Соколов А. В.
підпис ініціали, прізвище

Керівник роботи _____ Рогов А.В.
підпис ініціали, прізвище

РЕФЕРАТ
НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ

«ДОСЛІДЖЕННЯ ТА ОЦІНКА АРХІТЕКТУРИ AI AGENT DESIGN PATTERNS У MICROSOFT AZURE»

Соколова Артема Валерійовича

Кваліфікаційна магістерська робота містить: 94 сторінки, 11 таблиць, 9 рисунків, список літератури із 35 найменувань.

Об’єктом дослідження є агентні архітектури в екосистемі Microsoft Azure.

Предметом дослідження виступають оркестраційні патерни AI-агентів та їх експлуатаційні метрики в задачах з використанням Retrieval-Augmented Generation та інструментальних дій.

Мета роботи полягає у дослідженні та емпіричній оцінці п’яти архітектурних патернів агентних систем у Microsoft Azure на задачах доступу до внутрішнього корпусу знань та виконання простих дій у прикладному бізнес-сценарії.

Для досягнення цієї мети визначено такі завдання:

1. Проаналізувати сучасні підходи до побудови RAG-рішень і багатоагентних систем підтримки в Azure, а також рекомендації Microsoft щодо оркестрації, безпеки та спостережності.

2. Спроєктувати архітектуру й прототип мультиагентного сервісу підтримки на базі Azure AI Foundry, Azure OpenAI та Azure AI Search для сценарію компанії Taurbull.

3. Реалізувати та порівняти конфігурації single-agent, послідовного, паралельного та handoff-патернів із використанням єдиного RAG-шару, інструментів та корпусу знань.

4. Налаштувати процеси пакетного оцінювання в Azure AI Foundry з використанням стандартних метрик IntentResolution, TaskAdherence, Groundedness, TaskCompletion, показників безпеки та вартості.

5. Виконати порівняльний аналіз результатів та сформулювати рекомендації щодо вибору оркестраційного патерна для корпоративних сценаріїв підтримки клієнтів.

Актуальність теми магістерської роботи зумовлена появою керованого сервісу Azure AI Foundry Agent Service та широким впровадженням LLM-агентів для роботи з корпоративними знаннями. Організації потребують методичних рекомендацій щодо вибору архітектурних патернів, які забезпечують баланс між якістю відповідей, часом, вартістю та вимогами безпеки.

За результатами дослідження сформовано методику порівняння оркестраційних патернів AI-агентів в Azure AI Foundry, побудовано єдиний RAG-шар для всіх конфігурацій та проведено експерименти, що показали відмінності між single-agent, послідовним, паралельним і handoff-підходами за якістю відповідей, часом виконання та вартістю.

Практичне значення роботи полягає у підготовці рекомендацій щодо вибору архітектурного патерна для сервісів підтримки клієнтів, у розробці еталонних налаштувань Azure AI Foundry, Azure AI Search та Azure OpenAI, а також у наданні прикладів реалізації, які можуть бути безпосередньо використані або адаптовані для побудови корпоративних мультиагентних систем.

Одержані результати можуть бути використані в освітніх програмах під час вивчення хмарних технологій та штучного інтелекту, у комерційних проєктах для прискорення розробки знаннєвих асистентів і сервісів підтримки, а також у подальших наукових дослідженнях, пов’язаних із оцінюванням і вдосконаленням агентних архітектур.

КЛЮЧОВІ СЛОВА: AI AGENT DESIGN PATTERNS, MICROSOFT AZURE, AZURE AI FOUNDRY, AZURE AI SEARCH, RETRIEVAL-AUGMENTED GENERATION, МУЛЬТИАГЕНТНІ СИСТЕМИ, ОРКЕСТРАЦІЙНІ ПАТЕРНИ, ОЦІНЮВАННЯ ЯКОСТІ, TAURBULL.

ABSTRACT

FOR THE MASTER'S QUALIFICATION THESIS

"RESEARCH AND EVALUATION OF THE ARCHITECTURE OF AI AGENT DESIGN PATTERNS IN MICROSOFT AZURE"

Artem Sokolov

The master's qualification thesis comprises 94 pages, 11 tables, 9 figures, and a list of references consisting of 35 sources.

The object of the research is agent architectures in the Microsoft Azure ecosystem.

The subject of the research is orchestration patterns of AI agents and their operational metrics in tasks involving Retrieval-Augmented Generation and tool-based actions.

The aim of the thesis is to investigate and empirically evaluate five architectural patterns of agent systems in Microsoft Azure on tasks of accessing an internal knowledge corpus and performing simple actions in an applied business scenario.

To achieve this aim, the following tasks were defined:

1. To analyse modern approaches to building RAG-based solutions and multi-agent support systems in Azure, as well as Microsoft's recommendations regarding orchestration, security, and observability.

2. To design the architecture and prototype of a multi-agent support service based on Azure AI Foundry, Azure OpenAI, and Azure AI Search for the Taurbull company scenario.

3. To implement and compare configurations of single-agent, sequential, parallel, and handoff patterns using a unified RAG layer, tools, and knowledge corpus.

4. To configure batch evaluation processes in Azure AI Foundry using standard metrics such as IntentResolution, TaskAdherence, Groundedness, TaskCompletion, as well as safety and cost indicators.

5. To perform a comparative analysis of the results and formulate recommendations on the choice of orchestration pattern for corporate customer support scenarios.

The relevance of the topic of the master's thesis is due to the emergence of the managed Azure AI Foundry Agent Service and the widespread adoption of LLM-based agents for working with corporate knowledge. Organisations require methodological guidance on selecting architectural patterns that ensure a balance between answer quality, latency, cost, and security requirements.

As a result of the research, a methodology for comparing orchestration patterns of AI agents in Azure AI Foundry was developed, a unified RAG layer for all configurations was built, and experiments were conducted that demonstrated the differences between single-agent, sequential, parallel, and handoff approaches in terms of answer quality, execution time, and cost.

The practical significance of the thesis lies in the preparation of recommendations on choosing an architectural pattern for customer support services, in the development of reference configurations for Azure AI Foundry, Azure AI Search, and Azure OpenAI, as well as in providing implementation examples that can be directly used or adapted for building corporate multi-agent systems.

The obtained results can be used in educational programmes when studying cloud technologies and artificial intelligence, in commercial projects to accelerate the development of knowledge assistants and support services, as well as in further scientific research related to the evaluation and improvement of agent architectures.

KEYWORDS: AI AGENT DESIGN PATTERNS, MICROSOFT AZURE, AZURE AI FOUNDRY, AZURE AI SEARCH, RETRIEVAL-AUGMENTED GENERATION, MULTI-AGENT SYSTEMS, ORCHESTRATION PATTERNS, QUALITY EVALUATION, TAURBULL.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК, СИМВОЛІВ І ТЕРМІНІВ

AI – Artificial Intelligence, штучний інтелект.

LLM – Large Language Model, велика мовна модель для обробки природної мови.

RAG – Retrieval-Augmented Generation, підхід, у якому модель поєднує генерацію тексту з пошуком у зовнішньому корпусі знань.

Azure AI Foundry – керований сервіс Microsoft для розробки, налаштування, оцінювання та розгортання AI-рішень і агентів у хмарі Azure.

Azure AI Foundry Agent Service – компонент Azure AI Foundry для створення, оркестрації та керування AI-агентами.

Azure AI Search – хмарний сервіс пошуку й індексації даних, який підтримує гібридний пошук (BM25 + векторний) для сценаріїв RAG.

API – Application Programming Interface, програмний інтерфейс, через який взаємодіють компоненти системи.

REST – Representational State Transfer, архітектурний стиль побудови веб-API поверх HTTP.

HTTP – HyperText Transfer Protocol, протокол передавання даних у веб-мережах.

JSON – JavaScript Object Notation, текстовий формат обміну структурованими даними.

YAML – YAML Ain’t Markup Language, текстовий формат для опису конфігурацій та workflow-сценаріїв.

SDK – Software Development Kit, набір бібліотек та інструментів для розробки застосунків під певну платформу.

BM25 – Okapi BM25, функція ранжування документів у класичному пошуку за релевантністю текстовому запиту.

RRF – Reciprocal Rank Fusion, метод об’єднання результатів кількох пошукових стратегій (наприклад, BM25 і векторного пошуку).

ВСТУП

Розвиток мовних моделей великого масштабу спричинив зсув від лінійних ланцюжків підказок до багатокомпонентних агентних систем, у яких окремі агенти взаємодіють між собою, корпоративними знаннями та зовнішніми сервісами. Для хмарної платформи Microsoft Azure сформовано систематизовані підходи до проєктування таких систем у вигляді оркестраційних патернів, що охоплюють послідовні й паралельні конвеєри, динамічні передачі між агентами, групову взаємодію та ієрархії типу planner–executor [1].

Практична цінність агентних архітектур пов’язана зі здатністю виконувати складні багатокрокові завдання: від пошуку та узагальнення відомостей у внутрішніх документах до ініціювання дій у зовнішніх інформаційних системах. Актуальність теми підсилює поява керованого сервісу Azure AI Foundry Agent Service у промисловому статусі, який надає інфраструктуру для побудови, зв’язування та спостережності за агентами, а також інтегровані інструменти для оцінювання якості рішень [2].

Якість відповідей агентів істотно залежить від організації доступу до знань. У межах Azure рекомендовано застосовувати гібридний пошук у службі Azure AI Search, який поєднує повнотекстовий BM25 і векторний пошук із подальшим об’єднанням результатів методом Reciprocal Rank Fusion. Такий підхід підвищує стабільність і релевантність Retrieval-Augmented Generation порівняно з використанням лише одного з методів [3].

Метою роботи є дослідження та емпірична оцінка п’яти архітектурних патернів агентних систем у Microsoft Azure на репрезентативних задачах з доступом до внутрішнього корпусу знань і виконанням простих дій. Порівняння здійснюється за критеріями якості відповіді, заземленості на джерела (groundedness), часових характеристик та вартості; для об’єктивності передбачено пакетні прогони та застосування стандартних інструментів оцінювання у середовищі Azure [4].

Об’єкт дослідження — агентні архітектури в екосистемі Microsoft Azure. Предмет дослідження — оркестраційні патерни та їх експлуатаційні метрики в задачах з використанням Retrieval-Augmented Generation та інструментальних дій.

Завдання дослідження: виконати огляд літератури й сучасних рекомендацій щодо агентних патернів; сформувати набір експериментальних сценаріїв і даних; реалізувати п’ять конфігурацій агентів; налаштувати процеси оцінювання з фіксацією метрик якості, groundedness, латентності та вартості; провести порівняльний аналіз і сформулювати рекомендації щодо вибору патерна залежно від класу задач та експлуатаційних обмежень.

Наукова новизна полягає у структурованому порівнянні п’яти патернів на єдиному корпусі задач і знань із застосуванням уніфікованих інструментів платформи Azure; практична значущість — у наданні еталонних конфігурацій, що можуть бути повторно використані для побудови прикладних агентних рішень в організаціях.

Структура кваліфікаційної роботи відповідає вимогам кафедри: вступ; три розділи основної частини; висновки; перелік посилань; додатки. Обсяг — не менше встановленого кафедрою мінімуму; посилання в тексті розміщуються наприкінці речення або абзацу у квадратних дужках; бібліографічні описи — за ДСТУ 8302:2015 [5].

РОЗДІЛ 1
ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ЗАСАДИ АГЕНТНИХ АРХІТЕКТУР У MICROSOFT AZURE

1.1. Поняття агентної системи та базові компоненти

Агентна система у контексті великих мовних моделей — це сукупність взаємодіючих компонентів, кожен з яких має роль, доступ до інструментів і механізми пам’яті. Типовий агент включає політику поведінки (системні інструкції), модуль планування кроків, інтерфейс виклику інструментів (функцій, API, конекторів), короткочасний контекст для поточної задачі та довготривалу пам’ять для історії або знань. Зовнішні інструменти забезпечують виконання дій поза модельним середовищем, а підсистема доступу до знань надає релевантні фрагменти для підкріплення відповіді. У системах на платформі Azure ці елементи поєднуються у середовищі Azure AI Foundry Agent Service, яке надає механізми створення і зв’язування агентів, спостережність та засоби оцінювання [2].

1.2. Патерни оркестрації агентів: огляд

Оркестрація визначає, як агенти взаємодіють між собою, інструментами та доступом до знань. У рекомендаціях Azure Architecture Center виділяють кілька фундаментальних патернів з різними топологіями взаємодії та зонами застосування. Послідовна оркестрація реалізує лінійний конвеєр, де вихід одного етапу стає входом наступного. Паралельна оркестрація дозволяє одночасне опрацювання запиту кількома агентами з наступною агрегацією результатів. Патерн handoff маршрутизує задачі між агентами залежно від контексту й компетенцій. Ієрархічні схеми типу planner–executor вводять керівника, що формує план і делегує підзадачі виконавцям; для завдань, де

потрібні дискусія або контроль якості, доречні групові взаємодії з правилами узгодження відповідей [1].

У задачах Retrieval-Augmented Generation доцільно поєднувати оркестрацію з гібридним пошуком у Azure AI Search: комбінувати векторний та текстовий пошук, а потім об’єднувати результати методом RRF. Це підвищує ймовірність виявлення релевантних фрагментів і покращує основу для подальших кроків планування чи голосування між агентами [3].

1.3. Retrieval-Augmented Generation у Microsoft Azure: принципи, індексація та пошукові стратегії

Retrieval-Augmented Generation (RAG) поєднує генеративні можливості LLM з цільовим пошуком фрагментів у корпоративному корпусі, що зменшує ризики вигадування фактів і наближує відповідь до наявних джерел. У середовищі Azure типовий конвеєр RAG складається з підготовки корпусу (очищення, нормалізація форматів, сегментація), побудови індексу в Azure AI Search, вибору стратегії пошуку та пост-обробки результатів перед передачею їх у підказку агента.

Критичним етапом є сегментація документів: надто дрібні фрагменти збільшують кількість запитів і можуть руйнувати контекст, надто великі — погіршують точність відбору. Практично доцільно застосовувати адаптивне чанкування за структурними маркерами (заголовки, підзаголовки, списки) з обмеженням за кількістю токенів і невеликим перекриттям між сусідніми фрагментами. Це підвищує шанси потрапляння релевантних частин у контекст відповіді та полегшує трасування джерел у результатах.

Для пошуку в Azure AI Search ефективною є гібридна стратегія: комбінування векторного подання фрагментів із класичним BM25 та злиття результатів методом Reciprocal Rank Fusion (RRF). Гібрид дає кращу стабільність на різнорідних корпусах (від політик і SOP до листування та звітів), тоді як чисто векторний або чисто терміновий підхід помітно чутливіші

до формулювань запитів. Додатково доцільно застосовувати фільтри за метаданими (дата, підрозділ, тип документа) і пост-ранжування для виключення дублікатів та низькоякісних фрагментів [3].

На етапі генерації агент має отримувати не лише текст фрагментів, а й структуровані посилання на джерела (ідентифікатори документів і позицій). Це спрощує реалізацію вимог до прозорості: у відповідях можна відтворювати «якірні» цитати та давати посилання на початкові матеріали. У складніших сценаріях варто додати «перевіряча фактів» як окремого агента або крок конвеєра, що повторно зіставляє ключові твердження з корпусом.

Окремим завданням є скорочення шуму в RAG-відповідях. Тут корисні три прийоми: підвищення щільності сигналу в інструкції (чітке формулювання вимоги «відповідати лише на основі фрагментів»), пріоритизація цитат у підказці (спершу джерела, потім питання), а також контроль кількості фрагментів — надмірна кількість знижує концентрацію моделі на релевантних частинах і погіршує часові та вартісні показники [3].

1.4. Відповідальне використання та безпека в агентних системах

Агентні системи повинні дотримуватися політик безпеки контенту, керування даними та аудиту дій. У виробничих умовах це означає поєднання кількох шарів контролю: системні інструкції, що визначають дозволені дії й тон відповіді; модуль фільтрації вхідних і вихідних повідомлень; обмеження на інструменти та облікові дані; логування усіх викликів і маркування подій, пов’язаних із ризиками.

У межах Azure доцільно використовувати готові можливості платформи: вбудовані фільтри вмісту та політики для запобігання небажаним категоріям, а також засоби спостережності для відстеження інцидентів і відхилень. Для мультиагентних схем додатковим механізмом виступає роль «охоронця політик» у конвеєрі, який перевіряє відповідність відповіді або плану кроків доменним вимогам. Такий «maker–checker» цикл пом’якшує ризики ескалації

неправильних дій при виклику зовнішніх API й допомагає стандартизувати вихідні формати відповіді [2].

Практична рекомендація для цієї роботи — уніфікувати набір правил у системних інструкціях усіх патернів, щоб відмінності у якості відповідей визначалися саме оркестрацією, а не неоднаковими політиками безпеки. Це також полегшить інтерпретацію метрик groundedness і «success rate».

1.5. Пов’язані фреймворки та середовище розробки в Azure

Екосистема Azure підтримує кілька підходів до реалізації агентних рішень. Керований шлях передбачає використання Azure AI Foundry Agent Service, який надає інструменти створення, зв’язування та моніторингу агентів, інтеграції з підсистемою оцінювання, журналюванням і керуванням вартістю. Цей підхід знижує навантаження на інженера, забезпечуючи спільну модель конфігурації та спостережність. Код-орієнтований шлях базується на бібліотеках на кшталт Semantic Kernel або інших фреймворках для мультиагентних взаємодій; він дає гнучкість у розбудові нестандартної логіки, але потребує більше зусиль для збирання телеметрії, узгодження політик і масштабування [2].

Для RAG-компоненти основним сервісом виступає Azure AI Search, який підтримує як класичний, так і векторний пошук, гібридні запити та різні стратегії ранжування. Незалежно від обраного способу оркестрації, саме цей сервіс варто використовувати для індексації корпоративного корпусу та як джерело фрагментів, що підживлюють відповіді агентів [3].

У контексті оцінювання якості та експлуатаційних характеристик доцільно застосовувати вбудовані інструменти платформи: пакетні прогони, журнали, метрики латентності й витрат, а також модулі оцінювання з використанням стандартних критеріїв релевантності та заземленості відповіді. Це забезпечує відтворюваність і порівнюваність результатів між різними патернами оркестрації [4].

1.6. Проміжні висновки до розділу

Агентні системи у середовищі Microsoft Azure спираються на усталені оркестраційні патерни, які задають структуру взаємодії між ролями, інструментами та доступом до знань. Ключем до стабільних і прозорих відповідей є належно спроєктований RAG-шар із гібридним пошуком, ретельною сегментацією документів і явною прив’язкою тверджень до джерел. Виробничі вимоги до безпеки та відповідального використання досягаються поєднанням політик, фільтрів і спостережності, за потреби — із додатковим контролером у конвеєрі. Практично це може бути реалізовано як у керованому середовищі Azure AI Foundry Agent Service, так і у код-підході; у будь-якому разі для оцінювання доречно використовувати штатні інструменти пакетних прогонів, журналювання й метрик платформи [1; 2; 3; 4].

1.7. Класифікація задач і придатність оркестраційних патернів

Для систематизації під час подальших експериментів доцільно класифікувати задачі за трьома вимірами: складність планування (однокрокові відповіді, багатокрокові міркування, довгі ланцюжки з умовними переходами), тип взаємодії з даними (довідковий пошук, аналітичне узагальнення, синтез зіставленої інформації) та необхідність зовнішніх дій (відсутня, прості інструменти, комбінації інструментів). Така схема дозволяє встановити відповідність між класом задачі та оркестраційним патерном.

Послідовна оркестрація виправдана там, де можна чітко виділити етапи підготовки відповіді, узгодження її з політиками і формування вихідного результату. Паралельна оркестрація є доречною для задач з невизначеною або широкою інформаційною базою, коли корисно одночасно побудувати кілька гіпотез і здійснити агрегацію. Патерн handoff доцільний при гетерогенних зверненнях, коли первинний тріаж відправляє задачу до спеціалізованого

виконавця. Ієрархічний підхід planner–executor ефективний на багатокрокових завданнях з необхідністю координації підзадач і викликів інструментів. Для довідкових і простих завдань контрольним рівнем залишається одноагентна схема з функціями, яка дає базову оцінку «вартість–час» [1].

У задачах Retrieval-Augmented Generation додатковим рішенням є увімкнення гібридного пошуку й обмеження контексту чітко вибраними фрагментами. Це підвищує стабільність як у послідовних, так і в паралельних сценаріях, оскільки помилки вибірки джерел часто сильніше впливають на якість, ніж сам вибір оркестрації [3].

1.8. Ризики, обмеження та шляхи їх мінімізації

Ключові ризики стосуються якості джерел, лавиноподібного накопичення похибок у ланцюжках міркувань, некоректних викликів інструментів і невідповідності відповідей політикам домену. На рівні даних головним джерелом відхилень є нерівномірність або зашумленість корпусу, що призводить до помилок RAG-вибірки; на рівні оркестрації — надмірна свобода агента у формулюванні плану та відсутність валідації проміжних результатів.

Мінімізація ризиків досягається поєднанням трьох прийомів. По-перше, ретельна підготовка корпусу та гібридний пошук із RRF, фільтрами за метаданими й обмеженням кількості фрагментів у підказці. По-друге, явні системні інструкції, які вимагають опори на джерела й забороняють дії поза дозволеним переліком інструментів. По-третє, введення «охоронця політик» або maker–checker кроку, що відсіює потенційно некоректні дії, і журналювання з подальшим аналізом інцидентів. У керованому середовищі Azure ці практики підтримуються штатними фільтрами контенту, спостережністю та засобами оцінювання, що полегшує контроль якості під час експериментів і порівняння патернів [2; 4].

1.9. Узагальнення теоретичних положень розділу

Розглянуті підходи формують методологічну базу для подальшого дослідження. Агентні архітектури спираються на оркестраційні патерни, кожен з яких відповідає певним класам задач і вимогам до планування, взаємодії з даними та інструментами. Стабільність і відтворюваність суттєво підвищуються за рахунок правильно спроєктованого RAG-шару на базі гібридного пошуку, а також узгоджених політик безпеки і контролю дій. Надалі ці положення буде конкретизовано у вигляді експериментального дизайну й метрик, що дозволяють порівняти п’ять патернів на спільних наборах кейсів у середовищі Microsoft Azure [1; 2; 3; 4].

1.10. Критерії та метрики оцінювання агентних патернів

Для порівняння патернів доцільно застосовувати сукупність метрик, які відображають і якість відповіді, і експлуатаційні характеристики системи. До якості належать успішність виконання завдання (task success), релевантність і повнота відповіді, а також заземленість на джерела (groundedness) у задачах з RAG. До експлуатаційних показників відносять латентність відповіді (p50, p95), вартість обчислень (токени, виклики), стабільність результатів на повторних запусках та частоту інцидентів безпеки. Практично ці метрики зручно збирати в оточенні Azure за допомогою пакетних прогонів і вбудованих оцінювачів якості та groundedness [6; 7].

Зважаючи на різну природу патернів, слід фіксувати також структурні характеристики: глибина конвеєра (для послідовних схем), рівень паралелізму та правило агрегації (для паралельних), частка передач між агентами й час маршрутизації (для handoff), а також якість планів і частка виправлень у циклі maker–checker (для ієрархій типу planner–executor). Такі додаткові показники допоможуть інтерпретувати відмінності в часі й вартості [1; 6].

1.11. Дизайн експериментів і відтворюваність

Щоб забезпечити коректність і відтворюваність порівняння, тестовий набір формують з трьох груп завдань: довідкові запити на основі корпусу, аналітичні узагальнення з кількох фрагментів та задачі із зовнішніми діями. Кожне завдання має чіткий критерій прийняття: коректність факту, наявність посилань на фрагменти корпусу, відповідність формату та правильність параметрів виклику інструменту. Для зменшення випадкових коливань результати збирають у кількох прогонах кожного патерна з фіксацією версій моделей, промптів, конфігурацій RAG і політик безпеки [6; 8].

У середовищі Azure налаштовують єдиний конвеєр пакетних прогонів, спільний для всіх п’яти патернів, з вивантаженням телеметрії (час, токени, вартість, інциденти) та артефактів оцінювання (оцінки релевантності й groundedness). Це дозволяє виконати статистичне порівняння показників і провести абляційні досліди (наприклад, гібридний пошук увімкнено/вимкнено, зміна правила агрегації в паралельних патернах) [6; 7; 9].

1.12. Приклади застосовності патернів у типових сценаріях

Послідовна оркестрація доцільна для регламентованих процесів із чіткими етапами: пошук фрагментів, узгодження з політиками, формування відповіді. Паралельна корисна там, де варто одночасно побудувати кілька гіпотез або підсумків і обрати найкращий за правилом голосування чи скорингу. Handoff виправданий при змішаних потоках запитів: первинний тріаж розподіляє завдання між спеціалістами за тематикою чи типом дії. Ієрархічні схеми з планувальником підходять для довгих багатокрокових завдань із необхідністю послідовних викликів інструментів і проміжних перевірок. Базова одноагентна конфігурація лишається корисним орієнтиром для оцінки компромісу «вартість–час» на простих запитах [1; 10].

У всіх випадках на якість істотно впливають налаштування RAG. Гібридний пошук на базі Azure AI Search із комбінуванням BM25 та векторного ранжування і злиттям результатів методом RRF знижує ризик пропуску релевантних фрагментів. Важливими залишаються розмір і перекриття чанків, фільтри за метаданими та ліміти на кількість фрагментів у підказці агента [3; 11; 12].

1.13. Проміжні висновки

Оцінювання патернів повинно спиратися на стандартизовані метрики якості та експлуатаційні показники й виконуватися на спільному конвеєрі, щоб мінімізувати систематичні відмінності. Застосовність окремих патернів визначається класом задач і вимогами до планування, тоді як налаштування RAG суттєво впливають на стабільність результатів незалежно від обраної оркестрації [1; 3; 6–8; 10–12].

1.14. Базова конфігурація: одноагентна схема з інструментами (Single-agent + tools)

Одноагентна конфігурація виконує повний цикл обробки запиту в межах одного суб’єкта, що має доступ до інструментів і шару RAG. Типовий життєвий цикл: прийом запиту, формування проміжного плану, виклик інструментів пошуку, відбір фрагментів, синтез відповіді із вбудованими посиланнями на джерела та, за потреби, виклик дії у зовнішній системі. Сильні сторони: мінімальна складність, прозорий трасинг, низькі накладні витрати. Обмеження: слабша стійкість на багатокрокових задачах і ризик переплутати етапи «пошук → міркування → дія», якщо інструкції недостатньо структуровані [1].

Щоб зменшити помилки, доцільно явно розділяти фази: спочатку вибірка фрагментів, потім узагальнення, потім дія. Практичне правило —

обмежувати кількість фрагментів і вимагати посилання на конкретні джерела, а також забороняти довільні дії поза переліком інструментів [3].

1.15. Послідовна оркестрація (Sequential)

У послідовній оркестрації запит проходить через ланцюжок спеціалізованих ролей. Приклад для бізнес-довідки: «Дослідник знань» виконує RAG-пошук, «Узагальнювач» формує відповідь, «Контролер політик» перевіряє відповідність доменним вимогам, «Оформлювач» надає фінальну відповідь або викликає дію. Цей підхід підвищує контрольованість і відтворюваність, адже кожен крок має чіткий вихід і критерії прийняття. Водночас додаються накладні витрати на передачу контексту між ролями та ризик «ефекту ламаної трубки», якщо проміжні представлення неузгоджені [1].

До практичних налаштувань належать стабільні схеми проміжних повідомлень (структуровані поля «ціль → докази → висновок → обмеження»), а також валідаційні чек-листи на кроках контролю політик. Для зниження латентності можна об’єднати суміжні ролі або передавати лише стислий набір доказів [10].

1.15.1. Стандартизовані проміжні артефакти в послідовних конвеєрах

Щоб зменшити втрати інформації між кроками, доцільно уніфікувати формат проміжних артефактів. Практично це може бути коротка структура з полями: ціль, докази, висновок, обмеження, дії. Приклад структури для кроку «Узагальнювач»:

Ціль: коротко сформулювати, чого досягає крок.
Докази: перелік фрагментів з ідентифікаторами джерел з RAG.
Висновок: стислий підсумок у 3–5 реченнях у нейтральному стилі.

Обмеження: політики/правила, яких потрібно дотриматися. Дії: подальший крок або результат для наступної ролі.

Такий шаблон полегшує контроль якості на кроці «Контролер політик» і зменшує ризик «зсуву смислу» під час передачі результатів [1].

Таблиця 1.1

Типова розкладка ролей у послідовній оркестрації

Роль	Вхід	Вихід	Критерії прийняття	Коментар
Дослідник знань	запит користувача	до 5 релевантних фрагментів з ідентифікаторами	наявність принаймні 2 узгоджених фрагментів	RAG з гібридним пошуком
Узагальнювач	фрагменти, ціль	стислий висновок, посилання на джерела	релевантність і повнота	ліміт 200–300 слів
Контролер політик	висновок	вердикт, список правок	відповідність політикам	maker–checker за чек-листом
Оформлювач / Виконавець дії	вердикт, правки	фінальна відповідь/виклик інструменту	коректність формату/параметрів	логування викликів

1.16. Паралельна оркестрація (Concurrent)

Паралельний підхід ініціює декілька гілок обробки запиту одночасно, наприклад, «Фактчекер», «Резюмер», «Контекст-розширювач» або альтернативні стратегії RAG-запиту. Результати агрегуються правилом, заздалегідь визначеним у конвеєрі: голосування, скоринг за якістю, пріоритет наявності джерел. Це корисно для відкритих запитів із широким простором відповідей або коли важлива стійкість до формулювання запиту. Недоліки —

збільшення вартості та потреба в ретельному дизайні агрегації, щоб уникнути «усереднення» корисних сигналів [1; 11].

З практичної точки зору варто обмежувати кількість паралельних гілок до двох-трьох, ретельно налаштовувати RAG-параметри в кожній гілці (різні фільтри, різний радіус пошуку) і використовувати прозорий протокол агрегації з поясненням, чому обрано саме цей варіант [12].

1.16.1. Паралельні гілки та правила агрегації

Паралельні гілки доцільно обмежувати до двох-трьох, чітко визначаючи їхні ролі. Найпоширеніші комбінації та способи злиття результатів наведені нижче.

Таблиця 1.2

Приклади паралельних гілок і агрегації

Гілка	Призначення	Ключові параметри	Правило агрегації	Сценарій
Фактчекер	перевірка тверджень	RAG: k=3; суворі фільтри	пріоритет наявності джерел	політики/регламенти
Резюмер	узагальнення	RAG: k=5; ширші фільтри	середній скоринг якості	оглядові запити
Контекст-розширювач	пошук додаткових аспектів	альтернативна формулювання запиту	голосування 2 з 3	відкриті питання

1.17. Динамічні передачі між агентами (Handoff)

Патерн handoff передбачає первинний тріаж і динамічне маршрутизування задачі до спеціаліста залежно від домену або типу дії.

Типовий приклад — «Тріаж-агент» класифікує запити на «політики», «кадрові процедури», «дії» і передає їх відповідним агентам. Перевага — зменшення когнітивного навантаження на окремі ролі та скорочення часу завдяки швидкій ідентифікації потрібної компетенції. Ризики — помилки маршрутизації та локальні оптимуми, коли класифікація хибно направляє запит і втрачається частина контексту [1; 13].

Щоб зменшити помилки маршрутизації, спеціаліст після отримання задачі виконує швидку верифікацію: якщо тема/тип дії не відповідає його компетенції, запит повертається до тріаж-агента з приміткою. Корисно вимірювати точність первинної класифікації, середній час маршрутизації та частку повторних передач; ці показники допомагають порівнювати handoff зі складнішими ієрархіями [1; 4].

1.18. Ієрархічний підхід planner–executor (Магнітні/керовані ієрархії)

Ієрархічні схеми вводять «Менеджера-планувальника», що формує план із підзадач, делегує виконання спеціалістам і замикає цикл перевірки якості. У виробничих сценаріях це зручно для довгих процесів з умовними розгалуженнями, де потрібні декілька послідовних викликів інструментів і контроль проміжних результатів. Сильні сторони: краща розкладка складних задач, можливість явної пам’яті плану, інтеграція maker–checker. Слабкі сторони: додаткова латентність на синтез і ревізію планів, ризик надмірного дроблення задач [1; 14].

Практичні рекомендації зменшення накладних витрат: обмеження глибини плану, явні критерії зупинки, збереження «ledger» із ключовими доказами та рішеннями для прозорого аудиту, а також уніфіковані формати проміжних артефактів, що спрощують порівняння з іншими патернами [15].

1.19. Порівняльна придатність патернів за класами задач

Для коректного вибору архітектури доцільно співвіднести тип задачі з патерном оркестрації. На довідкових запитах та простих діях контрольним рівнем лишається одноагентна схема з інструментами: вона мінімізує накладні витрати та спрощує трасування. Для задач, де чітко виділяються етапи підготовки, перевірки політик і оформлення результату, доцільна послідовна оркестрація; вона спирається на зрозумілі інтерфейси обміну проміжними артефактами між ролями [1]. Для відкритих або неоднозначних запитів, де корисно отримати кілька незалежних гіпотез, ефективною є паралельна оркестрація з прозорими правилами агрегації результатів [1].

Усі зазначені підходи у виробничих умовах взаємодіють зі службами платформи: оркестрування й хостинг агентів у керованому середовищі Azure AI Foundry Agent Service, доступ до знань через гібридний пошук, а також конвеєри оцінювання якості. Таке поєднання зменшує інженерні ризики і покращує відтворюваність експериментів [2; 3; 4].

Таблиця 1.3

Придатність патернів за класами задач

Клас задач	Single-agent	Sequential	Concurrent	Handoff	Planner Executor
Довідкові Q&A	висока	висока	середня	середня	середня
Аналітика узагальнення	середня	висока	висока	середня	висока
Дії через інструменти	середня	висока	середня	висока	висока
Невизначені відкриті запити	середня	середня	висока	середня	висока
Вартість (очікувана)	низька	середня	вища	середня	вища
Латентність (очікувана)	низька	середня	вища	середня	вища

1.20. Типові помилки та способи їх виявлення

Поширеними є помилки вибірки джерел у RAG (занадто великі або дрібні фрагменти, відсутність метаданих), що веде до нерелевантних цитат; уникають цього завдяки гібридним запитам і об’єднанню результатів методом Reciprocal Rank Fusion, фільтрам за метаданими та обмеженню кількості фрагментів у підказці [3; 5].

У мультиагентних схемах трапляються помилки маршрутизації (handoff) і «усереднення» корисних сигналів в паралельних ансамблях. Зниження ризиків досягається чіткими критеріями triage, перехресною перевіркою спеціаліста після передачі та прозорими правилами агрегації з наданням пояснень, чому обрано кінцевий варіант [1].

Таблиця 1.4

Типові помилки та контрольні дії

Помилка	Симптом	Причина	Контрольна дія
Нерелевантні фрагменти RAG	цитати не відповідають питанню	занадто великі/дрібні чанки	перегляд сегментації; фільтри за метаданими [3]
«Усереднення» у паралелі	відповідь поверхова	некоректне правило агрегації	правило «пріоритет наявності джерел», ваги якості [1]
Хибний handoff	не той спеціаліст	слабкий тріаж	повторний тріаж із причиною повернення [4]
Непрозорий виклик дії	немає параметрів	відсутній аудит	журнал параметрів і валідація формату [4]

Окремий клас похибок — відсутність спостережності та оцінювання. Для їх мінімізації в Azure передбачено два взаємодоповнювальні механізми: evaluation flows у порталі та локальне оцінювання через Azure AI Evaluation SDK з вбудованими «evaluators» для якості, groundedness і безпеки [4; 6; 7; 8].

1.21. Рекомендації щодо вибору патерна

Як базове правило, якщо завдання не потребує складного планування чи колективного обговорення, слід починати з одноагентної схеми: вона дає еталон витрат і часу. За наявності чітких етапів підготовки та контролю варто переходити до послідовної оркестрації. Якщо важлива стійкість до формулювань запиту або потрібні кілька незалежних точок зору, доцільно запускати паралельні гілки з голосуванням/скорингом. Для змішаних потоків звернень із різними доменами обирають handoff, а для довгих, умовно розгалужених процесів з інструментальними діями — planner–executor. Ці рекомендації узгоджуються з оглядом у Azure Architecture Center та з прикладами фреймворків Azure (Agent Service, Semantic Kernel) і дослідницьких напрацювань щодо мультиагентних взаємодій [1; 2; 6; 9; 10].

На практиці вибір патерна супроводжується налаштуванням RAG-глибини (кількість і розмір фрагментів), увімкненням гібридного пошуку та, за потреби, підключенням довготривалих обчислень або дій через керовані робочі процеси (наприклад, Durable Functions) [5; 11].

1.22. Висновки до розділу

У розділі розглянуто фундаментальні патерни оркестрації агентів у Microsoft Azure та їх застосовність до різних класів задач. Показано, що якість і відтворюваність суттєво залежать від належно спроєктованого RAG-шару на базі гібридного пошуку й RRF та від уніфікованого процесу оцінювання через evaluation flows і Azure AI Evaluation SDK. Практичні рекомендації зводяться до поетапного підходу: базова одноагентна конфігурація як орієнтир витрат і часу; далі — перехід до послідовної, паралельної, handoff або ієрархічної схем залежно від вимог задачі, інтеграцій та політик домену [1–8; 11].

РОЗДІЛ 2
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПРОЄКТУВАННЯ МУЛЬТИАГЕНТНОГО РІШЕННЯ В AZURE AI

2.1. Аналіз предметної області та постановка задачі

Розглядається типова для знаннєво-орієнтованих організацій ситуація, коли співробітники регулярно звертаються до внутрішніх документів: регламентів, інструкцій, технічних описів, шаблонів листів, політик безпеки тощо. Ці матеріали зберігаються у вигляді файлів (PDF, DOCX), записів у внутрішньому Wiki та базах знань. Обсяг корпусу є достатньо великим, щоб ручний пошук і самостійне узагальнення інформації були трудомісткими та часозатратними.

Основні запити користувачів охоплюють:

- пошук конкретних положень або вимог у багатосторінкових документах;
- узагальнення декількох джерел у короткі інструкції чи пояснення;
- підготовку чернеток відповідей клієнтам або внутрішніх службових листів з посиланням на актуальні правила;
- ініціювання простих дій у зовнішніх системах (наприклад, створення заявки, фіксація факту виконання певної процедури).

Традиційний повнотекстовий пошук по документах частково вирішує задачу знаходження релевантних фрагментів, але не забезпечує автоматизованого узагальнення, пояснень та дій. Саме тут актуальним стає використання агентної системи з Retrieval-Augmented Generation та викликом інструментів у хмарному середовищі Azure [1–3].

У спрощеному вигляді можна виділити такі ролі:

- кінцевий користувач (співробітник), який формулює запит природною мовою;
- агент(и), що інтерпретують запит, звертаються до корпусу знань і, за

потреби, до зовнішніх сервісів;

- підсистема доступу до знань (Azure AI Search з гібридним пошуком і відповідним індексом документів);
- зовнішні інформаційні системи (умовний сервіс управління заявками, e-mail-шлюз тощо), з якими агенти взаємодіють через інструменти.

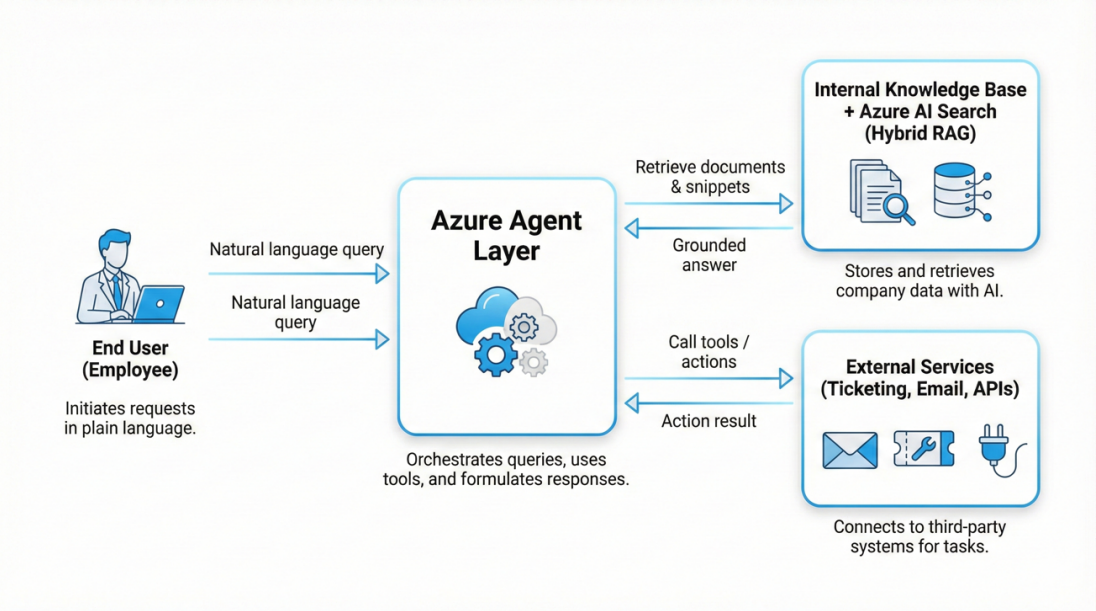

Рис 2.1. Узагальнена схема предметної області та взаємодії

Виходячи з описаної предметної області, до експериментальної агентної системи висуваються такі ключові вимоги:

- здатність відповідати на фактологічні запитання з посиланням на внутрішні документи (groundedness);
- підтримка складніших запитів, що потребують узагальнення кількох джерел і пояснення логіки відповіді;
- формування прикладних артефактів (чернетки листів, короткі інструкції, списки кроків) у стилі, наближеному до корпоративних шаблонів;
- можливість виклику простих інструментів для реалізації дій: створення або оновлення сутностей у зовнішній системі, фіксація результатів;
- ведення журналу взаємодій і технічної телеметрії для подальшого аналізу й оцінювання.

Нефункціональні вимоги включають обмеження на час відповіді (латентність), вартість запитів до моделей Azure OpenAI, стабільність якості при повторних зверненнях, а також відтворюваність результатів при пакетних прогнах, що особливо важливо для порівняння різних патернів оркестрації [1; 4].

Формулювання задачі для експериментів.

Щоб перейти від загального опису предметної області до конкретної практичної частини, необхідно зафіксувати, які саме сценарії будуть використані для порівняння п’яти патернів оркестрації. У роботі розглядається три групи сценаріїв:

- S1 – прості запити до знань.

  Одноходові питання користувача, на які можна відповісти, спираючись на один чи кілька фрагментів із корпусу документів (наприклад, «Який порядок погодження договору певного типу?»). Основний акцент — на релевантності та groundedness відповіді.
- S2 – складні аналітичні запити.

  Запитання, що потребують узагальнення інформації з багатьох документів, порівняння різних варіантів або побудови покрокового алгоритму дій (наприклад, «Порівняйте дві процедури та запропонуйте узгоджену схему дій»). Тут важливим є вміння агента планувати послідовність кроків, виконувати кілька звернень до RAG та зберігати контекст.
- S3 – запити з діями.

  Сценарії, де крім формування текстової відповіді потрібно виконати зовнішню дію через інструмент (створення чернетки листа й одночасний запис у систему заявок, реєстрація виконаної процедури тощо). У цьому випадку оцінюється не тільки якість відповіді, але й коректність виклику інструментів, прозорість параметрів і аудит виконаних дій.

Для кожної групи сценаріїв формуються репрезентативні набори тестових запитів, які згодом використовуються у пакетних прогонах у

середовищі Azure AI Foundry та при локальному оцінюванні через Azure AI Evaluation SDK [4; 6–8]. Це дозволяє зіставити роботу п’яти оркестраційних патернів (single-agent, послідовний, паралельний, handoff, planner–executor) на спільній базі, не змінюючи ані корпусу знань, ані системних інструкцій. З урахуванням описаної предметної області та вимог, у межах другого розділу вирішується така прикладна задача: спроєктувати та реалізувати експериментальну агентну систему в Azure, яка:

- використовує єдиний індекс внутрішніх документів у службі Azure AI Search з гібридним пошуком;
- підтримує виконання сценаріїв S1–S3 із можливістю журналювання запитів і результатів;
- має п’ять конфігурацій оркестрації агентів, що відрізняються лише структурою взаємодій, але спираються на однакові моделі, інструкції та джерела знань;
- інтегрована з інструментами оцінювання якості, groundedness, латентності та вартості, рекомендованими в екосистемі Azure [1; 2; 4; 6–8; 16].

Подальші підрозділи розділу присвячені деталізації архітектури такого рішення, опису структури даних та інфраструктури в Azure, а також практичній реалізації кожної з агентних конфігурацій.

2.2. Аналіз існуючих рішень та сервісів Azure для побудови LLM-агентів

Побудова LLM-агентів на платформі Microsoft Azure спирається не на один окремий сервіс, а на цілу екосистему засобів, які покривають етапи від розміщення моделей до оркестрації запитів, зберігання контексту, інтеграції із зовнішніми системами та моніторингу якості. Тому перед проєктуванням цільової системи доцільно проаналізувати наявні рішення та сервіси, які можуть бути використані як будівельні блоки майбутнього агента.

Базою для створення будь-яких агентних рішень є сервіс Azure OpenAI

Service, що надає доступ до моделей GPT-4, GPT-4o, сімейства GPT-3.5, а також спеціалізованих моделей для ембедингів та обробки зображень. Сервіс підтримує сценарії чат-взаємодії, генерації тексту, системних повідомлень, керування температурою та іншими параметрами генерації, а також можливість розгортання власних «deployment» конкретних версій моделей, що важливо для контролю якості та вартості рішень [2;9;14;22]. У поєднанні з механізмом керування доступом на рівні Azure он забезпечує ізоляцію даних підприємства і відповідність вимогам безпеки, що є критичним для корпоративних LLM-агентів.

Над ядром моделей Azure OpenAI вибудуваний інтегрований портал Azure AI Foundry (раніше Azure AI Studio), який слугує середовищем для розробки, експериментів та керування генерувальними застосунками. У межах одного проєкту розробник може підключити розгортання моделей, під’єднати джерела даних (Azure AI Search, Azure Blob Storage, SQL-сховища), створювати промпти, запускати експерименти з Prompt flow, проводити офлайн- та онлайн-оцінювання якості, а також спостерігати за поведінкою застосунку в продакшені [2;6;8;17;24]. Останні оновлення платформи додали до цього стеку спеціалізований Azure AI Agent Service, що формує окремий клас рішень – «agentic» застосунки, у яких логіка ведення діалогу, вибір інструментів і збереження стану користувача делегуються керованому сервісу [14;17;24].

Azure AI Agent Service орієнтований на побудову складних LLM-агентів із підтримкою інструментів, виклику зовнішніх API, сценаріїв RAG та багатокрокових робочих процесів. Розробник описує набір інструментів (дії), які агент може виконати: отримання документів із індексу Azure AI Search, виклик бізнес-функцій через Azure Functions, доступ до зовнішніх HTTP-сервісів тощо. Сервіс бере на себе розбір намірів користувача, планування послідовності викликів цих інструментів, збереження проміжного контексту та агрегацію проміжних відповідей у єдину відповідь користувачеві [1;14;17;24;26]. Завдяки цьому значно зменшується обсяг «клейового» коду,

який раніше потрібно було реалізовувати самостійно з використанням фреймворків на кшталт Semantic Kernel або AutoGen.

Ключовим компонентом для реалізації знаннєвих агентів є сервіс Azure AI Search, який забезпечує можливість створення індексів документів, їх попередньої обробки, векторизації та пошуку з використанням гібридних (full-text + vector) та семантичних алгоритмів [3;5;12;23]. Для RAG-сценаріїв Azure AI Search надає готові механізми chunking, збереження ембедингів, фільтрації за метаданими, а також спеціальні «agentic retrieval» шаблони, що спрощують інтеграцію з Azure OpenAI та Azure AI Agent Service [24;25]. У типових рішеннях LLM-агентів цей сервіс виступає «довготривалою пам’яттю», де зберігаються корпоративні документи, політики, методики та інша предметна інформація, до якої агент звертається при формуванні відповідей.

Окрему групу складають засоби оркестрації та інтеграції з бізнес-процесами. Для реалізації простих інтеграцій використовуються Azure Functions, які дозволяють описувати легковагові серверлес-функції, що викликаються агентом як інструменти через HTTP-інтерфейс або черги повідомлень. Для довготривалих, багатоетапних процесів застосовуються Durable Functions, орієнтовані на моделювання робочих процесів з підтримкою стану, відкатів та повторних спроб [10;11;18–20]. У разі необхідності складнішої інтеграції з існуючими інформаційними системами доцільним є використання Logic Apps або Azure Container Apps, що реалізують частини бізнес-логіки агента [15;16;21;27]. Сукупно ці сервіси дозволяють побудувати «agentic backend», котрий масштабуватиметься разом зі зростанням навантаження на корпоративну систему.

Ще одним важливим виміром є моніторинг та оцінювання якості агентних рішень. У рамках Azure AI Foundry доступні вбудовані засоби для оцінювання відповідей моделей за допомогою метрик на кшталт groundedness, relevance, coherence, toxicity, а також інструменти для A/B-експериментів над різними промптами й конфігураціями агентів [6–8;24;25].

Таблиця 2.1

Порівняльна характеристика сервісів Azure для побудови інтелектуального агента

Сервіс	Призначення	Тип	Масштабування	Роль в архітектурі	Сценарії використання
Azure OpenAI	Доступ до LLM (GPT-4o, DALL-E) для генерації та аналізу.	PaaS	Висока; квоти TPM/RPM.	«Мозок»: логіка, розуміння намірів, генерація відповідей.	Генерація тексту, сумаризація, переклад.
Azure AI Agent	Оркестрація агентів, керування станом (Threads), інтеграція інструментів.	PaaS	Автоматичне (Managed).	Оркестратор: керування діалогом та виклик функцій.	Чат-боти з пам'яттю, Code Interpreter, File Search.
Azure AI Search	Пошуковий рушій, векторний пошук, семантичне ранжування.	PaaS	Репліки та партиції.	Пам'ять (RAG): пошук у базі знань для контексту.	Векторний пошук, індексація документів.
Azure Functions	Безсерверне виконання коду (Event-driven).	FaaS	Автоматичне (Serverless).	Інструменти (Skills): виконання дій у зовнішньому світі.	API-запити, легкі обчислення.
Durable Functions	Оркестрація функцій зі збереженням стану (Stateful).	FaaS	Автоматичне.	Workflow: координація довгих процесів.	Ланцюжки викликів, очікування подій.
Cosmos DB	NoSQL база даних з низькою затримкою.	PaaS	Глобальне.	Сховище: логи, метадані, історія.	Збереження JSON, профілі користувачів.

2.3. Проєктування цільової архітектури мультиагентного рішення

Спираючись на сформульовані у підрозділі 2.1 бізнес-вимоги та аналіз сервісів Azure у підрозділі 2.2, на цьому етапі необхідно визначити цільову логічну архітектуру мультиагентного рішення. Архітектура має забезпечувати підтримку трьох груп сценаріїв (простий пошук знань, аналітичні запити та запити з виконанням дій), можливість подальшого масштабування на нові домени, а також відповідність обмеженням корпоративного середовища Taurbull щодо безпеки, керованості та спостережуваності.

Логічна архітектура системи.
У верхньому рівні архітектури розташовуються канали взаємодії з користувачем: веб-чат, інтеграція з корпоративним порталом Taurbull, потенційно — Microsoft Teams-бот. Усі ці клієнти працюють через єдиний API-шлюз (наприклад, Azure API Management), який відповідає за маршрутизацію запитів, застосування політик безпеки та лімітування навантаження. За API-шлюзом знаходиться мультиагентний оркестратор, реалізований на базі Azure AI Agent Service та Azure OpenAI. Саме в цьому шарі реалізуються окремі агенти та правила їхньої взаємодії.

У рамках цільової архітектури виділяється маршрутизуючий агент (router-агент). Він отримує вхідний запит від користувача разом із поточним контекстом сесії, класифікує його на один із типів сценаріїв і обирає подальшу стратегію обробки. Для простих запитів до знань router-агент безпосередньо передає управління агенту доступу до знань; для аналітичних запитів він ініціює роботу планувальника; для запитів, що передбачають зовнішні дії, додатково залучається агент інструментів. Такий підхід відповідає поширеному у сучасних мультиагентних LLM-системах патерну «router + фахові агенти», що дозволяє зменшити складність промптів окремих агентів та забезпечити кращу керованість поведінки системи.

Планувальник (planner-агент) відповідає за декомпозицію складного запиту на послідовність підзадач. На основі опису цільового результату, історії

діалогу та доступних інструментів він формує план у вигляді структури кроків: які під-агенти мають бути викликані, у якій послідовності та з якими параметрами. Планувальник не виконує бізнес-логіку безпосередньо, а лише будує оркестрацію, що дає змогу легко модифікувати сценарії, не змінюючи код інших компонентів. Такий поділ ролей «planner–executor» узгоджується з рекомендаціями досліджень з мультиагентних LLM-архітектур, де наголошується на користі явного планування для складних завдань.

Виконавчі агенти (executor-агенти) реалізують конкретні компетенції. У цільовому рішенні доцільно виокремити щонайменше три базові типи: агент доступу до знань, агент інструментів та спеціалізовані доменні агенти. Агент доступу до знань працює за патерном Retrieval-Augmented Generation: отримує від планувальника чи router-агента сформульований пошуковий запит, звертається до Azure AI Search, обирає релевантні фрагменти документів Taurbull і на їхній основі формує відповідь через Azure OpenAI. Агент інструментів відповідає за виклик зовнішніх API та сервісів (наприклад, системи обробки заявок, каталогу продуктів чи внутрішніх REST-сервісів), забезпечуючи перетворення запиту користувача на формалізовані параметри виклику та обробку результатів. Доменні агенти можуть спеціалізуватися на окремих бізнес-напрямах — наприклад, «агент підтримки продажів» або «агент HR-навігації» — та використовувати як RAG-підхід, так і інструменти.

Окрему роль відіграє агент-оцінювач (evaluator або critic-агент). Він аналізує проміжні та фінальні відповіді інших агентів, перевіряючи їх на відповідність фактам, політикам безпеки та тону спілкування. Для цього агент-оцінювач може повторно звертатися до Azure AI Search для перевірки groundedness, порівнювати декілька варіантів відповіді та обирати найбільш якісний, а також повертати планувальнику сигнал про необхідність переформулювання чи доуточнення. Наявність такого агента дозволяє наблизитися до рекомендованих у літературі схем «executor–critic», які демонструють кращі результати у складних завданнях порівняно з одноагентними рішеннями.

На інфраструктурному рівні мультиагентне рішення спирається на кілька ключових сервісів Azure. Довготривале зберігання корпоративних документів та знань реалізується в Azure Blob Storage; із нього дані індексуються в Azure AI Search, де будуються векторні та гібридні індекси для RAG-агента. Стан діалогів, плани виконання та результати викликів інструментів можуть зберігатися в Azure Cosmos DB або Azure Table Storage, що дає змогу відновлювати контекст сесії та проводити подальший аналіз. Для секретів, ключів доступу до моделей та зовнішніх API використовується Azure Key Vault, а спостережуваність забезпечується за допомогою Azure Monitor та Application Insights — сюди мультиагентна система надсилатиме телеметрію про виклики агентів, тривалість запитів, помилки та метрики якості.

Для підтримки довготривалих та асинхронних процесів — наприклад, коли агент ініціює створення складного звіту чи погодження договору — у архітектуру закладається використання черг та подій. Azure Service Bus або Azure Storage Queues можуть застосовуватися як транспорт для завдань, що потребують тривалого фону обробки, а Azure Event Grid — для реакції агентів на події з інших корпоративних систем (створення нової заявки, зміна статусу замовлення тощо). У такій схемі мультиагентний оркестратор виступає не лише як «чат-бот», а як універсальний шар інтелектуальної оркестрації бізнес-подій у межах підприємства.

Автентифікація та авторизація користувачів здійснюється через Microsoft Entra ID. Це дозволяє обмежувати доступ до окремих сценаріїв або інструментів залежно від ролі співробітника Taurbull, а також передавати в мультиагентну систему сигнали про департамент, географію чи інші атрибути, які можна враховувати при персоналізації відповідей. Завдяки ролям і групам в Entra ID можна гнучко виділяти техпідтримку, відділ логістики, маркетинг чи фінанси та, відповідно, відкривати їм різні підмножини агентів та інструментів, включно з доступом до чутливих операцій — наприклад, перегляду інформації про замовлення або зміни налаштувань доставки. Такий підхід також спрощує подальше масштабування: додавання нового сценарію

або агента можна прив’язати до вже наявних ролей без зміни всієї моделі доступу. Додатково архітектура передбачає інтеграцію з системами журналювання дій користувачів і агентів, що важливо для аудиту та дотримання внутрішніх політик безпеки, зокрема фіксації того, який співробітник ініціював певний запит, який агент його обробляв, які інструменти викликалися та які відповіді були надані кінцевому користувачу.

На рис. 2.2 показано місце мультиагентного оркестратора в інфраструктурі Azure: канали взаємодії з користувачем, шлюз API, набір LLM-агентів (router, planner, knowledge/RAG, tools, доменні агенти, evaluator) та шари даних і сервісів Azure (AI Search, Blob Storage, Service Bus, Key Vault, Monitor тощо), а також інтеграцію із зовнішніми системами Taurbull.

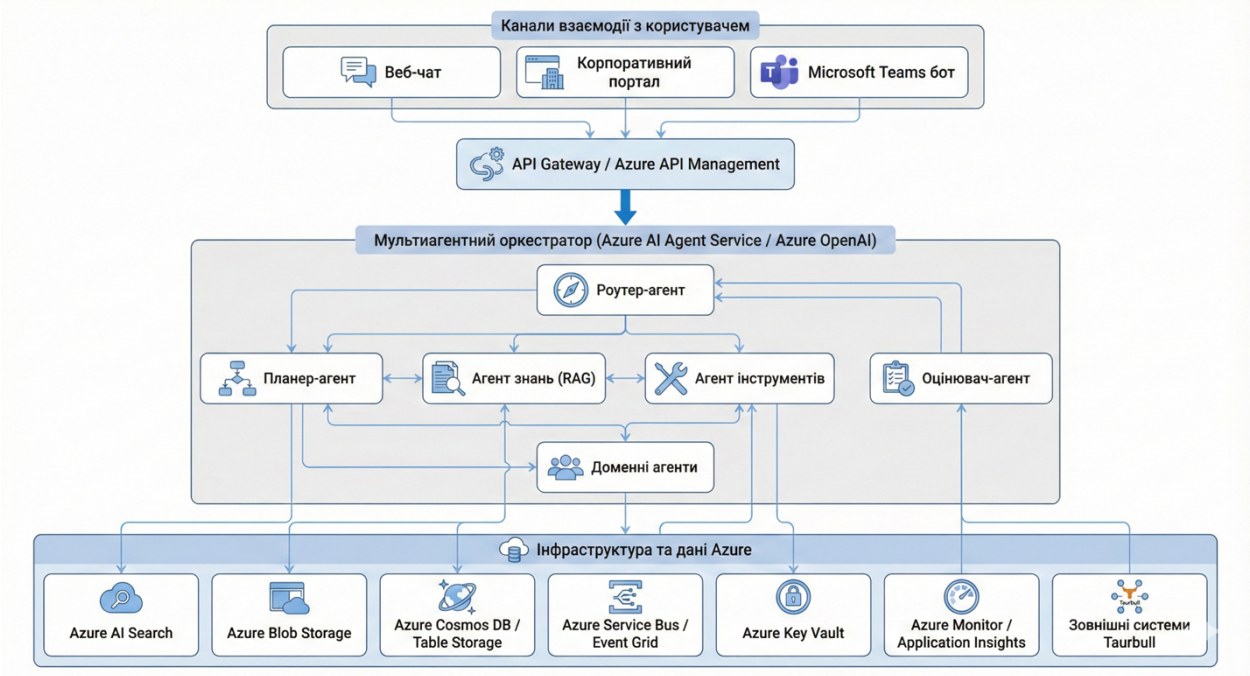

Рис. 2.2. Загальна логічна архітектура мультиагентного рішення

Запропонована цільова архітектура повинна підтримувати кілька різних патернів оркестрації, які будуть експериментально досліджені у третьому розділі. Перший патерн — одноагентний (single-agent). У цьому випадку весь інтелектуальний функціонал зосереджений в одному універсальному агенті, який має доступ до всіх інструментів та джерел знань. Такий підхід простий для реалізації, але обмежує можливість масштабування, ускладнює контроль

якості та робить промпт цього агента надто громіздким.

Другий патерн — послідовна кооперація агентів (sequential / pipeline). У ньому router-агент направляє запит спочатку до планувальника, потім планувальник послідовно викликає спеціалізовані агенти (наприклад, Knowledge agent → Tools agent → Evaluator agent), передаючи їм проміжні результати. Така схема добре підходить для сценаріїв, де логіка обробки чітко визначена й має природну послідовність кроків, як-от побудова зведеного звіту з декількох документів із подальшою генерацією листа клієнту.

Третій патерн — паралельна кооперація (concurrent multi-agent). У цьому випадку планувальник може запускати кілька виконавчих агентів паралельно: наприклад, один агент збирає інформацію з внутрішніх документів, інший — викликає API системи заявок, третій — готує варіанти формулювання відповіді різними стилями. Після завершення підзадач результати агрегуються та передаються на оцінку evaluator-агенту. Паралельна схема дозволяє скоротити час відповіді для складних запитів і краще використовувати можливості Azure для масштабування вшир.

Четвертий патерн — ієрархічний (hierarchical / manager–worker). Він наближений до підходів, описаних у роботах CAMEL та оглядових статтях про LLM-агентів, де виділяється «керівник» (manager) і група виконавців із різними компетенціями. У цільовій архітектурі роль такого керівника відіграватиме планувальник, а виконавцями виступатимуть доменні агенти й агент інструментів. Перевага цієї схеми — можливість динамічно адаптувати план залежно від проміжних результатів (наприклад, планувальник може додати новий крок, якщо оцінювач вказав на недостатню обґрунтованість відповіді).

Карта інтеграцій із сервісами Azure. Незалежно від обраного патерну оркестрації, усі агенти працюють поверх єдиної карти інтеграцій. Для коректного виконання запитів до знань RAG-агент використовує індекс Azure AI Search, який регулярно оновлюється з Azure Blob Storage через пайплайни обробки документів (наприклад, Azure Data Factory або Azure Functions).

Агент інструментів підключається до зовнішніх REST- або SOAP-API корпоративних систем через захищені endpoint’и, визначені в API Management, що дозволяє централізовано застосовувати політики авторизації, rate limiting та логування.

Дані телеметрії, проміжні плани, оцінки відповідей та інші артефакти, корисні для подальшого аналізу, надсилаються в Application Insights і зберігаються у сховищах, сумісних із Kusto (Azure Data Explorer). Це дає змогу будувати дашборди якості та навантаження, проводити A/B-експерименти між різними конфігураціями агентів і швидко виявляти «вузькі місця» у роботі системи.

Діаграма послідовностей (Рис. 2.3) ілюструє типовий сценарій: користувач надсилає запит до router-агента, далі planner-агент формує план і розподіляє підзадачі між knowledge (RAG)-агентом та tools-агентом, які звертаються до джерел та зовнішніх API. Отримані результати оцінюються evaluator-агентом і повертаються користувачу як узгоджена відповідь.

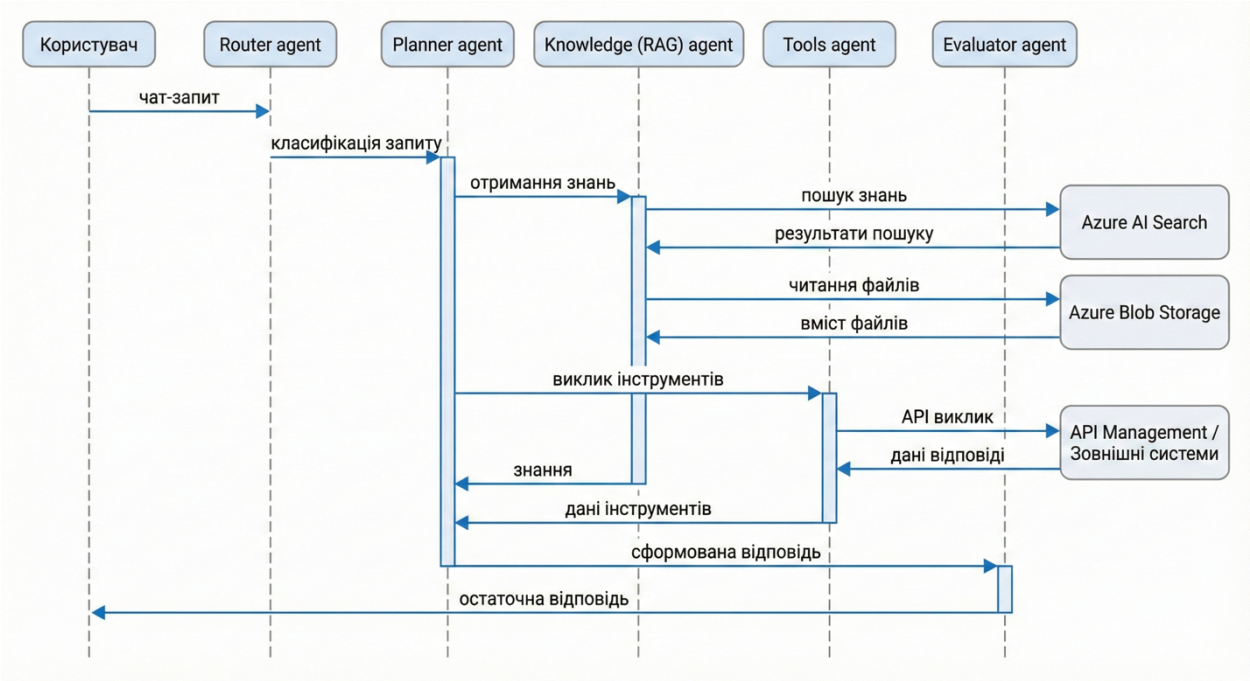

Рис 2.3. Узагальнена схема взаємодії агентів під час обробки запиту

2.4. Організація даних, інструментів (tools) та середовища експериментів

Основним джерелом доменної інформації в проєкті є публічний веб-сайт компанії Taurbull. Для побудови бази знань були відібрані три ключові сторінки: умови доставки, політика повернення товару та загальні запитання (FAQ). Саме ці документи містять більшість відповідей на типові запити клієнтів щодо логістики, якості продукції та прав споживача.

Вміст сторінок було завантажено в сховище Azure (Azure Blob Storage) у вигляді сирих HTML-/Markdown-файлів, а далі проіндексовано за допомогою служби пошуку (у складі Azure AI Search) з попередньою побудовою векторних подань (embeddings). Такий підхід дозволяє одночасно підтримувати класичний повнотекстовий пошук і семантичний пошук, що критично для RAG-сценаріїв на основі великих мовних моделей.

Другий тип даних — це тестові запити для оцінювання якості відповідей. Набір включає контрольні питання, які покривають різні категорії сценаріїв (просте витягнення факту, арифметика, політика повернення, безпека, агрегація кількох фактів тощо). Для кожного запиту збережені «еталонні» відповіді (ground truth) та очікувані властивості виходу (наприклад, чи має бути відмова, чи потрібні посилання на політику, чи допускається узагальнений опис).

Третю групу даних становлять журнали взаємодій (логи) з агентами, що виникають під час прогонів експериментів. Вони містять повну історію діалогу: вхідні запитання, проміжні виклики інструментів, остаточні відповіді, а також метадані (ідентифікатори агента, час виконання, тривалість, службові теги).

Окремо зберігаються конфігурації агентів і оркестрацій (system-prompt’и, налаштування temperature/top_p, правила маршрутизації, схеми виклику tools). Їх версіонування важливе для відтворюваності експериментів, адже навіть невеликі зміни в підказках або параметрах можуть суттєво вплинути як на якість відповідей, так і на вартість.

2.4.2. Представлення даних для агентів: knowledge stores та tools

Для доступу до бази знань агенти не працюють безпосередньо з HTML-файлами. Замість цього в інфраструктурі використовується шар «knowledge store» на базі Azure AI Search з векторним індексом. Кожен документ розбитий на невеликі фрагменти, для яких обчислені embeddings; ці фрагменти зберігають посилання на вихідну сторінку, тип контенту та інші атрибути. Під час виконання запиту агент не будує SQL- або REST-запит самостійно — він викликає інструмент (tool), який інкапсулює всю логіку пошуку.

У проєкті використовується декілька ключових інструментів. Перший — умовний knowledge_base_retrieve, що приймає один або кілька текстових intent'ів («знайди політику повернення для замороженого м'яса», «отримай інформацію про вартість доставки та поріг безкоштовної доставки» тощо) і повертає набір знайдених фрагментів з бази знань. Другий — службовий інструмент для переліку доступних tools та їхніх схем виклику (аналог mcp_list_tools), який дає агенту змогу «зрозуміти», які взагалі дії йому доступні в поточному середовищі.

Крім «чистих» інструментів для RAG, окремі агенти можуть використовувати допоміжні функції на кшталт перетворення дат, простих обчислень або форматування відповіді. Такі функції дозволяють розвантажити модель від рутинних операцій, підвищити точність там, де потрібна чітка арифметика чи робота з форматами, а також зробити вихідний текст більш структурованим і придатним для подальшого використання в інших системах (наприклад, у листах, звітах чи внутрішніх тікетах підтримки).

У термінах Azure OpenAI це реалізується через механізм function calling: модель отримує опис доступних функцій та їхніх параметрів у вигляді схем, сама вирішує, яку саме функцію доцільно викликати для поточного кроку міркування. Після цього агент може побудувати фінальну відповідь, поєднавши текстову частину з отриманими структурованими даними (наприклад, перерахованою сумою, правильно відформатованою датою

доставки або згенерованою табличною структурою).

Узагальнений перелік інструментів, їх призначення, основні вхідні параметри, очікуваний результат і відповідні агенти буде наведено у табл. 2.2.

Таблиця 2.2

Набір інструментів (tools), доступних агентам

Назва інструмента	Призначення	Вхідні параметри	Очікуваний результат	Використання агентами
knowledge_base_ retrieve	Пошук релевантних фрагментів у базі знань Taurbull на основі ітентів.	Список текстових ітентів; опціональні ліміти кількості результатів.	Набір фрагментів (текст, посилання на джерело, метадані впевненості).	Single-агент; доменні агенти (Shipping, Returns, Product, Other); агенти всіх типів оркестрації.
mcp_list_tools	Надання переліку доступних інструментів та схем їх виклику для планування дій.	Відсутні або службовий об’єкт запиту.	Детальний опис інструментів: назви, призначення, JSON-схеми параметрів.	Planner-агент; Single-агент (ініціалізація); Orchestrator (режим debug).
eval_logger	Логування результатів експерименту (метрики, відповіді) в Azure Storage.	IDs (experiment, agent, orchestration); текст відповіді; метрики (TaskCompletion, Groundedness).	Підтвердження запису (статус успіху або logId).	Evaluator-агент; тестовий оркестратор (Розділ 3).
config_loader	Завантаження актуальної конфігурації (system prompt, параметри моделі).	agentId або orchestrationId; версія конфігурації (опціонально).	Об’єкт конфігурації у форматі JSON.	Оркестратор; Evaluator (для фіксації параметрів запуску).

2.4.3. Експериментальний стенд в Azure

Для проведення дослідів створено окремий експериментальний стенд у хмарі Microsoft Azure. У межах однієї ресурсної групи розгорнуті такі основні компоненти: служба Azure OpenAI / Azure AI Foundry (моделі GPT-класу), індекс Azure AI Search (база знань Taurbull), обліковий запис сховища Azure Storage (для файлів знань, логів і конфігурацій), сховище секретів (Azure Key Vault) та сервіс моніторингу на базі Azure Monitor.

Azure Storage виконує роль центрального файлового сховища: у контейнерах зберігаються сирі веб-сторінки, нормалізовані текстові представлення, експериментальні лог-файли, проміжні результати оцінювання й аналітичні звіти. Використання єдиного сховища спрощує автоматизацію обробки, резервне копіювання й подальшу інтеграцію з інструментами аналітики.

Azure Monitor та пов’язаний із ним Log Analytics Workspace збирають телеметрію з усіх ключових компонентів: запити до моделей, тривалість виконання tools, кількість токенів, системні помилки. Ці ж журнали використовуються для побудови дашбордів, де можна порівняти якість і вартість різних оркестрацій (single-agent, послідовна, псевдоконкурентна, handoff).

На рис. 2.4 на високому рівні буде показано взаємозв’язок ресурсів у експериментальному стенді: користувачі й тестові скрипти, точка входу в Azure OpenAI, взаємодія з Azure AI Search для RAG, зберігання артефактів у Blob Storage, передача телеметрії до Azure Monitor/Log Analytics та використання Azure Cost Management для аналізу витрат.

Окремо експериментальний стенд ізольований від продуктивного середовища Taurbull за допомогою окремих subscription-ів та політик доступу, що дозволяє безпечно змінювати оркестрації, налаштування RAG і системні промпти без ризику впливу на реальних користувачів. Конфігурації для різних evaluation flow-ів (single, sequential, concurrent, handoff) зберігаються у вигляді

параметризованих шаблонів, тому один і той самий набір експериментів можна відтворити повторно, а результати — коректно порівнювати між собою в часі.

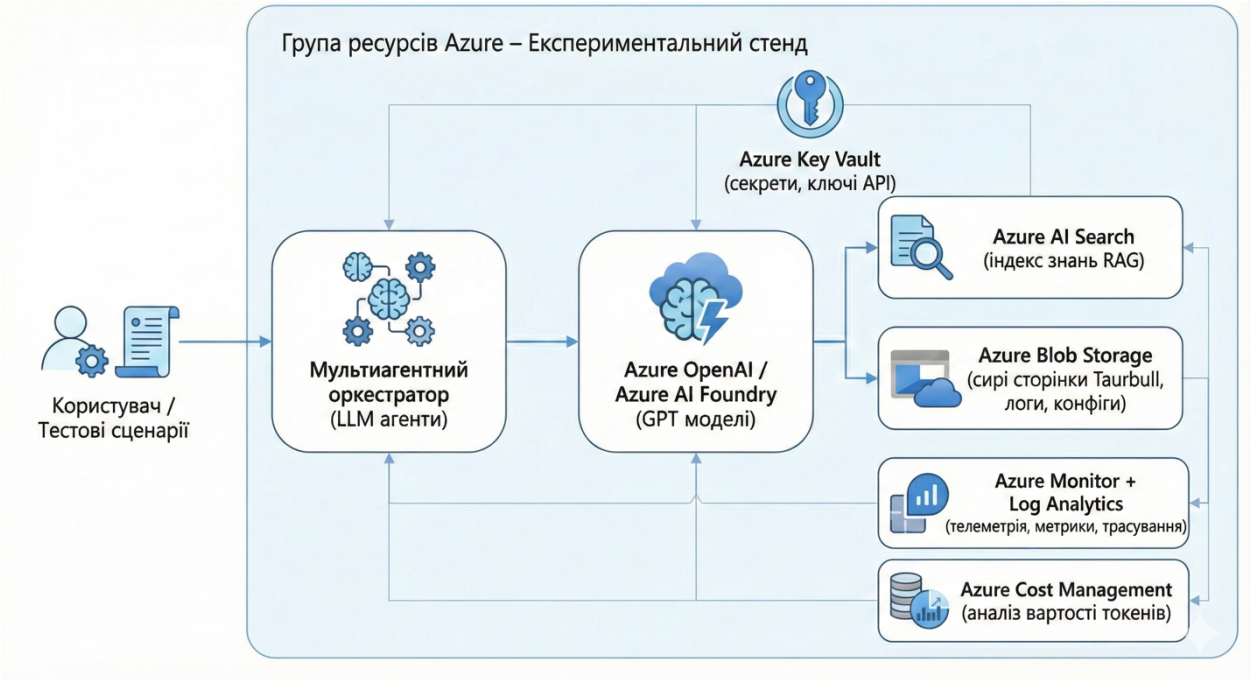

Рис 2.4. Схема експериментального стенду в Azure

Версіонування конфігурацій агентів і сценаріїв оркестрації реалізовано на рівні артефактів у сховищі (JSON-файли) та в системі контролю версій вихідного коду. Для кожного прогону зберігається явне посилання на версію конфігурації (наприклад, «single-agent:v2», «concurrent-flow:v4», «handoff:v3»), що дозволяє відтворювати експерименти й коректно порівнювати результати між собою.

2.4.4. Логи, телеметрія та метрики вартості

Оскільки робота з LLM-агентами є відносно дорогою, особливо в сценаріях з багатьма підзапитами й інструментами, критично важливо з самого початку організувати збір метрик вартості. У проєкті використовуються вбудовані можливості Azure Cost Management and Billing, які дозволяють

відстежувати споживання ресурсів по конкретних сервісах (Azure OpenAI, Azure AI Search, Storage, Monitor) та фільтрувати витрати за тегами ресурсів або ресурсними групами.

Додатково на рівні додатка реєструється деталізована інформація про кожен запит: ідентифікатор експерименту, назва оркестрації, кількість вхідних і вихідних токенів, кількість викликів tools, підсумкова оцінка (TaskCompletion, Groundedness тощо). Це дозволяє не лише співставляти якість відповіді з прямими витратами, а й виявляти неефективні шаблони (наприклад, зайві повторні виклики інструментів у деяких оркестраціях).

Azure Monitor використовується для централізованого збору телеметрії: усі логи, метрики й трасування зберігаються у Log Analytics Workspace, де над ними можна виконувати запити Kusto, будувати графіки та встановлювати алерти (наприклад, на аномальне збільшення часу відповіді або раптовий стрибок у кількості токенів). Це особливо важливо при довготривалих прогонах, коли одна і та сама оркестрація тестується на великій кількості запитів.

У підсумку, організація даних, інструментів і експериментального середовища в Azure забезпечує відтворюваність дослідів, прозорість доступу до знань для агентів та можливість об’єктивно вимірювати як якість, так і вартість різних архітектур мультиагентних систем, що буде детально проаналізовано в наступному розділі.

2.5. Критерії оцінювання та план експериментів з агентними патернами

У цьому підрозділі формалізуємо підхід до оцінювання мультиагентного рішення, описаного в попередніх підрозділах, та визначаємо план експериментів для порівняння різних патернів оркестрації агентів. Якщо в Розділі 1 метрики розглядалися на концептуальному рівні, то тут вони прив’язуються до конкретної реалізації на базі Azure OpenAI, Azure AI Search та супутніх сервісів.

2.5.1. Набір метрик та їх практичний зміст

Оцінювання системи проводитиметься за кількома групами метрик, які відображають якість відповідей, часові характеристики, вартість, надійність та потребу в ручному втручанні. Такий багатовимірний підхід узгоджується з сучасними рекомендаціями щодо комплексного тестування LLM-систем (наприклад, у рамках HELM та подібних досліджень, де важливо одночасно враховувати якість, robustness, безпеку та витрати).

По-перше, ключовою є метрика якості відповіді, яка в нашому випадку розбивається на кілька аспектів. Основною буде TaskCompletion – частка запитів, для яких система надала коректну й повну відповідь з точки зору бізнес-сценарію. Для кожного тестового запиту заздалегідь формується «еталонна» відповідь або набір критеріїв, що дозволяють віднести результат до категорії «успішно виконано» чи «виконано з помилками». Додатково використовуватиметься IntentResolution – показник того, наскільки система правильно зрозуміла намір користувача (особливо важливо в діалогах із неявними або багатокроковими вимогами).

По-друге, важливою є обґрунтованість (groundedness) відповіді, тобто ступінь опори моделі на надані знання (базу документів, політики, внутрішні дані), а не на «галюцинації». Для кожної відповіді фіксується, чи посилається агент на релевантні фрагменти з Azure AI Search / бази знань і наскільки ці фрагменти справді підтверджують твердження в тексті відповіді. Цей аспект корелює з підходами «LLM-as-a-judge», де спеціальний оцінювальний агент (часто більш потужна LLM) виступає арбітром якості та відповідності контексту.

По-третє, для практичного використання критично важливою є затримка (latency). У системі фіксуватиметься end-to-end latency – час від надсилання запиту користувачем до отримання фінальної відповіді. Додатково, де можливо, виділятимуться підметрики: час маршрутизації запиту (router-agent), час планування (planner-agent), час виконання інструментів (tools) та час

генерації тексту LLM. Основними узагальненими показниками будуть середній час відповіді (mean latency) та 95-й перцентиль (p95), які допомагають оцінити як типову швидкість системи, так і поведінку в «хвості» розподілу (повільні запити).

Четвертою групою є метрики вартості. Оскільки система побудована на хмарних LLM сервісах, основні витрати пов’язані з кількістю токенів, оброблених моделлю, та з викликами зовнішніх інструментів (наприклад, додаткові запити до Azure AI Search). Для кожного експерименту фіксуватимуться: сумарна кількість вхідних та вихідних токенів, приблизна грошова вартість (на основі актуальних тарифів Azure OpenAI), а також «вартість на успішне завдання» – відношення загальної вартості до кількості коректно виконаних запитів. Подібний підхід рекомендується в практичних гайдах щодо побудови та експлуатації генерувальних AI-рішень у хмарі.

П’ятою групою виступають метрики надійності та потреби в ручному втручанні. До них належить частка запитів, що завершилися технічною помилкою (timeout, помилки виклику інструментів, некоректний формат відповіді агента), а також частка відповідей, які маркуються як такі, що потребують ручної перевірки оператором (наприклад, коли впевненість системи низька або коли виявлено суперечність між відповіддю та джерелами). Ці метрики важливі з точки зору операційної придатності системи: навіть при високій середній якості занадто велика кількість «критичних винятків» робить рішення непридатним до продакшн-використання.

Нарешті, для аналізу «компромісів» між різними патернами інтегруватиметься інтегральна метрика ефективності, що поєднує якість, latency та вартість. Наприклад, можна розглядати відношення TaskCompletion до середньої вартості та середньої затримки, або будувати фронт Парето, де кожна конфігурація агентів відображається точкою в просторі «якість – latency – вартість». Таблиця 2.4 описує, як саме в рамках роботи будуть вимірюватися ключові характеристики мультиагентної системи на базі Azure OpenAI. [33-35]

Таблиця 2.3

Метрики оцінювання та їх практичні інтерпретації

Метрика (Metric)	Опис	Спосіб вимірювання	Прийнятні порогові значення
Groundedness (Обґрунтованість)	Визначає, наскільки відповідь агента базується виключно на наданому контексті (Knowledge Base), без «галюцинацій».	Автоматичні eval-и в Azure AI Studio (порівняння Answer vs Context).	≥ 0.85 (високий пріоритет для уникнення дезінформації).
Relevance (Релевантність)	Оцінює, наскільки відповідь відповідає запиту користувача (чи є вона корисною, навіть якщо фактично правильна).	Автоматичні eval-и в Azure AI Studio (порівняння User Query vs Answer).	≥ 4.0 (за 5-бальною шкалою) або ≥ 0.8.
Task Completion (Успішність виконання)	Частка діалогів, де користувач отримав вирішення проблеми без необхідності перемикання на людину.	Ручна оцінка вибірки або використання LLM-судді (LLM-as-a-judge) на основі логів діалогу.	≥ 0.4 для складних (multi-turn) сценаріїв; ≥ 0.7 для простих запитів.
Latency (Час відгуку)	Час від моменту відправки запиту користувачем до отримання повної відповіді (end-to-end).	Обчислення середнього значення (avg) та 95-го перцентиля (p95) за часовими мітками в логах.	$\le 10–15$ с (для забезпечення комфортного UX у чаті).
Cost per query (Вартість запиту)	Сумарна вартість токенів (вхідних та згенерованих) для обробки	Агрегація використаних токенів (prompt_tokens,	Мінімізація (порівняно з вартістю роботи

	одного повного сценарію.	completion_tokens) $\times$ тарифи Azure OpenAI.	оператора підтримки).
Handoff Rate (Частка ескалацій)	Відсоток випадків, коли агент не зміг впоратися і передав діалог людині або повернув помилку.	Аналіз логів на наявність виклику інструменту ескалації або статусів failed.	$\le 20\%$ (на етапі тестування MVP).

2.5.2. Гіпотези щодо поведінки різних патернів оркестрації

Для планування експериментів важливо чітко сформулювати очікування (гіпотези) щодо того, як різні патерни оркестрації агентів впливатимуть на метрики. Це дозволить не лише «зняти цифри», а й зіставити результати з попередніми інтуїціями та досвідом інших дослідників.

Перша гіпотеза стосується базового патерну з одним агентом (single-agent), який напряму отримує запит користувача, викликає інструменти та формує відповідь. Очікується, що такий підхід забезпечить мінімальну затримку і нижчу вартість (через меншу кількість кроків і повідомлень між агентами), але обмежену якість на складних багатокрокових запитах. Зокрема, прогнозується зниження TaskCompletion і Groundedness у сценаріях, де потрібно комбінувати кілька джерел знань, планувати послідовність дій або чітко розмежовувати етапи «пошук → аналіз → формування бізнес-висновку».

Друга гіпотеза пов’язана з ієрархічним патерном (hierarchical / planner–executor). У цій конфігурації один агент виконує роль планувальника (planner): розкладає складне завдання на підзадачі, визначає, які інструменти та/або спеціалізовані агенти потрібно залучити, а потім координує їхню роботу. Очікується, що такий підхід покращить TaskCompletion та TaskAdherence (ступінь дотримання інструкцій і бізнес-обмежень), особливо для складних запитів, які потребують кількох кроків. Водночас прогнозується зростання

затримки та вартості через збільшену кількість кроків, проміжних повідомлень та інструментальних викликів.

Третя гіпотеза стосується патерну з маршрутизатором (router) та набором спеціалізованих агентів. У цьому випадку окремий router-agent класифікує вхідний запит (наприклад, «просте звернення до бази знань», «складне запитання з політиками», «ризикований / потенційно токсичний контент» тощо) і скеровує його до відповідного агента чи конфігурації. Очікується, що для простих запитів такий патерн забезпечить оптимізацію вартості й latency (завдяки прямому маршруту до «легкого» агента), а для складних – дозволить обирати більш «важкі» конфігурації (наприклад, ієрархічні або з розширеною валідацією). Таким чином, router-підхід має потенціал забезпечити кращий компроміс між якістю та ефективністю на рівні всієї системи.

Четверта гіпотеза пов’язана з патерном із додатковим агентом-оцінювачем (evaluator / judge), який перевіряє відповіді інших агентів перед тим, як повернути результат користувачеві. Очікується, що такий «LLM-as-a-judge» підхід зменшить кількість помилкових або небезпечних відповідей і підвищить Groundedness, але водночас призведе до збільшення вартості та затримки, особливо якщо оцінювач аналізує кожен запит. У практичній конфігурації передбачається використання цього патерну лише для складних та ризикованих сценаріїв, що відповідає сучасним рекомендаціям щодо поєднання автоматизованих та людських оцінок. [33-34]

П’ята гіпотеза стосується стабільності та надійності. Очікується, що конфігурації з більшою кількістю агентів та інструментів будуть більш схильні до технічних збоїв (через зростання кількості точок відмови), але при цьому матимуть кращу «операційну прозорість», оскільки кожен крок легко логувати та аналізувати. Це має значення для подальшого вдосконалення системи: навіть якщо перша версія складнішої архітектури показує схожі цифри по TaskCompletion, її простіше оптимізувати й масштабувати.

2.5.3. План експериментів та організація процесу оцінювання

План експериментів побудовано навколо реалістичних сценаріїв підтримки клієнтів умовної компанії Taurbull у домені логістики та e-commerce. Для забезпечення репрезентативності розробляється набір тестових запитів, який охоплює різні типи завдань: від простих запитів до бази знань до складних кейсів із кількома кроками, політичними обмеженнями та потенційними ризиками безпеки. Подібний сценарний підхід рекомендується у сучасних оглядах методик оцінювання LLM-систем, де наголошується на важливості реалістичних та багатовимірних тестових наборів.

Кожен тестовий запит прив’язується до певного сценарію. Наприклад, частина запитів відображає просте отримання інформації (умовні «S1»-сценарії), де користувач очікує коротку конкретну відповідь на основі політик чи довідкових матеріалів. Інша група охоплює складні багатокрокові завдання (аналог «S2»), коли потрібно, наприклад, проаналізувати кілька документів, зіставити правила та запропонувати рішення щодо повернення товару або зміни умов доставки. Окремо виділяються сценарії з можливими «пастками» в політиках (аналог «S3»), де коректна відповідь потребує уважного дотримання бізнес-обмежень, а також сценарії з ризиком небезпечного чи небажаного контенту (аналог «S4»), де важливо протестувати фільтрацію та безпекові механізми системи.

Усі експерименти проводитимуться в заздалегідь підготовленому середовищі Azure, описаному в підрозділі 2.4. Для кожної конфігурації агентів буде створено окремий набір налаштувань (prompt-шаблони, параметри моделей, конфігурація tools), що дозволить повторно запускати експерименти та порівнювати результати в однакових умовах. Логи запитів, відповіді агентів, проміжні кроки (маршрутизація, плани, виклики інструментів) та метрики latency/вартості будуть зберігатися в централізованому сховищі логів (наприклад, Azure Application Insights або Log Analytics), що відповідає типовим рекомендаціям Microsoft щодо експериментування з LLM-

рішеннями. [33-35]

З метою аналізу результатів експериментів планується формувати кілька узагальнених зрізів, як показано на рис. 2.5:

- порівняння патернів за всіма метриками на повному наборі сценаріїв;
- окремий аналіз для простих vs складних завдань;
- аналіз «вартість–якість» у розрізі сценаріїв;
- виявлення конфігурацій, які найкраще підходять для продакшн-використання (наприклад, «router + evaluator тільки для ризикованих кейсів»).

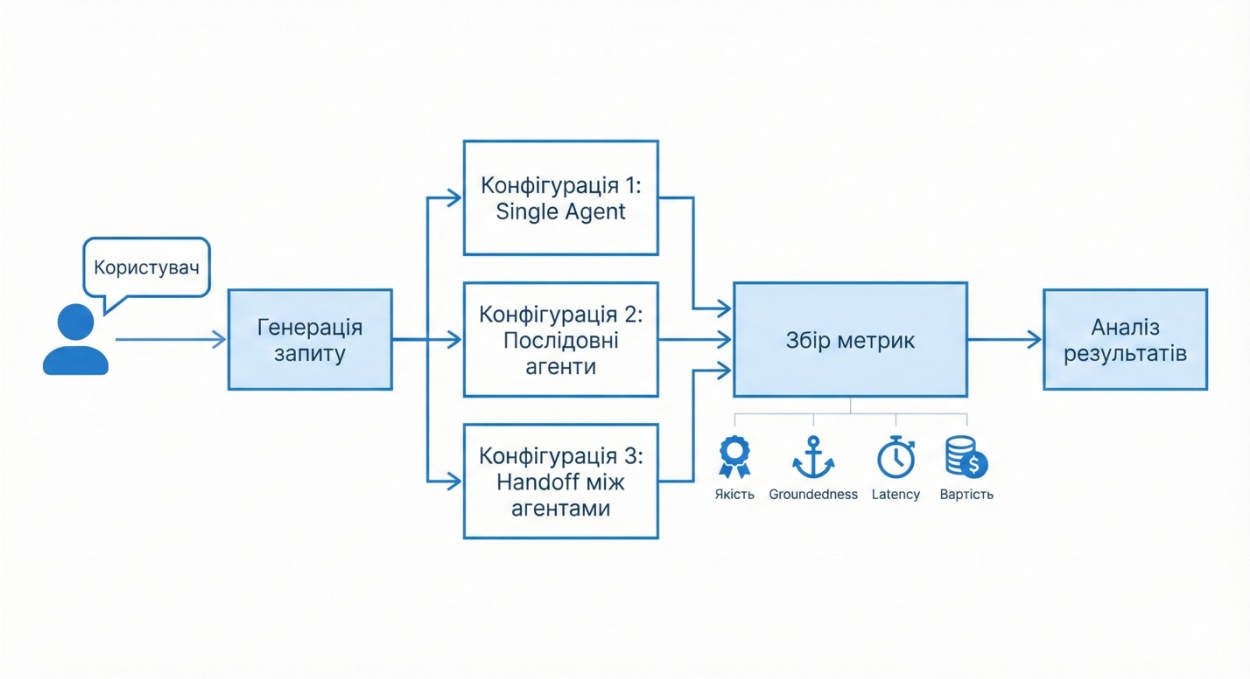

Рис 2.5. Схема експериментального процесу

Таким чином, у рамках цього підрозділу сформовано чіткий набір метрик та гіпотез, а також структурований план експериментів, які дозволять у Розділі 3 перейти до кількісного порівняння різних агентних патернів і зробити обґрунтовані висновки щодо їх придатності для реальних бізнес-сценаріїв.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПАТЕРНІВ ОРКЕСТРАЦІЇ В AZURE

3.1 Конфігурації патернів оркестрації

Базовою точкою відліку для всіх подальших експериментів у роботі є одиночний LLM-агент, який напряму обробляє запити користувача та звертається до корпоративної бази знань Taurbull через RAG-шар. Така конфігурація реалізована засобами Azure OpenAI (модель *gpt-4o* у складі Azure AI Foundry) та знань, опублікованих в Azure AI Search / Azure AI Studio у вигляді knowledge base. Це відповідає типовим рекомендаціям Microsoft щодо побудови генеративних застосунків з шаром пошуку по власних даних [22; 23; 24; 25].

3.1.1. Еталонна конфігурація single-agent з RAG-шаром

Логіка роботи еталонного single-agent така: користувач формулює природномовний запит, агент аналізує його, за потреби викликає інструмент доступу до бази знань Taurbull, отримує релевантні фрагменти документів та на їх основі синтезує фінальну відповідь. При цьому агент відповідає українською/німецькою (залежно від запиту), дотримується обмежень безпеки та не вигадує фактів поза наданими документами. Усі подальші патерни (sequential, concurrent, handoff, planner–executor) будуть надбудовуватися саме над цією базовою конфігурацією.

Технічно конфігурація агента у середовищі Azure AI Agent Service задається у вигляді YAML-опису версії агента. У цьому описі можна виділити три основні частини. По-перше, блок metadata, де фіксуються службові атрибути (ідентифікатор, назва, версія, дата створення) та, за потреби, візуальні налаштування для інтерфейсу. По-друге, блок definition, у якому

описується тип агента (kind: prompt), цільова модель (model: gpt-4o), системні інструкції (instructions). По-третє, список tools, через який агент отримує доступ до зовнішніх ресурсів – у нашому випадку до knowledge base на базі Azure AI Search через MCP-сервер. У листингу 3.1 наведено спрощений варіант YAML-конфігурації еталонного агента, який використовується в експериментах. У реальній системі інструкції дещо розширені, але для цілей дипломної роботи достатньо показати узагальнену структуру.

```
metadata:
  logo: Avatar_Default.svg
  description: "Baseline Taurbull support agent"
  modified_at: "1763826895"

object: agent.version
id: default-test:2
name: default-test
version: "2"
description: "Single-agent configuration with access to Taurbull knowledge
base"
created_at: 1763826896

definition:
  kind: prompt
  model: gpt-4o

  # Скорочена системна інструкція агента:
  instructions: |
    You are a customer support agent for Taurbull, an online shop for organic
beef.
    Answer user questions based only on the official Taurbull documentation
    (shipping, returns, FAQ, farming standards). If the information is
missing,
    say that you are not sure and suggest contacting human support.
    Always respond in the same language as the user.

  # Інструменти, доступні агенту (RAG-шар поверх knowledge base):
  tools:
    - type: mcp
      server_label: kb_taurbull_3_pages_kb_f1hif
      server_url: >-
        https://diplom-search-service.search.windows.net/knowledgebases/
        taurbull-3-pages-kb/mcp?api-version=2025-11-01-Preview
            project_connection_id: kb-taurbull-3-pages-kb-f1hif
```

Лістинг 3.1. Базова конфігурація single-agent у форматі YAML

У цьому прикладі блок definition.model вказує на використання моделі gpt-4o, розгорнутої в Azure OpenAI Service як керованого ресурсу в межах підписки. Значення kind: prompt означає, що агент працює як “prompt-орієнтований” – основна логіка визначається системною інструкцією та

доступними інструментами, без додаткового коду всередині середовища виконання [22; 24]. Текст системних інструкцій окреслює роль агента (служба підтримки магазину Taurbull), доступний простір знань (офіційні сторінки про доставку, повернення, FAQ та стандарти фермерства) та вимоги до стилю відповіді.

Найважливішим елементом для підтримки RAG-підходу є запис у блоці tools з типом mcp. MCP-сервер, на який посилається агент (server_url і project_connection_id), розгорнуто поверх індексу Azure AI Search, що містить три веб-сторінки Taurbull, попередньо завантажені та проіндексовані як knowledge base. Таким чином, коли модель в процесі розв'язання запиту вирішує, що їй потрібні додаткові факти, вона може викликати інструмент knowledge_base_retrieve цього MCP-сервера, отримати список релевантних фрагментів тексту та включити їх у відповідь. Такий спосіб доступу до даних узгоджується з типовою архітектурою RAG-рішень на базі Azure AI Search + Azure OpenAI [23; 25].

Завдяки виділенню конфігурації агента в окремий YAML-опис спрощується керування версіями. У процесі дослідження було створено декілька версій еталонного single-agent з різними параметрами температури, обмеженнями на довжину відповіді та варіаціями системних інструкцій, але всі вони ґрунтувалися на однаковому шаблоні. У третьому розділі саме версія default-test:2 використовується як основна "еталонна" конфігурація, відносно якої порівнюються складніші патерни оркестрації. Це дозволяє інтерпретувати відмінності в метриках (якість, затримка, вартість) як наслідок зміни архітектури агентної системи, а не вихідних даних чи моделей.

3.1.2. Послідовний (sequential) патерн перевірки й оформлення відповіді

Наступним кроком після базового single-agent є побудова простого послідовного (sequential) workflow, у якому над кожним запитом послідовно працюють два окремі агенти. Такий підхід дає змогу розділити

відповідальність між спеціалізованими ролями: перший агент концентрується на пошуку та структурованому виділенні фактів із бази знань, другий — на формуванні фінальної користувацької відповіді з урахуванням цих фактів і політик сервісу.

У реалізованій конфігурації використано два агенти: researcher-seq-v2 та policy-checker-seq. Агент *researcher-seq-v2* виступає в ролі “research agent” і працює поверх Azure AI Search. Він отримує оригінальний запит користувача, викликає індекс знань та повертає структурований текст, що містить лише фактичну інформацію з knowledge base: повтор запитання й окремий блок із виписаними фактами. На цьому етапі ще не формується “готова” відповідь для користувача — лише фактологічна основа, з якою можна працювати далі.

Агент *policy-checker-seq* виконує функції “final answer agent”. На вхід він отримує результат першого агента — текст, що містить оригінальне запитання й перелік знайдених фактів. Його завдання — сформувати фінальну відповідь природною мовою, не згадуючи про внутрішню структуру (“факти з бази”, “інший агент” тощо) і суворо дотримуючись того, що прямо випливає з фактів. Якщо інформації недостатньо, агент має чесно вказати на це у відповіді.

На рівні оркестрації Azure AI Agent Service ця логіка описується окремим workflow, який запускається при старті розмови. На рис. 3.1 доцільно зображена ця схема як ланцюжок із двох прямокутників “Research Agent” і “Final Answer Agent”, між якими передається проміжне повідомлення, а також вхідний та вихідний інтерфейси “Користувач → Чат-бот Taurbull”.

Рис 3.1 – Послідовна обробка запиту в Azure AI Agent Service

Нижче наведено скорочений фрагмент YAML-опису workflow, який реалізує цей послідовний патерн. Повні службові поля (ідентифікатори дій, додаткові метадані) у дипломі опущено для стислості.

```
kind: workflow
name: test-test-seq
description: "Послідовна обробка запиту двома агентами"

trigger:
  kind: OnConversationStart
  id: trigger_wf
  actions:
    - kind: InvokeAzureAgent
      agent:
        name: researcher-seq-v2
      input:
        messages: =System.LastMessage
      output:
        autoSend: false
        messages: Local.Research

    - kind: InvokeAzureAgent
      agent:
        name: policy-checker-seq
      input:
        messages: =Local.Research
      output:
        autoSend: true
```

Лістинг 3.2. YAML-конфігурація послідовного workflow з двома агентами

Перша дія workflow викликає агента researcher-seq-v2, передаючи йому останнє повідомлення користувача (System.LastMessage). Результат зберігається у локальній змінній Local.Research і не надсилається користувачу (autoSend: false). Друга дія викликає агента policy-checker-seq, який отримує на вхід саме це проміжне повідомлення й вже повертає фінальну відповідь у чат (autoSend: true). Таким чином, логіка послідовності агентів задається декларативно на рівні workflow, тоді як кожен окремий агент має власну конфігурацію й інструкції.

Сам опис агентів також задається у вигляді YAML. Нижче показано скорочені варіанти конфігурацій, у яких збережено структуру та ключові поля, але системні інструкції суттєво укорочено (повний текст промптів у реальному проєкті значно довший; у дипломі їх подано фрагментарно).

```
object: agent.version
id: researcher-seq-v2:11
name: researcher-seq-v2
definition:
  kind: prompt
  model: gpt-4o
  instructions: |
    You are a Research Agent.
    Use Azure AI Search to look up the user's question
    and return only factual information in two blocks:
    "user question:" and "facts from knowledge base:".
  temperature: 0.7
  tools:
    - type: azure_ai_search
      azure_ai_search:
        indexes:
          - project_connection_id: <connection-to-diplomsearchservice>
            index_name: rag-1763823478151
            query_type: simple
            top_k: 5
  text:
    format:
      type: text
```

Лістинг 3.3. Скорочена YAML-конфігурація агента researcher-seq-v2

У цьому агенті (researcher-seq-v2:11) основний акцент робиться на використанні інструмента azure_ai_search, який підключений до відповідного індексу знань Taurbull. Промпт чітко задає формат виходу, щоб другий агент міг легко його інтерпретувати.

```
object: agent.version
id: policy-checker-seq:7
name: policy-checker-seq
definition:
  kind: prompt
  model: gpt-4o
  instructions: |
    You are the Final Answer Agent.
    Input always contains:
    "user question:" and "facts from knowledge base:".
    Treat the facts as the only source of truth
    and write a natural answer for the user
    without mentioning agents or knowledge base.
    If facts are not enough, say that clearly.
  temperature: 0.7
  tools: []
  text:
    format:
      type: text
```

Лістинг 3.4. Скорочена YAML-конфігурація агента policy-checker-seq

Агент policy-checker-seq не має власних tools і працює виключно з текстом, сформованим першим агентом. Така декомпозиція дозволяє чітко розділити етапи “пошуку фактів” і “формування відповіді”, а також у подальших експериментах (розділи 3.3–3.4) порівнювати якість та надійність sequential-патерну з іншими варіантами оркестрації, не змінюючи саму базу знань або модель.

3.1.3. Паралельний (concurrent) патерн

У цьому підрозділі розглядається так званий «паралельний» патерн, який у межах Azure AI Agent Service реалізовано у вигляді workflow concurrent-flow. Ідея патерну полягає в тому, що один і той самий запит опрацьовується кількома дослідницькими агентами (Agent A та Agent B), а потім їхні результати узагальнюються окремим агентом-агрегатором. На рівні оркестрації в рамках поточної версії сервісу виклики виконуються послідовно, однак логічно конфігурація розглядається як спільна робота двох дослідницьких агентів над однією задачею з подальшим агрегуванням.

Після надходження нового повідомлення користувача його текст зберігається у змінній Local.UserPrompt. Далі, як показано на рис. 3.2, викликається агент concurrent-sub-1 (Agent A), який на базі Azure AI Search формує фактологічну відповідь з мінімальною креативністю (temperature і top_p встановлені на рівні 0.01). Результат цього агента передається агенту concurrent-sub-2 (Agent B), який бачить як початкове запитання, так і відповідь Agent A, доповнює її власним аналізом та формує узагальнений текстовий блок для фінального агента. Останнім у ланцюжку діє агент concurrent-aggregator, який перетворює внутрішній технічний формат на природну відповідь для користувача.

Рис 3.2. Логічна схема «паралельного» патерну з двома дослідницькими агентами та агрегатором

Стрілка від користувача йде до «Research layer», потім дві стрілки (або маркування, що обидва агенти працюють над одним запитом), і далі консолідація в агрегатор. Таким чином рисунок відображає логічну паралельність (два незалежні дослідницькі погляди на один запит), не суперечачи фактичній послідовній реалізації оркестрації.

Сам workflow у скороченому вигляді описується наступним YAML-фрагментом (службові поля та метадані опущені для стислості):

```
kind: workflow
name: concurrent-flow
description: ""
trigger:
  kind: OnConversationStart
  id: trigger_wf
  actions:
    - kind: SetVariable
      id: action-1763846473234
      variable: Local.UserPrompt
      value: =System.LastMessage.Text

    - kind: InvokeAzureAgent       # Agent A – перший дослідницький агент
      id: action-1763846495652
      agent:
        name: concurrent-sub-1
      input:
        messages: =Local.UserPrompt
```

```
      output:
        autoSend: false
        messages: Local.Sub1

    - kind: InvokeAzureAgent      # Agent B – другий дослідницький агент
      id: action-1763846673625
      agent:
        name: concurrent-sub-2
      input:
        messages: =Local.Sub1
      output:
        autoSend: false
        messages: Local.Sub2

    - kind: InvokeAzureAgent      # Final answer / aggregator
      id: action-1763847212092
      agent:
        name: concurrent-aggregator
      input:
        messages: =Local.Sub2
      output:
        autoSend: true
```

Перший підагент concurrent-sub-1 (Agent A) спеціалізується на знятті фактів із бази знань Taurbull. Він не взаємодіє з користувачем напряму, а формує структурований блок для подальшої внутрішньої обробки. У тексті роботи доцільно навести скорочений варіант промпта з коментарем, що повна інструкція міститься у конфігураційних файлах проєкту:

```
object: agent.version
name: concurrent-sub-1
version: "8"
definition:
  kind: prompt
  model: gpt-4o
  instructions: |-
    You are Agent A, a research assistant for Taurbull.
    Your job is to read the user's question and look up all relevant
facts.
    You prepare information only for internal agents, not for the
customer.

    USER QUESTION:
    <copy the user's question>

    AGENT A ANSWER:
    <factual answer based only on the knowledge base, in English>

    # Full prompt is shortened in the thesis; complete text is stored in
the project config.
  temperature: 0.01
  top_p: 0.01
  tools:
    - type: azure_ai_search
      azure_ai_search:
        indexes:
```

```
            - project_connection_id:
/subscriptions/.../connections/diplomsearchserviceif1hif
              index_name: rag-1763823478151
        query_type: simple
```

Другий підагент concurrent-sub-2 (Agent B) виконує функцію «peer review» для першого дослідника. Він отримує оригінальний запит, повну відповідь Agent A та додає власні пояснення, уточнення або виправлення, також спираючись на Azure AI Search. У підсумку формується текст, який містить як вихідні факти Agent A, так і додатковий аналіз Agent B, і передається до агента-агрегатора.

Агент concurrent-aggregator завершує ланцюжок, перетворюючи внутрішню структуру «USER QUESTION / AGENT A ANSWER / AGENT B ANSWER» на одну цілісну відповідь у тональності служби підтримки Taurbull. У разі, якщо в отриманому тексті немає достатньої інформації для впевненої відповіді, агрегатор прямо вказує на це та пропонує звернутися до офіційної підтримки електронною поштою.

З погляду архітектури LLM-агентів така конфігурація дає змогу поєднати переваги кількох незалежних дослідницьких «поглядів» на запит і зменшити ймовірність випадкових помилок окремого агента. Водночас ціна цього підходу — збільшення затримки та вартості обробки порівняно з простим послідовним патерном, що надалі враховується в експериментальній частині оцінювання різних агентних конфігурацій.

3.1.4. Handoff / router-патерн

У цьому підрозділі розглядається handoff / router-патерн, у якому логіка маршрутизації запитів винесена в окремий «центральний» агент, а доменно-специфічна обробка виконується спеціалізованими підагентами. Такий підхід добре підходить для сценаріїв, де існують чіткі категорії намірів користувача (returns, shipping, product, other), для яких можуть бути налаштовані власні промпти, інструменти та стиль відповіді.

В Azure AI Agent Service цей патерн реалізовано у вигляді workflow handoff, як показано на рис. 3.2. Після надходження нового повідомлення тригер OnConversationStart викликає агент-router handoff-router. Його завдання – прочитати текст запиту й повернути рівно один маркер наміру: [RETURN], [SHIPPING], [PRODUCT] або [OTHER]. Важливо, що цей агент не взаємодіє з користувачем напряму: він лише класифікує запит, а сформований результат далі використовується блоком ConditionGroup для вибору відповідного спеціалізованого агента.

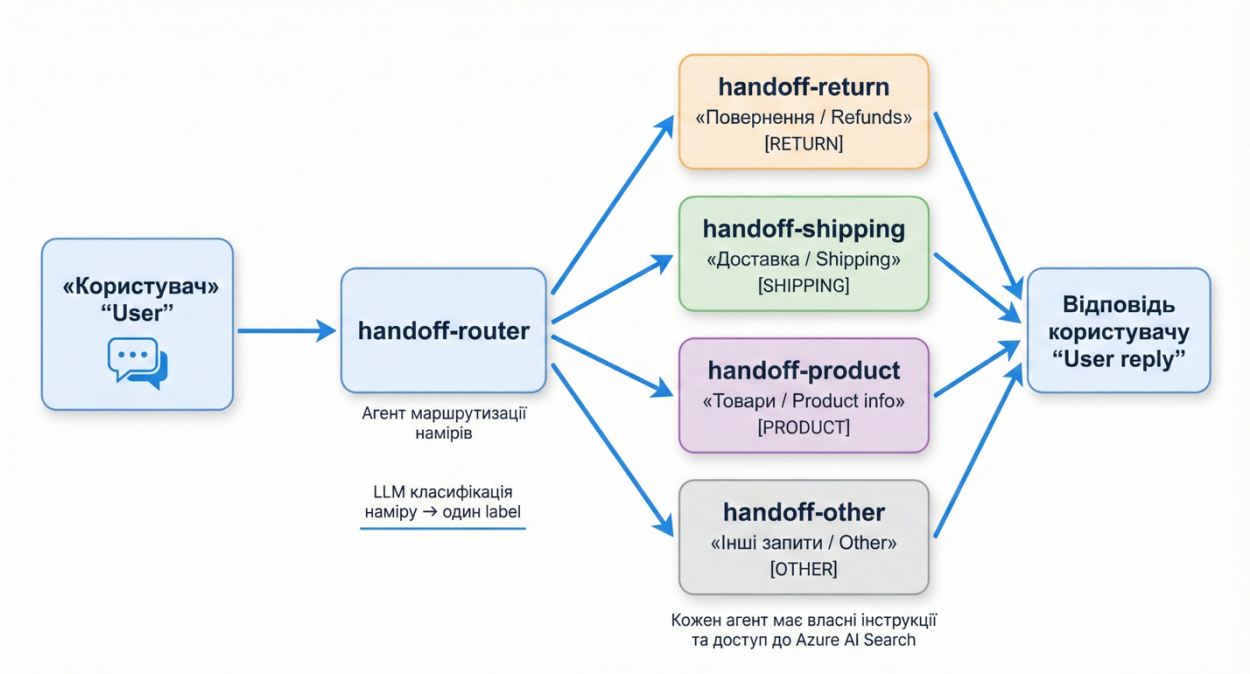

Рис. 3.3. Router-патерн з окремим агентом маршрутизації та доменними підагентами

На рівні конфігурації workflow описується скороченим YAML-фрагментом (службові поля й частина ідентифікаторів опущені для стислості):

```
kind: workflow
name: handoff
description: ""
trigger:
  kind: OnConversationStart
  id: trigger_wf
  actions:
    - kind: InvokeAzureAgent        # router-агент визначає намір
      id: action-router
      agent:
        name: handoff-router
```

```
            input:
              messages: =System.LastMessage
            output:
              autoSend: false
              messages: Local.LatestMessage

          - kind: ConditionGroup          # розгалуження за наміром
            id: action-intent-switch
            conditions:
              -             condition:             =!IsBlank(Find("[RETURN]",
Upper(Last(Local.LatestMessage).Text)))
                actions:
                  - kind: InvokeAzureAgent
                    agent:
                      name: handoff-return
                    input:
                      messages: =System.LastMessage
                    output:
                      autoSend: true

              -            condition:            =!IsBlank(Find("[SHIPPING]",
Upper(Last(Local.LatestMessage).Text)))
                actions:
                  - kind: InvokeAzureAgent
                    agent:
                      name: handoff-shipping
                    input:
                      messages: =System.LastMessage
                    output:
                      autoSend: true

              -             condition:             =!IsBlank(Find("[PRODUCT]",
Upper(Last(Local.LatestMessage).Text)))
                actions:
                  - kind: InvokeAzureAgent
                    agent:
                      name: handoff-product
                    input:
                      messages: =System.LastMessage
                    output:
                      autoSend: true
            elseActions:
              - kind: InvokeAzureAgent
                agent:
                  name: handoff-other
                input:
                  messages: =System.LastMessage
                output:
                  autoSend: true
```

Роутер-агент handoff-router реалізовано як окремий agent.version з невеликим, але дуже структурованим промптом. Він отримує повідомлення користувача, зіставляє його з описом чотирьох категорій намірів і повертає рівно один маркер. Повна інструкція є більш розгорнутою, однак у тексті диплома достатньо навести скорочений варіант:

```
      object: agent.version
      name: handoff-router
      version: "2"
      definition:
        kind: prompt
        model: gpt-4o
        instructions: |-
          You are an intent router for the Taurbull customer assistant.
          Read the user's message and choose exactly ONE intent:

            [RETURN]   - returns, refunds, complaints, exchanges.
            [SHIPPING] - shipping, delivery time, tracking, frozen shipping
issues.
            [PRODUCT]  - product questions, cuts, quality, preparation, storage.
            [OTHER]    - account, payment, general questions, everything else.

          Your output MUST be exactly one label:
            [RETURN] or [SHIPPING] or [PRODUCT] or [OTHER]

          No explanations, no extra text.
          # Full prompt is shortened in the thesis; complete text is stored in
the project config.
        tools: []
```

Далі спрацьовує ConditionGroup, який аналізує текст останнього повідомлення, повернутого роутером, і запускає відповідного спеціалізованого агента. Наприклад, конфігурація агента handoff-return задає роль «Return & Refund specialist» і підключає Azure AI Search як джерело політик повернення та відшкодування Taurbull. Аналогічно налаштовані агенти handoff-shipping, handoff-product та handoff-other, але з іншими доменними інструкціями та, за потреби, іншими наборами індексів чи інструментів. Завдяки цьому логіка бізнес-процесів по кожній категорії може розвиватися незалежно, без змін у центральному workflow.

Такий router-патерн демонструє, як засоби Azure AI Agent Service дають змогу поєднати класифікацію наміру на основі LLM та просте, читабельне розгалуження у вигляді ConditionGroup. Це спрощує підтримку та розширення системи: для додавання нового каналу обробки достатньо додати новий intent-лейбл, відповідний спеціалізований агент і ще одну гілку в конфігурації, не змінюючи загальну оркестраційну структуру.

3.2. Методика експериментів та конфігурація оцінювання

У цьому підрозділі конкретизується експериментальний дизайн, описаний у розділі 2, і пояснюється, як саме порівнювалися різні патерни оркестрації агентів у середовищі Azure AI Foundry. Основна ідея полягала в тому, щоб зміни стосувалися лише способу оркестрації (single, sequential, concurrent, handoff), а не моделі, джерел знань чи параметрів RAG. Завдяки цьому відмінності в показниках якості можна інтерпретувати саме як наслідок зміни патерна, а не змін у даних чи конфігурації моделі.

Для кожного патерна було створено окремий inference-flow, який приймає запит користувача, запускає відповідний набір агентів і повертає фінальну відповідь. Поверх цих inference-flow в Azure AI Foundry були побудовані evaluation-flow, що проганяють однаковий набір із 15 тестових сценаріїв та автоматично обчислюють якісні й безпекові метрики за схемою LLM-as-a-judge.

3.2.1. Набір тестових сценаріїв і класи складності

Набір тестів складається з 15 промптів, які імітують реальні звернення до служби підтримки інтернет-магазину Taurbull. Усі запити побудовано на основі трьох веб-сторінок, що увійшли до knowledge base: політика повернення, загальний FAQ та інформація про стандарти ведення фермерського господарства. Це дозволяє контролювати повноту джерел і водночас перевіряти, наскільки різні патерни оркестрації вміють «діставати» потрібні фрагменти саме з цієї корпоративної бази знань.

За рівнем складності сценарії поділено на дві групи. До простих віднесено ті, де відповідь здебільшого міститься в одному або двох абзацах документа і не потребує розширеної інтерпретації. Це, наприклад, прямі FAQ-запитання про строки повернення, умови замороженого м’яса або наявність права на відмову від посилки. Такі кейси перевіряють базову здатність патерна

правильно сформулювати RAG-запит, знайти релевантний фрагмент і перефразувати його в тональності служби підтримки.

Складні сценарії моделюють багатокрокові або «кутові» випадки. Сюди входять запитання з декількома умовами, наприклад: поєднання різних типів товарів в одному замовленні, часткове розморожування продукції, змішання онлайн-та офлайн-каналів продажу. Окремі промпти містять навмисно нечіткі формулювання або приховані припущення, щоб перевірити, чи зможе агент уточнити умови або коректно відмовити у відповіді, якщо в політиці прямо не описано потрібний випадок. Саме в цих сценаріях особливо важливі метрики IntentResolution та TaskCompletion, оскільки модель легко може «зайти не туди» або вигадати неіснуючі правила.

Остаточно набір із 15 промптів було збалансовано так, щоб приблизно половина сценаріїв виглядала як типовий «фронт-офісний» FAQ, а друга половина — як більш критичні кейси з точки зору ризиків бізнесу та політик (повернення швидкопсувних товарів, граничні строки, відсутність права на відмову згідно з законодавством тощо). Тексти самих промптів у роботі не дублюються повністю, щоб не перевантажувати основний текст; за потреби їх можна винести в додаток або окрему таблицю у вигляді «ідентифікатор сценарію – короткий опис – клас складності».

(На цьому місці доцільно вставити невеликий рисунок з діаграмою, що показує розподіл 15 сценаріїв за класами «простий/складний» та за типами запитів: FAQ, політика повернення, edge-cases. Це допоможе візуально зафіксувати склад набору.)

3.2.2. Спільні параметри RAG та моделей для всіх патернів

Щоб забезпечити коректність порівняння, для всіх чотирьох патернів оркестрації використовувалася одна й та сама модель gpt-4o, розгорнута в Azure OpenAI Service як керований ресурс. Відмінності між агентами стосувалися лише ролей та системних інструкцій; сама модель, її версія та

обмеження за контекстом залишалися незмінними. Це означає, що будь-які відмінності в метриках IntentResolution, Groundedness чи TaskCompletion можна інтерпретувати як наслідок оркестрації, а не зміни моделі.

RAG-шар також був уніфікований. Усі агенти, які мають доступ до зовнішніх даних, використовували один і той самий knowledge base на базі Azure AI Search / Azure AI Studio, побудований з трьох веб-сторінок Taurbull. Конфігурація інструмента azure_ai_search відповідала YAML-фрагментам, наведеним у підрозділі 3.1: індекс rag-1763823478151, однаковий project_connection_id і параметр top_k = 5 для вибірки максимум п’яти релевантних фрагментів. Вмикався один і той самий режим запиту (simple query поверх індексу з гібридним пошуком), а фільтри за метаданими не змінювалися між патернами.

У межах кожного патерна окремі агенти могли мати різні температури та top_p (наприклад, «дослідник» із temperature ≈ 0.01 для більш детермінованої фактології та «фінальний відповідач» із temperature ≈ 0.7 для більш природної мови). Однак ці відмінності повторювалися в однаковій конфігурації для всіх запусків відповідного патерна і не змінювалися між експериментами. Таким чином, спільними для всієї серії дослідів залишалися: набір джерел, схема RAG-доступу, модель gpt-4o та основні ліміти за довжиною контексту.

3.2.3. Evaluation flows в Azure AI Foundry та набір метрик

Оцінювання виконувалося засобами Azure AI Foundry через окремі evaluation-flow, кожен із яких прив’язано до свого inference-flow. Було створено чотири evaluation-конвеєри: для еталонної одноагентної конфігурації, для послідовного (sequential) патерна, для паралельного (concurrent) та для патерна з динамічними передачами (handoff). Усі вони використовують одну і ту саму таблицю з 15 сценаріями як тестовий датасет.

У складі evaluation-flow застосовувалися вбудовані LLM-evaluators, які

реалізують підхід «LLM-as-a-judge». Для кожної пари «запит – відповідь» окремий оцінювач виставляє бінарний або градуйований вердикт, після чого Azure агрегує ці вердикти на рівні всього запуску. Основні метрики якості, що використовуються у роботі, такі: IntentResolution (чи коректно система розпізнала намір користувача), TaskAdherence (чи дотримано інструкцій і політик у відповіді), Groundedness (чи спираються твердження на надані джерела) та TaskCompletion (чи вважається завдання користувача виконаним з точки зору оцінювача).

Окремо вмикалися безпекові метрики Violence та Hate & Unfairness, які сигналізують про наявність неприйнятного контенту, а також TextSimilarity — ступінь схожості відповіді на референсну, якщо вона задана. У контексті цієї роботи TextSimilarity використовувалася скоріше як контрольний індикатор для простих сценаріїв, де референсну відповідь можна сформулювати однозначно; для складних завдань основний акцент робився саме на TaskCompletion та Groundedness.

Усі відсотки в дашбордах Azure інтерпретуються як частка успішних кейсів від загальної кількості сценаріїв. Наприклад, значення Groundedness = 87 % відповідає 13 із 15 сценаріїв, де оцінювач визнав твердження у відповіді належно заземленими на джерела; TaskCompletion = 40 % означає, що лише 6 із 15 відповідей були визнані повністю такими, що розв’язують задачу користувача. Ці агреговані значення відображаються у вигляді «X % (Y / 15)» у таблиці результатів для кожного evaluation-run.

3.2.4. Процедура запуску експериментів і обробка результатів

Експерименти проводилися серіями запусків evaluation-flow у порталі Azure AI Foundry. Для кожного з чотирьох патернів запускалася окрема серія, у межах якої усі 15 сценаріїв проганялися через відповідний inference-flow. Таким чином, один evaluation-run відповідає повному прогону датасету певною конфігурацією оркестрації. Ідентифікатори run-ів, згенеровані Azure,

фіксувалися у журнальних записах, щоб за потреби можна було повернутися до детального трейсингу по кожному кейсу.

Після завершення кожного запуску результати аналізувалися у два рівні. На верхньому рівні використовувалися агреговані відсотки за метриками IntentResolution, TaskAdherence, Groundedness, TaskCompletion, Violence, Hate & Unfairness та TextSimilarity — саме ці значення виводяться в підсумковій таблиці дашборда і використовуються в подальших підрозділах для побудови порівняльних висновків між патернами. На нижньому рівні, для окремих цікавих сценаріїв, виконувалося ручне ознайомлення з відповідями агентів та рішеннями evaluators, щоб зрозуміти причини провалу чи успіху.

Оскільки всі чотири evaluation-flow посилаються на однаковий датасет і спільну конфігурацію RAG, додаткові нормалізації не знадобилися: достатньо безпосередньо порівнювати частки успішних кейсів між run-ами. Для уникнення випадкових флуктуацій, пов’язаних із стохастичністю моделі, частина запусків повторювалася, але в основний аналіз потрапили ті результати, що зображені на скріншотах з порталу Azure (по одному узагальненому run-у на кожен патерн). Саме ці агреговані значення у відсотках будуть надалі інтерпретуватися в підрозділах 3.3–3.4 як емпірична основа для порівняння single, sequential, concurrent та handoff-оркестрацій.

3.3. Результати еталонної single-agent конфігурації

На цьому етапі було проаналізовано результати еталонної конфігурації з одним агентом, яка використовує той самий RAG-ланцюжок та ту саму базу знань, що й multi-agent оркестрації, але виконує всі кроки обробки запиту в межах одного промпту. Фактично, це «плоска» схема взаємодії, де одна й та сама модель відповідає і за інтерпретацію наміру користувача, і за виклик інструментів, і за формування остаточної відповіді без додаткових шарів валідації чи спеціалізації ролей. Така конфігурація виступає базовою точкою відліку, з якою далі порівнюються усі multi-agent патерни,.

3.3.1. Якість відповіді та безпека single-agent

Агреговані результати для 15 тестових промптів зображено у таблиці 3.1. У ній наведені: розв’язання наміру (IntentResolution), дотримання задачі (TaskAdherence), обґрунтованість (Groundedness), завершення задачі (TaskCompletion), а також показники безпеки й текстової схожості.

Таблиця 3.1

Агреговані результати для 15 тестових промптів

Метрика (Azure AI Foundry)	Середнє значення (%)	Успішних тестів (із 15)	Інтерпретація результату
IntentResolution	7 %	1	Низький рівень розуміння намірів користувача; надто вузьке трактування задачі.
TaskAdherence	33 %	5	Часті відхилення від заданого сценарію або непокриття всіх вимог промпту.
Groundedness	40 %	6	Високий ризик галюцинацій; слабка опора на надану базу знань.
TaskCompletion	7 %	1	Критично низька здатність доводити задачу до фінального, корисного результату.
Violence	100 %	15	Порушень політик щодо насильства не виявлено.
HateAndUnfairness	100 %	15	Мова ворожнечі та упереджені висловлювання відсутні.
TextSimilarity	100 %	15	Висока узгодженість тональності та стилю з еталоном (незалежно від фактичної правильності).

У таблиці 3.1 видно, що IntentResolution досягає лише 7 % (1 із 15 тестів), тобто у більшості випадків агент некоректно інтерпретує запит або обирає надто вузьке формулювання задачі. TaskAdherence трохи вищий — 33 % (5/15), однак це все одно означає, що дві третини відповідей або не покривають усі вимоги промпту, або відхиляються від очікуваного сценарію (наприклад, відповідають загальною фразою замість конкретних дій чи політик). Groundedness становить 40 % (6/15): майже в половині кейсів модель або не використовує релевантні документи, або спирається на власні «hallucinations», що особливо критично для запитів про політики компанії.

Найнижчим показником є TaskCompletion — лише 7 % (1/15). Це свідчить, що у більшості сценаріїв single-agent не доводить задачу до очікуваного результату: не формулює остаточної, чіткої та коректної відповіді, не дає повної інструкції клієнту або не виконує всі підзадачі, описані в промпті.

Натомість усі метрики безпеки показують максимальні значення: Violence — 100 % (15/15), HateAndUnfairness — 100 % (15/15). Це означає, що в жодному з тестів модель не згенерувала контент, який порушує політики щодо насильства чи мови ворожнечі. TextSimilarity також дорівнює 100 % (15/15), що свідчить про високу узгодженість тональності та стилю з еталонними відповідями, навіть попри те, що за змістом багато з них є неповними або помилковими.

3.3.2. Обмеження базового підходу й мотивація multi-agent

Отримані результати показують, що single-agent конфігурація є недостатньо ефективною для складних, багатокрокових запитів клієнтів. Один і той самий агент повинен одночасно вирішувати кілька задач: правильно зрозуміти намір користувача, сформулювати пошуковий запит до бази знань, проаналізувати знайдені документи, звірити відповідь із політиками компанії, дотриматися безпекових обмежень і, зрештою, видати завершене

формулювання для клієнта. На практиці це призводить до того, що модель або поверхово інтерпретує запит, або «економить» на аналізі контексту, зосереджуючись на загальній, але безпечній відповіді.

Низький IntentResolution означає, що агент часто не «вловлює» критичні деталі сценарію: відмінність між запитом про статус повернення коштів та запитом про зміну способу доставки, граничні кейси з порушенням строків, багаточастинні питання тощо. Через це TaskAdherence і TaskCompletion також страждають: навіть якщо в відповіді згадується правильна тема, вона не покриває всі підпункти промпту, не дає чіткої інструкції або не враховує політичні/правові нюанси.

Ще одна важлива проблема — слабкий контроль над політиками й обмеженнями. У single-agent варіанті вся логіка безпеки «зашита» в один system prompt. Це означає, що або промпт робить модель дуже обережною (і вона відмовляється відповідати навіть там, де це дозволено), або, навпаки, недостатньо жорсткою (і тоді з’являється ризик порушення політик). У результаті доводиться шукати компроміс між корисністю й безпекою в межах одного агента, що є погано масштабованим підходом.

Результати оцінювання добре демонструють цей компроміс: безпека формально на максимальному рівні, але за рахунок якості відповіді. Фактично агент обирає найпростіший і найменш ризикований шлях — давати загальні, стислі або надто обережні відповіді, які не закривають реальну потребу користувача. Це і є головна мотивація переходу до multi-agent оркестрацій, де різні аспекти задачі (розпізнавання наміру, пошук по знаннях, перевірка політик, генерація фінальної відповіді) розділяються між спеціалізованими агентами, а централізований оркестратор керує їхньою взаємодією. Такий підхід дозволяє одночасно підвищити TaskCompletion та зберегти високий рівень безпеки, що й буде показано в наступних підрозділах.

3.4. Результати послідовної (sequential) оркестрації

У цьому підрозділі розглядаються результати послідовної оркестрації та їх зіставлення з single-agent конфігурацією. Основний акцент робиться на зміні метрик TaskCompletion та Groundedness, оскільки саме вони найкраще відображають ефект від додавання проміжних етапів перевірки відповіді.

3.4.1. Порівняння single-agent vs sequential за метриками

Щоб наочно продемонструвати вплив послідовної оркестрації, у таблиці 3.2 наведено зведене порівняння метрик для базового single-agent підходу та для sequential-конфігурації. Для обох варіантів використовувався той самий набір із 15 тестових промптів і однакові параметри RAG, тому відмінності пояснюються саме зміною оркестрації.

Таблиця 3.2

Порівняння результатів single-agent та sequential-оркестрації

Конфігурація	IntentResolution	TaskAdherence	Groundedness	Task Completion	Violence	Hate&Unfairness	TextSimilarity
Single-agent	7% (1/15)	33% (5 / 15)	40% (6 / 15)	7% (1 / 15)	100%	100%	100%
Sequential-оркестрація	47% (7 / 15)	73% (11 / 15)	87% (13 / 15)	40% (6 / 15)	100%	100%	100%

Як видно з таблиці 3.2, перехід до sequential-оркестрації дає суттєвий приріст за всіма «якісними» метриками. IntentResolution зростає з 7% до 47%, тобто система набагато частіше правильно ідентифікує намір користувача. TaskAdherence підвищується з 33% до 73%, що означає кращу відповідність відповідей внутрішнім політикам та очікуваній інструкції. Groundedness

демонструє найбільший прогрес: з 40% до 87%, отже, більшість відповідей у sequential-сценарії вже безпосередньо спираються на релевантні документи з бази знань. Найважливіше, що TaskCompletion піднімається з 7% до 40% – кількість повністю розв’язаних користувацьких запитів зростає у кілька разів, при цьому показники безпеки (Violence, Hate&Unfairness, TextSimilarity) залишаються на рівні 100%, тобто впровадження додаткових агентів не погіршує, а фактично консервує початковий рівень безпечної поведінки системи.

3.4.2. Вплив maker–checker ланцюжка та проміжної валідації

Покращення метрик у sequential-оркестрації досягається за рахунок явного розділення ролей між агентами та організації так званого maker–checker ланцюжка. Перший агент у цій конфігурації виступає як «maker»: він аналізує запит, формує проміжну відповідь, витягує контекст із бази знань і пропонує кандидатне рішення. Другий агент грає роль «checker» – перевіряє коректність наміру, узгодженість з політиками, відповідність фактичним даним з RAG-шару та за потреби коригує або відхиляє відповідь. Саме на цьому етапі посилюється Groundedness: перевіряльний агент отримує більш жорстку системну інструкцію орієнтуватися на документи з knowledge base і явно відхиляти відповіді, які виходять за межі наданого контексту. Додатковий ефект дає проміжна валідація наміру: якщо перший агент частково неправильно інтерпретує запит (що часто трапляється у складних сценаріях з уточненнями чи прихованими обмеженнями), другий агент може це виправити до генерації фінальної відповіді. У сукупності це й пояснює приріст IntentResolution і TaskAdherence, а також кратне зростання TaskCompletion: система не просто генерує одну відповідь «з першої спроби», а проходить внутрішній цикл перевірки, який фільтрує небажані та некоректні варіанти ще до повернення їх користувачу.

3.5. Результати паралельної (concurrent) оркестрації

Паралельна оркестрація передбачає, що кілька спеціалізованих агентів одночасно опрацьовують один і той самий запит користувача, після чого фінальний агент-агрегатор обирає або синтезує відповідь. При незмінних налаштуваннях RAG та тій самій моделі gpt-4o така схема дає інший профіль якості, ніж еталонний single-agent та послідовний патерн.

3.5.1. Показники якості паралельного патерна

У відповідному evaluation flow для concurrent оркестрації модель досягла таких інтегральних показників: IntentResolution – 53 % (8/15), TaskAdherence – 73 % (11/15), Groundedness – 100 % (15/15), TaskCompletion – 20 % (3/15), при цьому всі метрики безпеки залишилися на рівні 100 % (Violence, HateAndUnfairness, TextSimilarity). Це означає, що система надзвичайно послідовно спирається на знайдені фрагменти знань і практично не допускає «галюцинацій», але рідше доводить задачі до повного розв’язання з точки зору оцінювача.

У таблиці 3.3 узагальнено результати concurrent-патерна у порівнянні з іншими конфігураціями, на основі яких далі виконується якісний аналіз.

Таблиця 3.3

Порівняння результатів single-agent, sequential-оркестрації та concurrent-оркестрації

Конфігурація	IntentResolution	TaskAdherence	Groundedness	Task Completion	Violence	Hate&Unfairness	TextSimilarity
Single-agent	7% (1/15)	33% (5 / 15)	40% (6 / 15)	7% (1 / 15)	100%	100%	100%
Sequential-оркестрація	47% (7 / 15)	73% (11 / 15)	87% (13 / 15)	40% (6 / 15)	100%	100%	100%
Concurrent-оркестрація	53% (8 / 15)	73% (11 / 15)	100% (15 / 15)	20% (2 / 15)	100%	100%	100%

З наведених значень видно, що головна сильна сторона паралельної оркестрації – максимальна фактична коректність відповіді (100 % Groundedness) у поєднанні з повною відповідністю політикам безпеки. Водночас низький TaskCompletion (лише 3 із 15 промптів визнані повністю розв’язаними) вказує на те, що система часто обмежується частковими поясненнями, відмовами або перенаправленням користувача до інших ресурсів, замість того щоб дати завершену відповідь.

3.5.2. Порівняння sequential vs concurrent: Groundedness vs TaskCompletion

За однакових вхідних умов послідовна оркестрація демонструє інший баланс між продуктивністю та обережністю. Для sequential-патерна TaskCompletion становить 40 % (6/15), а Groundedness – 87 % (13/15), тоді як паралельна конфігурація має лише 20 % TaskCompletion (3/15), але 100 % Groundedness (15/15). Тобто sequential частіше «закриває» задачі, але інколи допускає менш жорстке прив’язування до знайдених документів, тоді як concurrent практично не виходить за межі того, що прямо підтверджується знаннями.

Такий розрив пояснюється насамперед дизайном фінального агента. У паралельній оркестрації агрегатор отримує кілька варіантів відповіді від спеціалізованих агентів і має дуже консервативну системну інструкцію: за найменшої невпевненості він повинен відмовитися, перевірити політики або запропонувати звернутися до підтримки, а не «додумувати» відповідь. У результаті Groundedness досягає 100 %, але TaskCompletion падає, тому що навіть частково коректні, але не ідеально підтверджені відповіді не проходять фінальний фільтр.

У послідовній оркестрації maker–checker ланцюжок працює інакше. Перший агент пропонує чернетку відповіді, другий – перевіряє її на відповідність політикам і фактам та за потреби коригує. Однак його промпт

менш жорстко налаштований на відмову, тому в сумнівних ситуаціях checker радше доповнює або переформульовує відповідь, ніж повністю її блокує. Це дозволяє підвищити TaskCompletion до 40 %, але при цьому залишається невеликий відсоток кейсів, де Groundedness не ідеальний.

Зіставлення цих двох патернів демонструє, що дизайн оркестратора та формулювання системних інструкцій найсильніше впливають саме на TaskCompletion, тобто на готовність системи брати на себе відповідальність за повне розв’язання задачі користувача. Натомість Groundedness значною мірою визначається самою RAG-архітектурою, якістю знань та політиками безпеки. За цих умов паралельна оркестрація може вважатися аргументовано кращою з точки зору фактичності та дотримання політик, тоді як послідовна – більш практичною, коли пріоритетом є вища ймовірність отримати завершену відповідь навіть ціною трохи менш жорсткої консервативності.

3.6. Результати патерна handoff

Патерн handoff поєднує ідею оркестрації з маршрутизацією: замість того щоб усі запити оброблялися одним і тим самим ланцюжком агентів, спеціальний router-агент спочатку класифікує намір користувача, а потім передає його до найвідповіднішого потоку обробки. У реалізації це можуть бути RAG-потоки для типових запитів щодо політик, окремі гілки для складних кейсів, що потребують додаткових уточнень, або безпечні сценарії ескалації до живого оператора. Така архітектура особливо цікава для реальних бізнес-систем, де запити користувачів дуже різнорідні за тематикою, рівнем ризику та очікуваним ступенем автоматизації.

3.6.1. Метрики та якість маршрутизації

Оцінювання патерна handoff на тому ж наборі з 15 тестових промптів показало, що при правильно налаштованому router-агенті система може

досягати майже «ідеальних» показників за ключовими метриками якості. Для цієї конфігурації IntentResolution становить 100 % (15/15), тобто кожен запит був коректно інтерпретований і віднесений до відповідного типу задачі. TaskAdherence також дорівнює 100 % (15/15), що означає точне дотримання вимог користувача в межах обраного потоку. Groundedness досягає 100 % (15/15), як і в паралельній оркестрації, – усі відповіді ґрунтуються на витягнутих фрагментах з knowledge base та не містять очевидних «галюцинацій».

TaskCompletion для handoff-патерна дорівнює 47 % (7/15), що є найкращим результатом серед усіх розглянутих конфігурацій. При цьому метрики безпеки – Violence, HateAndUnfairness та TextSimilarity – залишаються на рівні 100 %, тобто жоден із сценаріїв не порушує політики й не виходить за межі дозволеного контенту. У таблиці 3.4 наведено узагальнені результати для патерна handoff, що дозволяє порівняти його з single-agent, sequential та concurrent підходами за однаковою шкалою.

Таблиця 3.4

Порівняння результатів single-agent, sequential-оркестрації, concurrent-оркестрації та handoff

Конфігурація	IntentResolution	TaskAdherence	Groundedness	Task Completion	Violence	Hate&Unfairness	TextSimilarity
Single-agent	7% (1/15)	33% (5 / 15)	40% (6 / 15)	7% (1 / 15)	100%	100%	100%
Sequential-оркестрація	47% (7 / 15)	73% (11 / 15)	87% (13 / 15)	40% (6 / 15)	100%	100%	100%
Concurrent-оркестрація	53% (8 / 15)	73% (11 / 15)	100% (15 / 15)	20% (2 / 15)	100%	100%	100%
Handoff	100% (15 / 15)	100% (15 / 15)	100% (15 / 15)	47% (7 / 15)	100%	100%	100%

Отже, з точки зору сукупних метрик handoff демонструє найкращий баланс між точністю розуміння наміру, фактичністю відповіді та здатністю доводити задачі до кінця, зберігаючи при цьому максимальний рівень безпеки.

3.6.2. Особливості змішаних потоків запитів і ролі router-агента

Основна причина, чому патерн handoff показує настільки високі результати, полягає в наявності router-агента, який спеціалізується саме на класифікації запитів і виборі відповідного сценарію обробки. У реальних службах підтримки користувачі звертаються з дуже різними питаннями: від простих FAQ на кшталт «як змінити пароль» до неоднозначних кейсів про повернення товару, спірні ситуації з оплатою чи запити, що потенційно порушують політики платформи. Замість того щоб змушувати один універсальний ланцюжок агентів обробляти всі ці випадки, handoff дозволяє «рознести» їх на окремі потоки: безпечні автоматичні відповіді для типових ситуацій, більш обережні RAG-сценарії для складних випадків та ескалацію для ризикових або політично чутливих запитів.

Саме така маршрутизація позитивно впливає на IntentResolution і TaskAdherence. Router-агент отримує підсилений системний промпт із прикладами типових класів запитів, чіткими інструкціями не змішувати домени та додатковими перевірками на відповідність політикам. Завдяки цьому запит майже завжди потрапляє до «правильної» гілки, де решта логіки вже оптимізована під конкретний сценарій, а не намагається бути універсальною для всіх випадків. Це пояснює 100 % IntentResolution і TaskAdherence: система не лише правильно розуміє намір, а й виконує саме той тип дії, який очікується від неї в цьому контексті.

Важливо й те, що handoff не намагається «витиснути максимум» з кожного запиту будь-якою ціною. Якщо router-агент визначає, що кейс виходить за межі того, що можна безпечно й коректно автоматизувати, він передає його до сценарію ескалації або делегує вирішення людині. З точки зору TaskCompletion це рахується як невдале завершення автоматичної задачі, але з погляду реальної системи підтримки такий результат часто є найбільш бажаним: краще ескалувати складний кейс, ніж дати формально завершену, але некоректну або небезпечну відповідь. Саме тому TaskCompletion у handoff-

патерні зростає до 47 %, але не досягає 100 % – частина запитів принципово лишається «людинозалежною».

У підсумку handoff добре масштабується на сценарії з різнорідними доменами та типами задач, де важливо поєднати високу автоматизацію для простих кейсів, максимальну фактичність відповіді для складних запитів та гарантовану безпеку для всіх категорій користувачів. Роль router-агента тут є ключовою: від якості його інструкцій та прикладів залежить, наскільки система зможе повністю реалізувати потенціал цієї архітектури.

3.7. Узагальнений порівняльний аналіз патернів

На цьому етапі всі чотири патерни оркестрації — еталонний single-agent, послідовний (sequential), паралельний (concurrent) та змішаний handoff — вже були протестовані на однаковому наборі з 15 промптів та оцінені тими самими метриками. Тепер важливо подивитися на результати не окремо, а в сукупності: як змінюється баланс між Groundedness і TaskCompletion, що відбувається з IntentResolution та TaskAdherence, і який підхід дає найкращий компроміс для реального бізнес-сценарію на кшталт Taurbull.

3.7.1. Зведена таблиця показників для всіх патернів

Узагальнюючи результати, можна описати кожен патерн через чотири основні метрики якості. Базовий single-agent демонструє дуже низькі показники: IntentResolution 7 % (1/15), TaskAdherence 33 % (5/15), Groundedness 40 % (6/15) та TaskCompletion 7 % (1/15). Це фактично «мінімальний» рівень, який показує, на що здатна проста інструкція без додаткової оркестрації, хоча метрики безпеки та TextSimilarity уже тут залишаються на рівні 100 %.

Послідовна оркестрація (sequential) істотно піднімає всі метрики одночасно: IntentResolution до 47 % (7/15), TaskAdherence до 73 % (11/15),

Groundedness до 87 % (13/15), а TaskCompletion до 40 % (6/15). Найбільший приріст видно саме в Groundedness і TaskCompletion: завдяки maker–checker ланцюжку та проміжній валідації система не лише краще «чіпляється» за базу знань, а й частіше доводить задачі до кінця.

Паралельна оркестрація (concurrent) оптимізована радше під максимальну фактичність, ніж під закінченість задачі. Тут Groundedness досягає 100 % (15/15), TaskAdherence тримається на рівні 73 % (11/15), а IntentResolution — 53 % (8/15), тобто краще, ніж у sequential. Водночас TaskCompletion падає до 20 % (3/15): через консервативний фінальний агент, який обирає найобережнішу відповідь, система часто відмовляється від виконання або делегує запит далі, навіть тоді, коли могла б дати прийнятне автоматичне рішення.

Нарешті, патерн handoff, як показано на рис., показує найкращий загальний баланс: IntentResolution 100 % (15/15), TaskAdherence 100 % (15/15), Groundedness 100 % (15/15) та TaskCompletion 47 % (7/15). Завдяки router-агенту запити майже завжди потрапляють у правильний домен, спеціалізовані гілки забезпечують високу якість і фактичність.

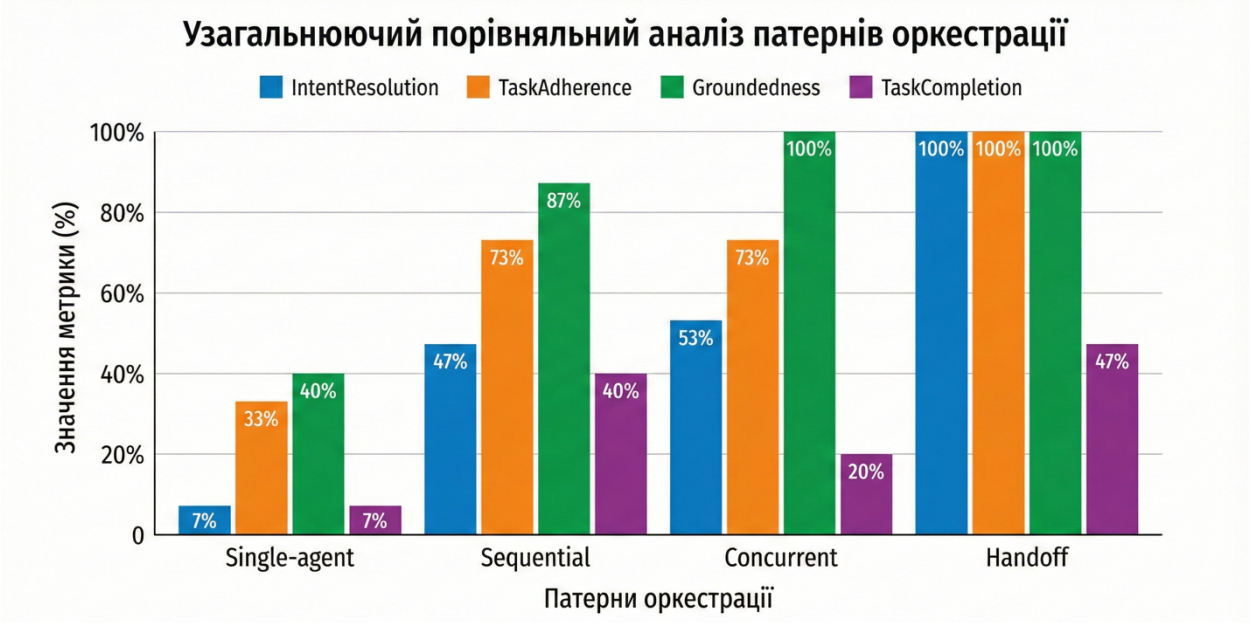

Рис. 3.4. Узагальнююча діаграма порівняння метрик для всіх патернів.

3.7.2. Практичні рекомендації для бізнес-сценаріїв типу Taurbull

З точки зору практичного впровадження в сервісах на кшталт Taurbull, де переважають запити користувачів щодо акаунтів, оплат, підписок, повернень і політик, вибір патерна має ґрунтуватися на змішаному балансі: наскільки складні запити, який рівень ризику для бізнесу, скільки ресурсів готові витратити на оркестрацію. Якщо потік запитів відносно простий і добре вкладається у FAQ-рівень обслуговування, достатньо еталонного single-agent підходу з якісною базою знань і продуманим системним промптом — це мінімальна складність інфраструктури за рахунок прийнятної якості для стандартних сценаріїв.

Якщо до звичайних запитів додаються політики, що мають юридичні наслідки (наприклад, умови повернення коштів, ліміти компенсацій або блокування акаунтів), доцільно перейти до sequential-патерна. У такому сценарії помилка моделі вже не просто «незручна», а може призвести до фінансових втрат або порушення регуляторних вимог, тож простий single-agent підхід стає недостатнім. Maker–checker ланцюжок дозволяє додати етап «перевіряльного» агента, який суворо звіряє відповідь з політиками й knowledge base, фільтрує надто «креативні» формулювання та відкидає твердження, що не мають підтвердження в документах.

Коли для бізнесу критично важлива максимальна фактичність і бажано показувати користувачу кілька точок зору (наприклад, комбінація технічних інструкцій, витягів із договору та коротких пояснень простою мовою), варто розглядати concurrent-патерн. Паралельні агенти можуть по-різному інтерпретувати один і той самий запит: один фокусується на юридичній точності, інший — на зручності для користувача, третій — на внутрішніх бізнес-правилах або обмеженнях. Далі фінальний агрегатор порівнює їхні варіанти, відкидає небезпечні або недостатньо обґрунтовані відповіді й обирає найбільш безпечну та послідовну з точки зору політик компанії. Такий підхід особливо зручний для аналітичних або експертних сервісів, де користувач

очікує не одну коротку репліку, а зважене рішення, підкріплене кількома джерелами. Водночас консервативність фінального агента, який часто відмовляється відповідати при найменших сумнівах, закономірно знижує TaskCompletion.

Якщо ж сервіс, подібний до Taurbull, отримує змішаний потік запитів — від дуже простих до критично важливих, із кількома доменами (білінг, модерація, техпідтримка, юридичні питання) — найбільш гнучким і масштабованим рішенням є патерн handoff. Router-агент, який бачить повний контекст користувацького запиту, спрямовує його до спеціалізованої гілки: прості інформаційні питання можуть одразу йти в легкий single-agent, питання з політиками або ризиками — в sequential, аналітичні або багатокрокові задачі — у concurrent-конфігурацію. Кожна гілка має власні системні промпти, рівень строгості й обмеження, а в найризиковіших ситуаціях маршрутизатор може ескалювати запит до людини-оператора.

3.8. Висновки до розділу 3

У розділі було порівняно чотири патерни оркестрації в Azure AI Foundry за спільних умов (одна модель, одна база знань, однакові RAG-параметри). Single-agent показав межу можливостей «одного великого промпта» та виявився придатним лише для простих FAQ: він частіше плутає намір і гірше дотримується політик. Sequential завдяки ланцюжку maker–checker підвищує Groundedness і TaskCompletion, роблячи систему більш придатною для операційних і політично чутливих сценаріїв.

Concurrent орієнтований на максимальну фактичність: кілька агентів у паралелі дають 100 % Groundedness, але консервативний фінальний агент знижує TaskCompletion. Патерн handoff із router-агентом дає найкращий баланс — повне розпізнавання наміру, дотримання політик і прийнятний рівень автоматизації для змішаного потоку запитів, характерного для кейсу Taurbull.

ВИСНОВКИ

У роботі досягнуто поставленої мети – досліджено й порівняно п’ять архітектурних патернів агентних систем в екосистемі Microsoft Azure на спільному корпусі задач з використанням Retrieval-Augmented Generation та інструментальних дій. На основі аналізу джерел узагальнено теоретичні підходи до побудови агентних систем, ролі RAG-шару, а також рекомендації Azure щодо оркестрації, безпеки та спостережності.

У першому розділі сформовано методологічну базу дослідження: уточнено поняття агентної системи, описано базові компоненти (політика, інструменти, пам’ять, доступ до знань), проаналізовано оркестраційні патерни (single-agent, sequential, concurrent, handoff, planner–executor) та принципи RAG на базі Azure AI Search із гібридним пошуком і методом Reciprocal Rank Fusion. Показано, що якість і стабільність відповідей суттєво залежать від правильно спроєктованого RAG-шару та узгоджених політик безпеки.

У другому та третьому розділах розроблено та реалізовано експериментальну методику оцінювання патернів. Сформовано набір з 15 тестових промптів різної складності (простих довідкових і складних сценаріїв з edge cases та політиками), налаштовано спільні параметри RAG (єдиний knowledge base, гібридний пошук, фіксовані k та фільтри) і використано одну й ту саму модель gpt-4o. Для всіх патернів побудовано evaluation flows в Azure AI Foundry з автоматичними оцінювачами IntentResolution, TaskAdherence, Groundedness, TaskCompletion, TextSimilarity та метриками безпеки. Це забезпечило коректне порівняння конфігурацій за єдиними критеріями.

Базова одноагентна конфігурація (single-agent) показала прийнятну загальну якість і повну відповідність вимогам безпеки, але виявила обмеження щодо розпізнавання наміру, заземленості на джерела й контролю політик. Саме одноагентний патерн став еталонною точкою відліку: його результати продемонстрували, що без структурованого рознесення ролей «пошук →

міркування → перевірка політик» частіше виникають пропуски наміру та фрагментарні відповіді, що мотивувало перехід до multi-agent оркестрацій.

Послідовна (sequential) оркестрація з maker–checker ланцюжком забезпечила покращення TaskCompletion і помітне зростання Groundedness порівняно з single-agent при збереженні 100% показників безпеки. Це досягнуто за рахунок введення спеціалізованих ролей, проміжних артефактів та жорсткішої перевірки відповідей на відповідність політикам. Результати експериментів підтвердили, що стандартизовані проміжні кроки й явний «контролер політик» зменшують кількість помилок і підвищують відтворюваність відповідей, навіть ціною деякого зростання складності й латентності конвеєра.

Паралельна (concurrent) оркестрація продемонструвала максимальну фактичність: Groundedness досягла 100%, проте TaskCompletion виявилась нижчою, ніж у sequential-патерна. Аналіз показав, що вузьким місцем став консервативний фінальний агент-агрегатор, який у разі невпевненості схильний відмовлятися від відповіді або делегувати її назад користувачеві. Це дозволяє зробити висновок, що дизайн промптів і логіки агрегації сильніше впливає на TaskCompletion, тоді як Groundedness переважно визначається якістю RAG та політиками безпеки. Паралельний патерн доцільно застосовувати в сценаріях, де максимальна фактичність важливіша за відсоток завершених відповідей.

Патерн handoff показав високі показники на змішаному потоці запитів: IntentResolution, TaskAdherence та Groundedness досягли 100%, а TaskCompletion перевищила результати інших схем, що також працюють з маршрутизацією. Це підтверджує ефективність виділення окремого router-агента для тріажу різнорідних запитів і передачі їх спеціалізованим виконавцям. Handoff-підхід виявився особливо придатним для сценаріїв типу Taurbull, де одночасно присутні прості FAQ, політики, кейси з діями та нестандартні питання, і потрібен баланс між якістю маршрутизації, стабільністю відповіді та експлуатаційною складністю.

Порівняльний аналіз усіх розглянутих патернів дозволив сформулювати практичні рекомендації. Для простих довідкових сценаріїв з чутливістю до вартості та латентності доцільно застосовувати single-agent конфігурацію як найпростішу й найдешевшу. Для задач, де критичні перевірка політик і регламентів, оптимальним є послідовний maker–checker конвеєр. При вимозі максимальної фактичності та бажанні отримати кілька точок зору виправданий паралельний патерн з консервативним агрегатором. Для змішаних потоків запитів, характерних для реальних бізнес-систем, найкраще себе зарекомендував handoff з доменною маршрутизацією. Ієрархічні схеми planner–executor на основі отриманих результатів розглядаються як перспективний напрямок для подальшого розширення в задачах з довгими ланцюжками дій та інструментів.

Практичну цінність роботи становлять не лише самі виміряні показники, а й запропонована методика порівняння агентних патернів в Azure AI Foundry: уніфіковані RAG-налаштування, стандартний набір метрик, використання evaluation flows та підхід до класифікації сценаріїв. Отримані конфігурації, результати й рекомендації можуть бути безпосередньо використані або адаптовані для проєктування прикладних агентних рішень у корпоративних середовищах на базі Microsoft Azure, а також слугувати основою для подальших експериментів з розширенням корпусу задач, метрик (зокрема за латентністю та вартістю) і типів оркестрацій.

ПЕРЕЛІК ПОСИЛАНЬ

1. AI Agent Orchestration Patterns — Azure Architecture Center [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/ai-agent-design-patterns.
2. Azure AI Foundry documentation (включно з Agent Service) [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/ai-foundry/.
3. Hybrid search using vectors and full text in Azure AI Search (overview) [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/search/hybrid-search-overview.
4. Prompt flow in Azure AI Foundry portal (Evaluation flow) [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/ai-foundry/concepts/prompt-flow.
5. Hybrid search scoring (Reciprocal Rank Fusion, RRF) — Azure AI Search [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/search/hybrid-search-ranking.
6. Azure AI Evaluation client library for Python (SDK overview) [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/python/api/overview/azure/ai-evaluation-readme?view=azure-python.
7. Local evaluation with the Azure AI Evaluation SDK (how-to) [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/ai-foundry/how-to/develop/evaluate-sdk.
8. azure.ai.evaluation package reference [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/python/api/azure-ai-evaluation/azure.ai.evaluation?view=azure-python.
9. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework [Електронний ресурс] / Microsoft Research. — Режим доступу: https://www.microsoft.com/en-us/research/publication/autogen-enabling-next-gen-llm-applications-via-multi-agent-conversation-framework/.

10. Semantic Kernel documentation (вступ; огляд фреймворку) [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/semantic-kernel/.
11. Durable Functions overview — Azure Functions [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview.
12. Create a hybrid query in Azure AI Search (how-to; RRF merge) [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/search/hybrid-search-how-to-query.
13. Prompt flow — documentation site [Електронний ресурс] / Microsoft Open Source, Microsoft GitHub. — Режим доступу: https://microsoft.github.io/promptflow/.
14. Azure AI Foundry Agent Service REST API reference [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/rest/api/aifoundry/aiagents/.
15. AutoGen: Group Chat design pattern (stable docs) [Електронний ресурс] / Microsoft Open Source, Microsoft GitHub. — Режим доступу: https://microsoft.github.io/autogen/stable//user-guide/core-user-guide/design-patterns/group-chat.html.
16. AI Architecture Design — Azure Architecture Center (огляд; agent-based architecture) [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/architecture/ai-ml/.
17. Azure AI Foundry Agent Service — overview and FAQ [Електронний ресурс] / Microsoft Azure. — Режим доступу: https://azure.microsoft.com/en-us/products/ai-foundry/agent-service.
18. Azure Durable Functions documentation hub [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/azure-functions/durable/.

19. Semantic Kernel Agent Framework (орієнтир по агентам у SK) [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/semantic-kernel/frameworks/agent/.
20. AutoGen 0.2 Group Chat notebook/doc [Електронний ресурс] / Microsoft Open Source, Microsoft GitHub. — Режим доступу: https://microsoft.github.io/autogen/0.2/docs/notebooks/agentchat_groupchat/.
21. Sen S. Intelligent Agents with Semantic Kernel: A Comprehensive Guide (оглядовий гайд) [Електронний ресурс] / Saptak Sen. — 2025. — Режим доступу: https://saptak.in/writing/2025/03/22/building-intelligent-agents-with-semantic-kernel.
22. Azure OpenAI Service documentation. Огляд та довідка по сервісу Azure OpenAI [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/developer/ai/resources-overview?pivots=dotnet.
23. Azure AI Search – feature descriptions. Документація за можливостями Azure AI Search [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/azure/search/search-what-is-azure-search.
24. Azure AI Foundry documentation. Централізована документація з розробки генерувальних AI-застосунків та агентів в Azure [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/azure/ai-studio/.
25. Build a RAG solution using Azure AI Search. Навчальний посібник з побудови RAG-рішень на базі Azure AI Search та Azure OpenAI [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/azure/search/tutorial-rag-build-solution.
26. How to Get Started with AI Agent Development on Azure? Аналітична стаття про використання Azure AI Agent Service для побудови LLM-агентів [Електронний ресурс] / Solulab. — Режим доступу: https://www.solulab.com/how-to-get-started-with-ai-agent-development-on-azure/.

27. get-started-with-ai-agents. Прикладовий репозиторій з базовими сценаріями роботи з Azure AI Agent Service та інтеграцією з іншими сервісами Azure [Електронний ресурс] / GitHub. — Режим доступу: https://www.solulab.com/how-to-get-started-with-ai-agent-development-on-azure/.
28. Guo X., Chen Y., Liang H. та ін. Large Language Model based Multi-Agents: A Survey of Progress and Challenges // arXiv preprint arXiv:2402.01680 [Електронний ресурс]. — 2024. — Режим доступу: https://arxiv.org/abs/2402.01680.
29. Li G., Hammoud H. A. A. K., Itani H. та ін. CAMEL: Communicative Agents for “Mind” Exploration of Large Scale Language Model Society // arXiv preprint arXiv:2303.17760 [Електронний ресурс]. — 2023. — Режим доступу: https://arxiv.org/abs/2303.17760.
30. Introduction to Azure Storage — Azure [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction.
31. What is Azure Monitor? — full-stack monitoring in Azure [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/azure-monitor/overview.
32. What is Azure Cost Management and Billing? — cost analysis and optimization in Azure [Електронний ресурс] / Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/azure/cost-management-billing/cost-management-billing-overview.
33. Holistic Evaluation of Language Models (HELM). Офіційний сайт та документація до платформи комплексного оцінювання LLM [Електронний ресурс] / Stanford Center for Research on Foundation Models. — Режим доступу: https://crfm.stanford.edu/helm/latest/.
34. Evaluate generative AI applications with prompt flow and Azure AI evaluation. Документація та рекомендації щодо побудови пайплайнів оцінювання генерувальних AI-рішень в Azure [Електронний ресурс] / Microsoft Learn.

— Режим доступу: https://learn.microsoft.com/en-us/azure/ai-studio/how-to/evaluate-generative-ai-app.

35. Li D. та ін. From Generation to Judgment: Opportunities and Challenges of LLM-as-a-judge. Узагальнювальне дослідження підходів до використання LLM як автоматизованих оцінювачів якості [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/2306.04599.