Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Комп'ютерного моделювання та інтелектуальних технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розробка методу інтелектуальної маршрутизації мікросервісів на основі великих мовних моделей для інформаційної системи трекінгу замовлень
(тема)

Виконав:
здобувач II року навчання,
групи СПРм-24-1
Назар БЕЗУГЛИЙ
(власне ім'я, прізвище)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системне проектування
(повна назва освітньої програми)
Керівник доц. каф. КМІТ Зульфія ІМАНГУЛОВА
(посада, власне ім'я, прізвище)

Допускається до захисту
Завідувач кафедри КМІТ _____ Ігор ГРЕБЕННІК
(підпис) (власне ім'я, прізвище)

2026 р.

Я як здобувач вищої освіти ХНУРЕ розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

11.05.2026  Назар БЕЗУГЛИЙ

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено 11.05.2026 р.

Керівник кваліфікаційної роботи  Зульфія ІМАНГУЛОВА

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
Кафедра Комп'ютерного моделювання та інтелектуальних технологій
Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
(код і повна назва)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системне проектування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри

_____
(підпис)
« 11 » 05 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Безуглому Назару Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Розробка методу інтелектуальної маршрутизації мікросервісів на основі великих мовних моделей для інформаційної системи трекінгу замовлень
затверджена наказом по університету від « 06 » 04 2026р. № 268СТ
2. Термін подання здобувачем роботи до екзаменаційної комісії 13.05.2026 р.
3. Вихідні дані до роботи Наукові праці та публікації з проблематики мікросервісної архітектури, API Gateway, оркестрації сервісних викликів, обробки природної мови та використання великих мовних моделей в інформаційних системах. Аналоги сучасних систем трекінгу замовлень та інтелектуальних сервісних шлюзів. Перелік використовуваних програмних засобів: .NET 8, C#, ASP.NET Core Web API, Redis, Docker, JSON Schema, Visual Studio 2022, Visual Studio Code, Swagger, draw.io, Lucidchart.
4. Перелік питань, що потрібно опрацювати в роботі Вступ. Аналіз предметної області інформаційних систем трекінгу замовлень та постановка задачі дослідження. Проектування методу інтелектуальної маршрутизації мікросервісів. Інструменти та технологічна база реалізації інтелектуального шлюзу. Реалізація прототипу та експериментальна оцінка методу. Архітектура прототипу та реалізовані компоненти.
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій Схема патерну API Gateway. Концептуальна схема інтелектуального шлюзу. Схема семантичного мапування. Потік даних обробки запиту в інтелектуальному шлюзі. Діаграма послідовності обробки запиту в детермінованому режимі. Діаграма послідовності обробки запиту в інтелектуальному режимі. Діаграма керованого процесу оркестрації.

Архітектура прототипу інтелектуального шлюзу. Графік розподілу сценаріїв перевірки за типом завершення. Графік кількості ітерацій і викликів інструментів у базових сценаріях перевірки.

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Розділи спеціальної частини	*доц. Імангулова З.І.*		11.05.2026

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів проекту (роботи)	Примітка
1	Отримання завдання на кваліфікаційну роботу та пошук літературних джерел	06.04.2026 – 07.04.26	
2	Аналіз предметної області, огляд існуючих API Gateways та формування вимог до системи	07.04.26 – 09.04.2026	
3	Огляд існуючих підходів до маршрутизації, оркестрації запитів та використання великих мовних моделей у мікросервісних системах	09.04.2026 – 11.04.2026	
4	Постановка задачі дослідження	11.04.2026 – 13.04.2026	
5	Розробка архітектури інтелектуального шлюзу та проектування взаємодії з мікросервісами	13.04.2026 – 17.04.2026	
6	Програмна реалізація MVP-прототипу інтелектуального API Gateway на C# та .NET	17.04.2026 – 28.04.2026	
7	Тестування прототипу, перевірка типових сценаріїв та аналіз результатів роботи системи	28.04.2026 – 03.05.2026	
8	Оформлення пояснювальної записки, підготовка графічного матеріалу та формування висновків	03.05.2026 – 09.05.2026	
9	Подання кваліфікаційної роботи на кафедру, проходження попереднього захисту та підготовка до захисту	09.05.2026 – 11.05.2026	

Дата видачі завдання 06 04 2026 р.

Здобувач _____ Назар БЕЗУГЛИЙ
(підпис) (власне ім'я, прізвище)

Керівник роботи _____ доц. Зульфія ІМАНГУЛОВА
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 120 с., 6 таблиць, 10 рис., 2 додатки, 44 джерела.

МІКРОСЕРВІСНА АРХІТЕКТУРА, API GATEWAY, ВЕЛИКІ МОВНІ МОДЕЛІ, ІНТЕЛЕКТУАЛЬНА МАРШРУТИЗАЦІЯ, ОРКЕСТРАЦІЯ ЗАПИТІВ, ТРЕКІНГ ЗАМОВЛЕНЬ, TOOL CALLING

Об’єкт дослідження — процес маршрутизації запитів у мікросервісних інформаційних системах.

Предмет дослідження — методи та засоби інтелектуальної маршрутизації мікросервісів на основі великих мовних моделей у системі трекінгу замовлень.

Мета роботи — розробка методу інтелектуальної маршрутизації, який забезпечує перетворення природномовного запиту користувача у структурований і контрольований план сервісних викликів.

У роботі використано методи системного аналізу, архітектурного проєктування, обробки природної мови, програмної реалізації та експериментальної перевірки.

Наукова новизна полягає в удосконаленні підходу до маршрутизації запитів у мікросервісному середовищі шляхом поєднання великих мовних моделей із формальними механізмами контролю виконання.

Результати роботи можуть бути використані під час створення інтелектуальних інтерфейсів для систем трекінгу замовлень та інших мікросервісних інформаційних систем.

Галузь застосування — системи електронної комерції, логістики та корпоративні сервісні платформи.

ABSTRACT

Explanatory note of the qualification work: 120 p., 6 tables, 10 pic., 2 appendices, 44 sources of information.

MICROSERVICE ARCHITECTURE, API GATEWAY, LARGE LANGUAGE MODELS, INTELLIGENT ROUTING, REQUEST ORCHESTRATION, ORDER TRACKING, TOOL CALLING

The object of research is the request routing process in microservice information systems.

The subject of research is methods and tools for intelligent microservice routing based on large language models in an order tracking system.

The purpose of the work is to develop an intelligent routing method that transforms a user's natural language request into a structured and controllable plan of service calls.

The research methods include system analysis, architectural design, natural language processing, software implementation, and experimental evaluation.

The scientific novelty lies in improving request routing in a microservice environment by combining large language models with formal execution control mechanisms.

The results can be used in the development of intelligent interfaces for order tracking systems and other microservice-based information systems.

Application areas include e-commerce, logistics, and enterprise service platforms.

Further research may focus on expanding the toolset, improving security, and enhancing routing quality evaluation.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

.NET — програмна платформа для розроблення застосунків;

API (Application Programming Interface) — інтерфейс програмування застосунків;

API Gateway — програмний шлюз, що виступає єдиною точкою входу для обробки, маршрутизації та оркестрації запитів до мікросервісів;

CI/CD (Continuous Integration / Continuous Delivery) — безперервна інтеграція та безперервне постачання програмного забезпечення;

gRPC (Google Remote Procedure Call) — технологія віддаленого виклику процедур для взаємодії між сервісами;

HTTP (HyperText Transfer Protocol) — протокол передавання гіпертексту;

JSON (JavaScript Object Notation) — текстовий формат обміну даними;

JSON Schema — мова опису структури та правил валідації JSON-документів;

LLM (Large Language Model) — велика мовна модель;

MVP (Minimum Viable Product) — мінімально життєздатний програмний продукт із базовим набором функцій;

NLP (Natural Language Processing) — обробка природної мови;

REST (Representational State Transfer) — архітектурний стиль побудови розподілених інформаційних систем і вебсервісів;

ІС — інформаційна система;

ПЗ — програмне забезпечення.

ВСТУП

Сучасний розвиток інформаційних систем характеризується переходом від монолітних рішень до мікросервісної архітектури. Такий підхід забезпечує гнучкість розроблення, масштабованість, незалежне оновлення компонентів і кращу адаптацію до змін бізнес-вимог. Особливо це важливо для систем трекінгу замовлень, у яких необхідно обробляти значну кількість подій, забезпечувати взаємодію між кількома сервісами та надавати користувачеві актуальну інформацію про стан замовлення в реальному часі.

Разом із перевагами мікросервісної архітектури виникають і нові проблеми. Один користувацький запит у таких системах часто потребує звернення до кількох сервісів, а логіка його обробки виходить за межі простої статичної маршрутизації. Традиційні рішення класу API Gateway є ефективними, коли маршрути та сценарії обробки визначені наперед, проте виявляються недостатньо гнучкими для роботи з природномовними, неповними або контекстно залежними запитами.

Водночас значний розвиток отримали великі мовні моделі, які здатні аналізувати природномовні звернення, визначати намір користувача, формувати структуровані дані та підтримувати багатоетапну взаємодію. Це відкриває можливість використання таких моделей як інтелектуальних компонентів для маршрутизації запитів і вибору необхідних сервісних викликів.

Отже, актуальність роботи зумовлена потребою поєднання переваг мікросервісної архітектури з можливостями великих мовних моделей для побудови контрольованого механізму обробки природномовних запитів у системі трекінгу замовлень. Такий підхід дає змогу зробити взаємодію з користувачем більш природною, зберігаючи при цьому інженерну керованість, валідацію параметрів, політики доступу та контроль виконання.

Метою дослідження є розробка методу інтелектуальної маршрутизації запитів у мікросервісному середовищі для інформаційної системи трекінгу замовлень, який забезпечує перетворення природномовного запиту користувача у структурований, перевірюваний і контрольований план сервісних викликів.

Для досягнення поставленої мети в роботі необхідно вирішити такі завдання:

– проаналізувати особливості функціонування інформаційних систем трекінгу замовлень і вимоги до обробки користувацьких запитів у мікросервісному середовищі;

– дослідити можливості та обмеження традиційних засобів маршрутизації запитів, зокрема API Gateway, у сценаріях роботи з природномовними зверненнями;

– проаналізувати підходи до семантичної інтерпретації запитів, багатоетапної оркестрації та використання зовнішніх інструментів;

– розробити метод інтелектуальної маршрутизації запитів, який поєднує семантичну інтерпретацію звернення з формальними механізмами валідації, політиками доступу та керуванням контекстом;

– спроектувати архітектуру інтелектуального шлюзу для системи трекінгу замовлень;

– реалізувати програмний прототип запропонованого підходу;

– провести перевірку працездатності розробленого методу на типових сценаріях взаємодії.

Об’єктом дослідження є процес маршрутизації та координації запитів у мікросервісних інформаційних системах.

Предметом дослідження є методи й програмні засоби інтелектуальної маршрутизації запитів у мікросервісному середовищі для системи трекінгу замовлень.

У роботі використано методи системного аналізу, архітектурного проєктування, моделювання процесів обробки запитів, підходи до обробки природної мови, а також методи програмної реалізації й експериментальної перевірки.

Наукова новизна одержаних результатів полягає у запропонованому методі інтелектуальної маршрутизації запитів у мікросервісному середовищі, який поєднує семантичну інтерпретацію природномовного звернення з формальними механізмами вибору інструментів, валідації параметрів, політик доступу та

контролю виконання. Також удосконалено підхід до обробки звернень у системі трекінгу замовлень за рахунок використання керованої багатоетапної оркестрації, підтримки уточнення неповних даних і збереження сесійного контексту для повторного використання проміжних результатів.

Результати цієї роботи були доповідані та обговорені на XXX Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» (Харків, 2026) [1] та на II Міжнародній науково-практичній конференції «СУЧАСНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ MIT&AIS-2026» (Харків-Яремче, 2026) [2].

Практичне значення одержаних результатів полягає в тому, що запропонований метод і реалізований програмний прототип можуть бути використані як основа для побудови інтелектуальних інтерфейсів доступу до систем трекінгу замовлень та інших мікросервісних інформаційних систем. Запропоновані рішення забезпечують можливість керованої обробки природномовних запитів, використання багатоетапної оркестрації, валідації параметрів, політик доступу та сесійного контексту під час виконання сервісних дій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Характеристика та особливості функціонування інформаційних систем трекінгу замовлень

В умовах стрімкого розвитку електронної комерції та глобалізації логістичних ланцюгів інформаційні системи трекінгу замовлень стали критично важливим компонентом сучасного бізнесу. Вони забезпечують прозорість доставки, підвищують довіру клієнтів і допомагають підприємствам оптимізувати операційні витрати [3]. Основна функція такої системи полягає у наданні актуальної інформації про місцезнаходження та статус замовлення протягом усього життєвого циклу — від оформлення до вручення кінцевому споживачу. На практиці сучасна система трекінгу є складною екосистемою, що інтегрує дані з багатьох джерел, зокрема зі складських і транспортних підсистем, а також із зовнішніх сервісів логістичних провайдерів [4].

Функціонування систем трекінгу визначається кількома ключовими вимогами. По-перше, це обробка подій майже в реальному часі, оскільки система має приймати й узгоджувати велику кількість повідомлень про зміни стану доставки, перетин контрольних зон або уточнення прогнозного часу прибуття з мінімальною затримкою. По-друге, важливою є інтероперабельність, адже система взаємодіє з різнорідними API кур’єрських служб і поштових операторів, які можуть використовувати різні формати даних та відмінні протоколи інтеграції [5]. По-третє, критичними є висока доступність і масштабованість, оскільки доступ до даних про замовлення здійснюється через клієнтські застосунки та може різко зростати в пікові періоди, наприклад під час сезонних розпродажів. Додатково суттєвою стає багатоканальність взаємодії з користувачем, коли оновлення та запити мають підтримуватися через веб-інтерфейс, мобільний застосунок, повідомлення та канали автоматизованої комунікації.

Типова система трекінгу включає модулі збору й агрегації даних із логістичного ланцюга, модуль обробки статусів, який перетворює технічні події та коди в зрозумілі користувачу стани доставки, а також модуль прогнозування орієнтовного часу прибуття на основі історичних даних і поточної ситуації [6]. Таким чином, система виступає інтеграційним вузлом, що об’єднує різні підсистеми та забезпечує цілісний погляд на процес доставки в термінах, зрозумілих кінцевому користувачу.

На сучасному етапі розвитку систем трекінгу зростає значущість не лише відображення статусів, а й активної взаємодії з користувачем. Клієнти очікують можливості ставити запитання щодо причин затримки, отримувати пояснення змін у маршруті чи часових оцінках, а в окремих випадках — ініціювати зміни умов доставки під час транспортування [7]. Це створює підвищені вимоги до інтерфейсів взаємодії, оскільки жорстко задані сценарії у вигляді меню або статичних форм не покривають усього різноманіття запитів, а також ускладнюють підтримку й розвиток системи при появі нових бізнес-правил та інтеграцій.

Отже, сучасна інформаційна система трекінгу замовлень є не лише інструментом моніторингу, а складним інтеграційним середовищем, ефективність якого залежить від узгодженої взаємодії мікросервісів і здатності швидко адаптувати обробку запитів до контексту користувача. Саме ця потреба в гнучкій інтерпретації звернень і керованому виконанні дій у межах доступних операцій створює підґрунтя для застосування інтелектуальних підходів до маршрутизації запитів у мікросервісній архітектурі.

1.2 Аналіз мікросервісної архітектури та ролі шлюзів API (API Gateway) у розподілених системах

Сучасний підхід до розробки складних інформаційних систем, зокрема систем трекінгу замовлень, дедалі частіше базується на мікросервісній архітектурі. На відміну від монолітного підходу, де всі функціональні компоненти реалізовані в межах одного застосунку, мікросервісна архітектура передбачає поділ системи на

набір слабкопов’язаних сервісів, що взаємодіють між собою через мережеві протоколи [8]. Кожен мікросервіс реалізує окрему бізнес-здатність і може розгортатися, масштабуватися та оновлюватися незалежно від інших компонентів. Така незалежність зменшує зв’язаність змін і в типових випадках дає змогу скоротити час виведення змін у продуктивне середовище та підвищити стійкість системи до відмов окремих компонентів [9].

Разом із тим, перехід до розподіленої архітектури ускладнює взаємодію клієнтських застосунків із серверною частиною. Якщо клієнт звертається безпосередньо до великої кількості дрібних сервісів, зростає кількість мережевих запитів і ускладнюється формування узгодженої відповіді, оскільки дані для одного екрану або одного користувацького сценарію можуть бути розподілені між кількома сервісами. Додатково виникають проблеми централізованого контролю доступу та наскрізної безпеки, а також потреба підтримувати різні протоколи й формати взаємодії на стороні клієнта, що ускладнює розвиток інтерфейсів і підвищує вартість супроводу.

Для розв’язання цих проблем у мікросервісних системах широко застосовується патерн API Gateway [10]. Шлюз виступає єдиною точкою входу для зовнішніх запитів і координує доступ до внутрішніх сервісів, що ілюструється на рисунку 1.1. З практичної точки зору API Gateway виконує роль маршрутизатора, який перенаправляє запити до відповідних мікросервісів за визначеними правилами, зберігаючи можливість змінювати внутрішню структуру системи без необхідності змін у клієнтських застосунках. Крім того, шлюз дозволяє агрегувати дані, коли для формування однієї відповіді потрібні відомості з кількох сервісів; у такому разі він виконує кілька внутрішніх звернень і об’єднує результат, зменшуючи кількість мережевих взаємодій між клієнтом і сервером.

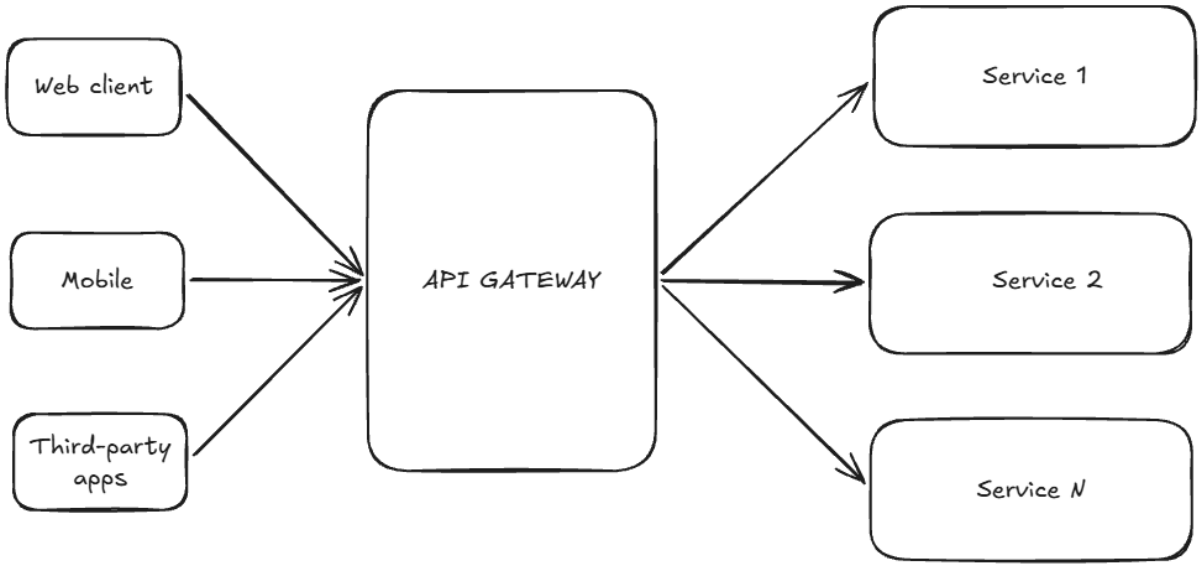

Рисунок 1.1 – Схема патерну API Gateway

API Gateway також є природним місцем для централізованої реалізації безпеки, оскільки саме на цьому рівні може виконуватися перевірка автентифікаційних токенів і застосовуватися єдині політики доступу для зовнішніх запитів. Окремим напрямом розвитку цього підходу є спеціалізація шлюзу під різні типи клієнтів, коли для веб-інтерфейсу та мобільного застосунку формуються адаптовані контракти взаємодії, що дозволяє оптимізувати структуру відповідей і зменшити надлишкові дані в мережі [11]. Додатково шлюз може виконувати перетворення протоколів і форматів, наприклад приймати зовнішні HTTP-запити та взаємодіяти з внутрішніми сервісами через інші протоколи або стандартизовані інтерфейси, що відповідає практиці побудови ефективних інтеграцій у розподілених системах [12].

У контексті систем трекінгу замовлень API Gateway є не лише технічним проксі, а рівнем, на якому природно зосереджуються рішення щодо оркестрації, узгодження даних та контролю доступу. Саме це робить шлюз придатним для подальшого розширення інтелектуальними механізмами обробки запитів: коли користувацьке звернення подано природною мовою або має сценарний характер, виникає потреба не лише перенаправити запит за статичним маршрутом, а інтерпретувати намір, вибрати допустиму операцію та керовано виконати послідовність дій у межах доступних інструментів і політик. Це підводить до

необхідності методів інтелектуальної маршрутизації мікросервісів, які розглядаються в наступних підрозділах.

1.3 Критичний аналіз існуючих методів та засобів маршрутизації мікросервісів

На сучасному ринку програмних засобів для організації API Gateway представлено широкий спектр рішень, які відрізняються продуктивністю, гнучкістю налаштувань та інтеграційними можливостями. Попри різницю у реалізаціях, базова логіка маршрутизації в більшості таких систем залишається статичною або декларативною: рішення про перенаправлення запиту приймається за заздалегідь визначеними правилами, що зіставляють характеристики вхідного HTTP-запиту, зокрема шлях, метод і заголовки, з конфігурацією шлюзу [13]. Такий підхід добре працює для технічної маршрутизації, однак суттєво обмежує можливості системи у випадках, коли запит користувача є контекстно залежним або вимагає виконання сценарію, що виходить за межі наперед визначених маршрутів.

В екосистемі .NET одним із поширених рішень є бібліотека Ocelot, яка вбудовується в конвеєр ASP.NET Core як набір проміжних компонентів. Маршрутизація в Ocelot задається конфігураційно через явну відповідність між зовнішніми адресами та внутрішніми адресами мікросервісів. Хоча Ocelot підтримує типові можливості шлюзу, зокрема агрегацію запитів, автентифікацію та інтеграцію з сервісами реєстрації, його принципова властивість полягає в детермінованості маршрутизації: додавання нового сценарію взаємодії або зміна логіки вимагає ручного оновлення конфігурації. У системах трекінгу, де запити користувача можуть бути варіативними та залежати від поточного стану доставки, це призводить до зростання складності підтримки конфігурацій і обмежує гнучкість обробки запитів, які не вкладаються в наперед визначені правила [14].

Хмарні платформи, зокрема Azure API Management, пропонують комплексні засоби керування життєвим циклом API, включаючи публікацію, контроль доступу

та застосування політик обробки запитів [15]. У таких рішеннях логіка часто реалізується через декларативні правила, які виконуються на різних етапах обробки запиту [16]. Це дає змогу будувати складні ланцюжки трансформацій і перевірок, однак узгодженість сценарію все одно залежить від того, наскільки повно розробник передбачив можливі гілки поведінки наперед. У випадках, коли запити користувача є варіативними, а правильна відповідь залежить від контексту та поточного стану процесу доставки, такий підхід призводить до зростання складності конфігурацій і не забезпечує природної адаптації до наміру користувача без додаткового програмування.

Більш масштабним рішенням є Kong API Gateway, який орієнтований на високу продуктивність і розширюваність за рахунок плагінної архітектури. Такі шлюзи ефективно забезпечують інфраструктурні функції — безпеку, журналювання, трансформацію трафіку — та дозволяють централізовано керувати правилами через адміністративні інтерфейси. Водночас маршрутизація в подібних рішеннях, як правило, залишається технічною і будується на зіставленні шляхів, регулярних шаблонів і заголовків. Це означає, що підтримка семантичної інтерпретації змісту запиту або контекстної логіки на рівні бізнес-сценаріїв потребує додаткових розширень і не є базовою можливістю шлюзу [17].

На рівні інфраструктури розподілених середовищ широко застосовуються рішення класу Ingress-контролерів, зокрема на базі Nginx. Вони забезпечують надійну маршрутизацію на рівні прикладного протоколу, орієнтуючись переважно на доменні імена, шляхи та інші технічні атрибути запиту. Такі компоненти є ефективними для побудови мережевого контуру доступу до сервісів, однак їхній рівень абстракції є недостатнім для інтерпретації бізнес-контексту або складних намірів користувача. З цієї причини використання Ingress-підходів не вирішує задачі семантичної маршрутизації і потребує додаткового прикладного прошарку, якщо необхідно обробляти сценарні або природномовні звернення.

Для узагальнення результатів аналізу доцільно порівняти найбільш поширені підходи та рішення класу API Gateway з позиції їх придатності до роботи з природномовними запитами, багатоетапними сценаріями та контрольованим

виконанням сервісних дій (таблиця 1.1). Таке порівняння дозволяє наочно визначити обмеження існуючих підходів і обґрунтувати доцільність розробки інтелектуального шлюзу.

Таблиця 1.1 – Порівняння існуючих підходів до маршрутизації з інтелектуальним шлюзом

Підхід	Тип маршрутизації	Підтримка природно-мовних запитів	Підтримка багатоетапних сценаріїв	Уточнення неповних параметрів	Політики доступу до інструментів	Формальна валідація параметрів
Ocelot	конфігураційна, детермінована	–	обмежена	–	частково	частково
Azure API Management	декларативна, політико-орієнтована	–	обмежена	–	+	частково
Kong API Gateway	конфігураційна, детермінована	–	обмежена	–	+	частково
Ingress / Nginx	мережева, маршрутна	–	–	–	обмежено	–
Запропонований інтелектуальний шлюз	гібридна	+	+	+	+	+

Наведене порівняння показує, що існуючі рішення класу API Gateway ефективно реалізують інфраструктурні функції маршрутизації, безпеки та централізованого контролю, однак не забезпечують повноцінної підтримки природномовних запитів, керованої багатоетапної оркестрації та уточнювальної взаємодії з користувачем. Саме це зумовлює доцільність розробки інтелектуального шлюзу, який поєднує технічну надійність класичних gateway із семантичними механізмами інтерпретації запитів і формальними обмеженнями виконання.

1.4 Дослідження проблем обробки неструктурованих запитів та складної оркестрації сервісів

Процес взаємодії користувача з сучасними системами трекінгу замовлень поступово зміщується від жорстко визначених графічних інтерфейсів у бік природніших форм комунікації. Водночас на технічному рівні виникає

фундаментальна перешкода — семантичний розрив між неструктурованою природною мовою користувача та строго формалізованими прикладними інтерфейсами [18]. Мікросервісні архітектури, зокрема у поєднанні з API Gateway, переважно орієнтовані на виконання атомарних, чітко визначених операцій із заданими параметрами, що ускладнює реалізацію складних сценаріїв, ініційованих вільним текстом.

Перший клас проблем пов’язаний з інтерпретацією наміру користувача та виділенням сутностей із неструктурованого запиту. У системах трекінгу звернення на кшталт «Перевір, де моє вчорашнє замовлення, і якщо воно ще на складі — зміни адресу доставки» одночасно містить кілька дій, умову виконання та параметри, які потрібно пов’язати з конкретними операціями прикладного інтерфейсу. Традиційні шлюзи не мають механізмів такого розбору, оскільки їхня логіка маршрутизації спирається на технічні атрибути HTTP-запиту та структуру параметрів, які вже сформовані клієнтом. У результаті клієнтський застосунок вимушено бере на себе роль оркестратора: послідовно викликає кілька сервісів, фільтрує дані, перевіряє статус і лише після цього ініціює зміну атрибутів доставки. Це призводить до перенесення частини бізнес-логіки на клієнтський рівень і ускладнює підтримку, тестування та розвиток системи, оскільки логіка сценарію розпорошується між інтерфейсом і серверними компонентами.

Другий клас проблем стосується оркестрації сервісів у динамічних сценаріях. У мікросервісному середовищі виконання комплексного завдання часто потребує послідовного або паралельного виклику декількох незалежних компонентів, а також умовних переходів залежно від проміжних результатів. Традиційні підходи до оркестрації базуються на заздалегідь визначених робочих процесах, описаних у коді або в спеціалізованих засобах керування процесами. Такі рішення є ефективними для типових сценаріїв, однак залишаються статичними: вони працюють добре лише в межах тих гілок поведінки, які були передбачені під час проектування. Коли запит користувача має варіативну структуру, або коли з’являються нові правила предметної області, виникає потреба змінювати логіку

оркестрації та підтримувати дедалі більшу кількість комбінацій взаємодій між сервісами, що ускладнює розвиток системи і збільшує витрати на супровід.

Третій клас проблем пов’язаний з роботою з неоднозначністю та неповнотою інформації у природномовних запитах. У реальних ситуаціях користувач часто не вказує всі параметри, необхідні для виконання операції, або покладається на контекст попередніх повідомлень. Наприклад, запит «Скасуй доставку» без уточнення ідентифікатора замовлення не може бути оброблений як стандартний виклик API без додаткової інтерпретації. Типові шлюзи, що працюють у парадигмі «запит–відповідь», зазвичай обмежуються поверненням помилки у випадку відсутності обов’язкових параметрів. Натомість людино-орієнтована взаємодія вимагає або уточнюючого діалогу, або використання наявного контексту сесії для визначення відсутніх даних. Це означає, що для підтримки сценарного трекінгу необхідні механізми керування станом сесії та короткочасною пам’яттю, які дозволяють накопичувати контекст і коректно продовжувати виконання після уточнення.

Таким чином, аналіз показує, що головними перешкодами на шляху до створення інтелектуальних систем трекінгу є обмеженість традиційних шлюзів у частині семантичного розбору природномовних запитів, складність підтримки варіативної оркестрації сервісів та відсутність механізмів коректної роботи з неповним контекстом. Усунення цих обмежень потребує введення прикладного прошарку на рівні API Gateway, який здатний інтерпретувати намір користувача та керовано перетворювати його у послідовність допустимих інструментальних викликів у межах наявних операцій і політик доступу. Саме це підводить до розробки методу інтелектуальної маршрутизації на основі великих мовних моделей, який поєднує можливості семантичної інтерпретації з формальними обмеженнями виконання, такими як каталог інструментів і перевірювані контракти параметрів.

1.5 Огляд потенціалу великих мовних моделей у задачах інтелектуальної оркестрації

Стрімкий прогрес у галузі штучного інтелекту, пов’язаний із появою архітектури Transformer, привів до розвитку великих мовних моделей (LLM), які на практиці використовуються не лише для генерації тексту, а й для розв’язання завдань інтерпретації запитів, планування дій та взаємодії з прикладними інструментами [19]. У контексті інтелектуальних API Gateway такі моделі розглядаються не як пасивне джерело знань, а як компонент, що підтримує прийняття рішень під час обробки запитів і може застосовуватись як оркестратор у багатоетапних сценаріях розподілених систем [20].

Однією з ключових властивостей LLM, що визначає їхній потенціал для оркестрації, є здатність семантично інтерпретувати намір користувача з урахуванням контексту та неявних умов [21]. На відміну від підходів, що спираються на жорсткі правила, такі моделі можуть бути використані для перетворення природномовного запиту у структуроване подання дії, придатне для виконання в мікросервісному середовищі. У практичному сенсі це означає можливість відображення природномовного опису задачі на одну або кілька операцій прикладного інтерфейсу за умови, що ці операції формально описані та доступні системі як інструменти виконання.

Технологічною основою інтеграції LLM у прикладні системи виступає механізм виклику інструментів, який дозволяє обмежити вихід моделі форматом структурованого виклику [22]. У межах цього підходу моделі надаються формальні описи доступних операцій, включаючи призначення та структуру параметрів, а результат роботи моделі має форму вибору операції та заповнення її аргументів у структурованому форматі. Важливим наслідком є те, що задача виділення сутностей із тексту користувача розглядається не як окремий евристичний етап на стороні клієнта, а як частина керованої процедури формування параметрів інструментального виклику. Водночас для промислових систем критичною є вимога перевірюваності такого результату, тому інтеграція LLM має здійснюватися

разом із формальними обмеженнями на допустимі інструменти та на коректність параметрів.

Особливе значення для систем трекінгу замовлень має здатність LLM підтримувати багатоетапні сценарії, де кінцева відповідь або дія визначається послідовністю кроків і залежить від проміжних результатів. У таких випадках модель може використовувати підхід, відомий як ReAct, який поєднує формування наступної дії з урахуванням спостереження від попереднього кроку [23]. Це дає змогу реалізувати кероване багатоетапне планування у ситуаціях, де необхідно звернутися до кількох операцій прикладного інтерфейсу та адаптувати наступний крок відповідно до отриманих даних. При цьому оркестрація не є довільною: вона має виконуватися в межах доступного каталогу інструментів і під контролем механізмів допуску, зокрема валідації параметрів та правил для дій із побічними ефектами, що відповідає вимогам безпеки та керованості.

Окремо варто відзначити, що використання LLM у складі шлюзу потенційно спрощує введення нових операцій у сценарну обробку запитів, якщо такі операції описані у стандартизованому вигляді [24]. У цьому випадку зміна набору інструментів зводиться до оновлення їх метаданих, і модель може використовувати нові можливості без обов’язкового донавчання. Водночас така гнучкість є досяжною лише за умови, що описи інструментів є формалізованими, а результати роботи моделі підлягають валідації, що дозволяє підтримувати контроль якості та передбачуваність поведінки системи [25].

LLM також можуть застосовуватися для підтримки діалогу уточнення у випадках, коли природномовний запит не містить достатньої інформації для коректного формування виклику. Замість повернення технічної помилки система може сформувати уточнююче питання і продовжити сценарій після отримання необхідних параметрів, використовуючи контекст сесії. У такій постановці шлюз підтримує більш природну взаємодію з користувачем, однак зберігає інженерну керованість за рахунок того, що будь-яке виконання здійснюється лише через структуровані інструментальні кроки та після перевірки їх коректності.

Отже, огляд показує, що великі мовні моделі поєднують семантичну інтерпретацію запитів, підтримку багатоетапного планування та механізми взаємодії з зовнішніми інструментами, що робить їх придатною основою для побудови методу інтелектуальної маршрутизації на рівні API Gateway. За умови введення формальних обмежень на доступні інструменти, перевірюваних контрактів параметрів і керованої оркестрації, такий підхід дозволяє зменшити обмеження статичних шлюзів і підвищити адаптивність інформаційних систем трекінгу замовлень у сценаріях, де запит формується природною мовою та залежить від контексту.

1.6 Постановка задачі дослідження

На основі проведеного аналізу предметної області та існуючих технологічних рішень можна зробити висновок, що сучасні системи трекінгу замовлень потребують нового підходу до організації взаємодії між користувачем і мікросервісним середовищем. Класичні рішення класу API Gateway забезпечують ефективну маршрутизацію детермінованих технічних запитів, проте виявляються недостатніми у випадках, коли звернення користувача подано природною мовою, містить кілька пов’язаних дій, умови виконання або неповні параметри. У таких ситуаціях виникає потреба не лише в перенаправленні запиту до відповідного сервісу, а й в інтерпретації наміру користувача, виборі допустимих операцій, формуванні параметрів виклику, урахуванні контексту сесії та керованому виконанні послідовності дій.

Аналіз потенціалу великих мовних моделей показує, що вони можуть бути використані як компонент семантичної інтерпретації та динамічного планування в межах прикладного шлюзу. Водночас застосування LLM у мікросервісному середовищі не може ґрунтуватися лише на вільній генерації тексту, оскільки промислова система потребує контрольованості, перевірюваності та безпечного виконання дій. Отже, постає науково-прикладна задача розробки методу інтелектуальної маршрутизації мікросервісів, який поєднує семантичну обробку

природномовних запитів із формальними механізмами вибору інструментів, валідації параметрів, керування контекстом та агентської оркестрації сервісних викликів.

Метою дослідження є розробка методу інтелектуальної маршрутизації мікросервісів на основі великих мовних моделей для інформаційної системи трекінгу замовлень, який забезпечує перетворення природномовного запиту користувача у структурований, перевірюваний і контрольований план сервісних викликів.

Для досягнення поставленої мети необхідно розв’язати такі задачі:

1) проаналізувати особливості функціонування інформаційних систем трекінгу замовлень та їхні вимоги до обробки користувацьких запитів;

2) дослідити роль API Gateway у мікросервісній архітектурі та виявити обмеження традиційних засобів маршрутизації у сценаріях роботи з природномовними зверненнями;

3) дослідити проблеми семантичного розбору неструктурованих запитів, багатоетапної оркестрації сервісів та роботи з неповним контекстом;

4) проаналізувати можливості великих мовних моделей щодо інтерпретації наміру користувача, підтримки tool calling та динамічного планування дій;

5) розробити метод інтелектуальної маршрутизації мікросервісів, що поєднує семантичний розбір, формальні контракти параметрів, правила допуску дій, керування контекстом сесії та агентську оркестрацію;

6) спроектувати архітектуру інтелектуального шлюзу для системи трекінгу замовлень;

7) реалізувати програмний прототип запропонованого підходу та провести його перевірку на типових сценаріях взаємодії.

Критеріями успішності запропонованого підходу є:

— здатність коректно визначати інструмент або послідовність інструментів для виконання користувацького запиту;

— здатність формувати параметри виклику у структурованому вигляді з подальшою формальною валідацією;

— підтримка уточнювальної взаємодії у випадках неповного запиту;

— забезпечення контрольованого виконання дій у межах дозволеного набору операцій;

— придатність методу до реалізації у вигляді прототипу інтелектуального API Gateway.

Отже, у межах даної кваліфікаційної роботи ставиться задача розробки методу інтелектуальної маршрутизації мікросервісів, орієнтованого на сценарну взаємодію з користувачем у системі трекінгу замовлень. Розв’язання цієї задачі має забезпечити перехід від статичної технічної маршрутизації до керованої семантичної обробки запитів із збереженням формальної перевірюваності, безпеки та інженерної керованості.

2 ПРОЕКТУВАННЯ МЕТОДУ ІНТЕЛЕКТУАЛЬНОЇ МАРШРУТИЗАЦІЇ

2.1 Концептуальна модель інтелектуального шлюзу як багаторівневої системи

У межах даної роботи інтелектуальний шлюз розглядається як багаторівнева система, яка поєднує властивості класичного шлюзу прикладних інтерфейсів з можливостями інтелектуальної інтерпретації запитів. Такий підхід орієнтований на збереження базових вимог корпоративної інфраструктури, зокрема безпеки, контролю доступу та аудиту, і водночас додає когнітивні можливості у вигляді розбору намірів, планування дій та адаптації плану за результатами виконання. Важливою перевагою багаторівневої декомпозиції є можливість незалежного масштабування компонентів відповідно до профілю навантаження, оскільки різні частини шлюзу мають різну вартість виконання та різні вимоги до продуктивності.

2.1.1 Вхідні та вихідні дані системи шлюзу

Функціонально роботу інтелектуального шлюзу доцільно описувати як перетворення запиту користувача у послідовність операцій над мікросервісами з подальшим формуванням відповіді. На вхід системи надходить природномовний або структурований запит користувача p, поточний контекст сесії s, який включає історію діалогу, результати попередніх кроків і службові стани, а також множина доступних інструментів T, що відповідають операціям API після фільтрації за правами доступу. Додатково використовується авторизаційний токен або авторизаційні дані τ, які мають бути проксійовані для доступу до ресурсів. Результатом роботи є відповідь користувачу r, тобто дані трекінгу, підтвердження операції або інший результат успішно виконаної транзакції у підсистемах логістичної чи «замовної» доменної області. Така постановка дозволяє відділити «розуміння запиту» від «виконання технічних викликів» та створює основу для подальшої формалізації семантичного розбору і агентської оркестрації.

2.1.2 Загальна архітектура та рівні абстракції

Для візуалізації взаємодії інтелектуальних компонентів із традиційною мікросервісною інфраструктурою у роботі наведено концептуальну схему шлюзу (рис. 2.1), яка реалізує три рівні обробки запиту з чітко визначеними межами відповідальності. Першим є транспортно-безпековий рівень, який виконує детерміновані операції, незалежні від інтерпретації природної мови, і тому має спрацьовувати до залучення когнітивних компонентів. Його функції включають завершення TLS-з'єднання та базову мережну ізоляцію, застосування обмеження частоти запитів для захисту від пікових навантажень і зловживань, первинну автентифікацію та валідацію токенів із перевіркою підписів і термінів дії, а також стандартні функції шлюзу, зокрема присвоєння ідентифікатора кореляції та додавання службових даних для трасування. Результатом роботи цього рівня є або відхилення запиту у випадку порушення політик, або передача запиту в когнітивний рівень разом із валідаційними атрибутами, зокрема ідентифікатором користувача, набором прав доступу та технічними метаданими запиту.

Другим є когнітивний рівень, який відповідає за перетворення неструктурованого запиту у формалізований план виконання і розглядається як агентська підсистема. У межах цього рівня компонент семантичної маршрутизації виконує попередню класифікацію запиту та визначає, чи належить він до інтелектуального режиму, або може бути оброблений традиційною детермінованою маршрутизацією. Оркестратор на основі великої мовної моделі будує план дій, спираючись на опис доступних інструментів і поточний контекст сесії. Реєстр інструментів надає формалізовані описи операцій мікросервісів, необхідні для коректного формування інструментальних викликів, а компонент стану сесії забезпечує збереження й доступ до контексту, який використовується в циклі планування та уточнення. У результаті когнітивний рівень формує не довільну текстову відповідь, а структурований план, придатний до подальшої перевірки та виконання.

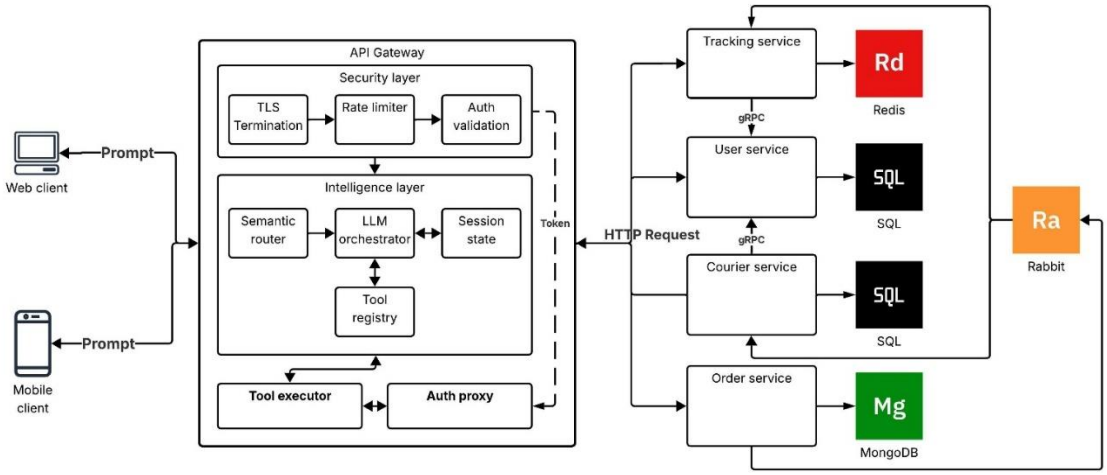

Рисунок 2.1 – Концептуальна схема інтелектуального шлюзу

Третім є інтеграційно-виконавчий рівень, який перетворює кроки плану на конкретні мережеві виклики до мікросервісів і повертає результати назад до оркестратора. Саме на цьому рівні реалізується практичне виконання інструментальних викликів і формування спостережень, що надалі впливають на наступні кроки оркестрації. Таким чином, архітектура підтримує замкнений контур обробки, де виконання не відокремлене від планування, а виступає джерелом даних для подальшого прийняття рішень у межах сесії.

2.1.3 Режими роботи шлюзу та принципи ізоляції

Для забезпечення сумісності з класичними клієнтами та одночасної підтримки природномовних сценаріїв інтелектуальний шлюз функціонує у двох режимах, які відрізняються рівнем детермінізму та способом прийняття рішень. Детермінований режим виконує стандартні для шлюзу операції та маршрутизацію за заздалегідь визначеними правилами, що забезпечує передбачувану затримку, відтворюваність поведінки та простішу верифікацію й експлуатацію. У цьому режимі обробляються запити типу «отримати статус замовлення за номером», «отримати список відправлень», «завантажити накладну», тобто сценарії, де клієнт звертається до конкретної операції API і не потребує оркестрації. З огляду на

вимоги стабільності, детермінований режим у роботі розглядається як базовий і має працювати незалежно від доступності когнітивної підсистеми.

Інтелектуальний режим активується тоді, коли запит подано природною мовою, містить кілька намірів або потребує послідовності викликів різних мікросервісів чи умовного виконання. Ключова відмінність цього режиму полягає в тому, що шлюз не просто пересилає запит, а формує план виконання у вигляді впорядкованого набору кроків із чіткими параметрами виклику та очікуваними результатами, після чого кожен крок перетворюється у конкретний виклик внутрішнього сервісу через виконуваний рівень. Для мінімізації ризиків і підтримки вимог до якості обслуговування у шлюзі передбачається механізм вибору режиму, який визначає спосіб обробки запиту як детерміноване рішення, що базується на типі запиту, наявності маркерів сценарності, налаштуваннях клієнтського каналу, а також доступності когнітивної підсистеми і бюджеті часу. Важливою вимогою є керована деградація: якщо інтелектуальний режим недоступний або перевищує допустимий бюджет, запит або переводиться у детермінований режим, або завершується коректним повідомленням, що вимагає структурованого звернення чи уточнення.

Щоб зробити систему безпечною, придатною для тестування й контрольованою, у роботі підкреслюється принцип ізоляції рівнів. Безпековий рівень не повинен залежати від великої мовної моделі і має відпрацьовувати до будь-яких когнітивних рішень, а виконуваний рівень не приймає рішень, а лише реалізує формалізований план. Додатково наголошується на тому, що множина інструментів T формується після авторизаційної фільтрації і саме вона використовується на когнітивному етапі, що обмежує інтелектуальний режим рамками дозволених операцій. Розділення стану і логіки підтримує ізоляцію: контекст сесії зберігається та використовується через компонент стану, що дозволяє відокремити процес прийняття рішень від технічних деталей виконання та забезпечити відтворюваність поведінки в межах сесії.

2.2 Формалізація методу семантичного розбору та моделі метаданих інструментів

Ключовою умовою інтелектуальної маршрутизації є перетворення природномовного запиту користувача у формалізований виклик (або послідовність викликів) мікросервісних операцій. На відміну від класичного API Gateway, де відповідність «запит → маршрут» задається детермінованою таблицею, у даній роботі вводиться метод семантичного розбору, який забезпечує виділення наміру користувача та релевантних сутностей предметної області трекінгу замовлень, вибір інструмента (операції API) з множини дозволених, а також формування параметрів виклику у форматі, що підлягає машинній перевірці. Загальна логіка такого перетворення відображена на рисунку 2.2, де показано концептуальне «семантичне мапування»: запит проходить через етап інтерпретації наміру та видобування параметрів і перетворюється у валідний виклик програмної функції або сервісної операції.

Рисунок 2.2 – Схема семантичного мапування

Вхідними даними методу є текст запиту користувача p, контекст сесії s (історія діалогу, відомі значення атрибутів, проміжні результати попередніх кроків), а також множина доступних інструментів T, сформована після авторизаційної фільтрації відповідно до прав доступу. Результатом є структуроване подання наміру у вигляді об’єкта виклику c, що містить ідентифікатор обраної

операції та аргументи, узгоджені з її контрактом. У загальному вигляді семантичний розбір можна розглядати як відображення $f(p,s,T) \to c$, де c не є довільним текстом а формальним описом майбутньої дії, який може бути валідований і, за необхідності, відхилений до моменту фактичного виконання запиту в мікросервісному середовищі. Якщо інформації недостатньо для однозначного вибору операції або заповнення її параметрів, коректним результатом методу є запит на уточнення, а не «домислення» відсутніх значень.

Щоб така валідація була можливою, кожен інструмент у реєстрі описується метаданими, які виконують роль семантичного «інтерфейсу» для оркестратора. У практичному сенсі інструментом є конкретна операція внутрішнього API (HTTP endpoint або gRPC-метод) мікросервісів системи трекінгу, а метадані задають не лише технічну сигнатуру, але й інтерпретований опис призначення, обмеження виклику та структуру параметрів. У даній роботі як базовий формат контракту параметрів застосовується JSON Schema, оскільки вона дозволяє формально описувати типи даних, обов’язковість полів, обмеження значень, вкладені структури та правила валідації [26]. Важливо, що відповідність параметрів JSON Schema розглядається як необхідна умова допуску сформованого виклику до виконання.

Семантичний розбір у межах запропонованого методу доцільно трактувати як багатокрокову трансформацію, де на початкових етапах із запиту вилучаються ключові сутності доменної області (наприклад, номер замовлення або відправлення, назва населеного пункту, часові умови), а далі виконується «зв’язування» (binding) вилучених сутностей з параметрами обраного інструмента [27]. Практично процедура організована у два послідовні етапи: спочатку множина дозволених інструментів T редукується до обмеженого набору кандидатів на основі семантичної близькості, після чого формується структурований виклик для одного з кандидатів із заповненими параметрами. Важливою відмінністю від пошуку за ключовими словами є використання семантичної близькості на основі векторних представлень (embeddings): замість прямого збігу термінів шлюз зіставляє векторне подання запиту користувача з векторними описами інструментів у реєстрі, що

дозволяє обробляти семантично еквівалентні формулювання на кшталт «де моя посилка?» та «надай статус доставки».

Разом з тим, імовірнісна природа LLM вимагає введення механізмів, які забезпечують детермінованість і перевірюваність результату семантичного розбору. Тому метод передбачає жорстке обмеження формату виходу мовної моделі через механізми примусового виклику функцій (function calling / tool calling): модель повинна повертати структурований об’єкт виклику з ідентифікатором інструмента та аргументами, які проходять перевірку на відповідність схемі. Після генерації застосовується програмна пост-валидація: перевіряється, що обраний інструмент належить множині дозволених T, що всі обов’язкові поля присутні, що типи та формати коректні, а значення параметрів не суперечать доменним обмеженням. Якщо жоден інструмент-кандидат не демонструє достатньої семантичної відповідності запиту або параметри не можуть бути визначені однозначно, система переходить у режим уточнення.

Послідовність виконання методу семантичного розбору включає такі етапи:

1) попередню нормалізацію запиту користувача та виділення ключових мовних конструкцій;

2) урахування контексту сесії s, зокрема історії попередніх звернень, уже відомих параметрів і проміжних результатів;

3) фільтрацію множини інструментів за правилами доступу та отримання множини дозволених інструментів $K \subseteq T$;

4) семантичне зіставлення запиту p з описами інструментів із множини K;

5) вибір найбільш релевантного інструмента або обмеженого набору кандидатів;

6) формування структурованого об’єкта виклику з аргументами у форматі, що підлягає машинній перевірці;

7) пост-валидацію об’єкта виклику на відповідність контракту інструмента, типам даних, обов’язковим полям і доменним обмеженням;

8) прийняття рішення про подальший перехід: виконання виклику, уточнення параметрів, відмову або завершення з помилкою.

Отже, на відміну від традиційної схеми маршрутизації, де відповідність між запитом і сервісом задається наперед визначеним правилом, запропонований метод виконує кероване семантичне відображення запиту користувача на допустиму сервісну операцію з обов’язковою перевіркою коректності її параметрів.

2.3 Проектування логіки агентської оркестрації та динамічного планування

У системах трекінгу замовлень значна частина користувацьких звернень не зводиться до простого виклику одного наперед визначеного маршруту. На практиці природномовний запит може бути неповним, неоднозначним, контекстно залежним або таким, що потребує кількох послідовних звернень до різних сервісів. Саме тому в межах запропонованого методу маршрутизація розглядається не як статичне зіставлення запиту з API-методом, а як процес поетапного прийняття рішень щодо подальших дій системи.

Основою такого підходу є агентська оркестрація, у межах якої інтелектуальний шлюз виконує роль координатора, що перетворює природномовне звернення користувача у контрольований план сервісних викликів. На відміну від традиційного API Gateway, який працює за жорстко визначеними правилами маршрутизації, інтелектуальний шлюз повинен уміти визначати намір користувача, встановлювати, яких саме даних бракує для виконання запиту, обирати допустимий інструмент або послідовність інструментів, а також коригувати план залежно від результатів попередніх кроків.

Процес агентської оркестрації доцільно подати як послідовність дискретних станів, кожен з яких має чітко визначену функцію у загальному циклі обробки запиту. На першому етапі виконується інтерпретація звернення користувача та визначення його основної інтенції. Далі формується початковий план дії, який містить або один виклик інструмента, або послідовність кількох викликів, або рішення про необхідність уточнення вхідних даних. Після цього здійснюється перевірка допустимості вибраного інструмента, валідація параметрів, виконання

сервісного виклику, аналіз отриманого результату та оновлення контексту сесії. Завершується цикл або формуванням відповіді користувачу, або переходом до наступної ітерації, якщо для досягнення цілі необхідно виконати ще один крок. Для більш формального подання логіки роботи оркестратора послідовність його дій доцільно подати у вигляді алгоритму:

1) отримання природномовного запиту користувача;

2) визначення основної інтенції звернення та виділення ключових сутностей і параметрів;

3) аналіз наявного контексту сесії та перевірка, чи достатньо даних для подальшого виконання;

4) формування початкового плану дій: вибір одного інструмента, послідовності інструментів або рішення про необхідність уточнення;

5) перевірка, чи входить вибраний інструмент до дозволеного набору для поточного сценарію;

6) валідація аргументів відповідно до контракту інструмента;

7) у разі коректності параметрів — виконання сервісного виклику;

8) аналіз отриманого результату та оновлення контексту сесії;

9) прийняття рішення про завершення сценарію або перехід до наступної ітерації планування;

10) формування підсумкової відповіді користувачу.

Таким чином, динамічне планування в запропонованому методі не є довільним або повністю неформалізованим. Кожен перехід між етапами визначається набором умов, що враховують наявність обов’язкових параметрів, результати попередніх викликів, обмеження доступу, ризиковість операції та ознаки завершеності сценарію. Завдяки цьому вдається поєднати гнучкість інтерпретації природної мови з вимогами до контрольованості виконання дій у мікросервісному середовищі.

У межах запропонованого підходу результат планування на кожній ітерації може належати до одного з трьох базових типів. Перший тип — це безпосереднє виконання інструмента, коли запит користувача містить достатньо даних і може

бути перетворений у коректний сервісний виклик. Другий тип — уточнювальна взаємодія, коли обов'язкові параметри відсутні, суперечливі або не можуть бути надійно виведені з контексту. Третій тип — завершення сценарію без виклику інструмента, якщо запит порушує політику доступу, виходить за межі підтримуваних можливостей системи або не потребує звернення до сервісів. Така класифікація дозволяє формально відокремити коректне виконання дій від випадків недостатності даних або недопустимості операції. Основні типи таких рішень наведено в таблиці 2.1.

Таблиця 2.1 – Основні типи рішень оркестратора в процесі динамічного планування

Тип рішення	Умова прийняття	Результат
Виконати інструмент	Намір визначено, потрібний інструмент дозволений, параметри повні та коректні	Виконується сервісний виклик і результат передається на наступний етап аналізу
Уточнити параметри	Відсутні обов'язкові параметри, дані суперечливі або їх неможливо надійно визначити з контексту	Користувачу надсилається уточнювальне запитання без виклику мікросервісу
Відхилити запит	Обраний інструмент заборонений політиками доступу, запит виходить за межі підтримуваних дій або містить недопустимі параметри	Сценарій зупиняється, користувач отримує повідомлення про неможливість виконання дії
Завершити сценарій	Досягнуто цільового результату, подальші кроки не потрібні або вичерпано ліміт ітерацій	Формується підсумкова відповідь користувачу та цикл оркестрації завершується

Наведена класифікація рішень дозволяє формалізувати поведінку оркестратора та відокремити коректне інструментальне виконання від ситуацій недостатності даних, порушення політик доступу або логічного завершення сценарію. Це, у свою чергу, підвищує контрольованість обробки природномовних запитів і забезпечує узгодження між гнучкістю інтерпретації звернення та вимогами до надійності сервісної взаємодії.

Особливістю задачі трекінгу замовлень є те, що один користувацький намір може вимагати кількох взаємопов'язаних сервісних викликів. Наприклад, запит про статус останнього замовлення спочатку потребує визначення останнього релевантного замовлення користувача, після чого виконується окремий виклик для отримання його поточного стану. Аналогічно звернення щодо очікуваного часу

доставки може вимагати попереднього встановлення ідентифікатора замовлення, перевірки статусу відправлення та лише потім отримання розрахункового часу прибуття. У таких сценаріях оркестратор повинен працювати не як механізм одноразового вибору інструмента, а як засіб побудови ланцюжка залежних дій.

Для забезпечення коректності такого ланцюжка кожен інструмент у системі розглядається як формалізований об’єкт із визначеним призначенням, допустимими параметрами, типом результату та обмеженнями використання. Це означає, що агентська оркестрація спирається не лише на семантичне зіставлення інтенції із сервісною функцією, а й на формальні контракти інструментів. Саме через це після формування плану обов’язково виконується перевірка структури аргументів, їх типів, повноти та допустимості. Якщо параметри не відповідають контракту, система не переходить до фактичного виклику мікросервісу, а або ініціює уточнення, або відхиляє операцію. Такий механізм знижує ризик некоректного або небезпечного виконання запиту.

Окрему роль у логіці оркестрації відіграють правила допуску дій. У системі трекінгу замовлень не всі сервісні операції мають однаковий рівень ризику. Запити на читання інформації про стан замовлення можуть виконуватися автоматично в межах дозволеного набору інструментів, тоді як операції, пов’язані зі зміною параметрів доставки, адреси або інших атрибутів замовлення, повинні додатково перевірятися з урахуванням ролі користувача, рівня доступу та характеру операції. Отже, оркестратор не може вважатися автономним виконавцем довільних дій. Його функція полягає у побудові плану лише в межах заздалегідь визначеної множини дозволених інструментів та політик безпеки.

Ще однією принциповою вимогою до агентської оркестрації є обмеження циклу виконання. У практичних умовах інтелектуальна система не повинна переходити в неконтрольований режим генерації нескінченних або надлишкових викликів. Тому в межах запропонованого методу доцільно вводити максимальну кількість ітерацій планування та виконання, після досягнення якої сценарій має бути завершений із поверненням результату, повідомленням про недостатність даних або діагностикою помилки. Такий підхід забезпечує відтворюваність

поведінки шлюзу, передбачуваність навантаження на зовнішні сервіси та спрощує подальший аналіз трасувальної інформації.

Важливою характеристикою динамічного планування є його адаптивність до проміжних результатів. Якщо на одному з етапів отримано неочікуваний або неповний результат, оркестратор не повинен механічно продовжувати попередньо сформований ланцюжок дій. Натомість він має оцінити новий стан сценарію, співвіднести його з кінцевою метою користувача і прийняти рішення про доцільність наступного кроку. Саме ця властивість відрізняє агентську оркестрацію від звичайного сценарного програмування, у якому послідовність викликів повністю фіксується наперед.

Отже, логіка агентської оркестрації у запропонованому методі полягає у перетворенні природномовного звернення користувача на послідовність контрольованих, перевірюваних і допустимих дій у межах мікросервісного середовища. Її ключовими складовими є інтерпретація інтенції, формування плану, перевірка допустимості інструмента, валідація параметрів, виконання сервісного виклику, аналіз результату та оновлення поточного стану сценарію. Такий підхід створює основу для реалізації інтелектуального шлюзу, який поєднує гнучкість роботи з природною мовою з вимогами до надійності, безпеки та керованості сервісної взаємодії. Логічним продовженням цієї моделі є розгляд механізмів управління контекстом, станом і пам’яттю сесії, оскільки саме вони забезпечують зв’язність багатоетапної взаємодії між користувачем і системою.

2.4 Управління контекстом, станом та пам'яттю сесії

Ефективна обробка природномовних запитів у системі трекінгу замовлень неможлива без збереження та цілеспрямованого використання інформації, накопиченої в процесі взаємодії користувача з інтелектуальним шлюзом. Якщо кожне звернення розглядати як повністю ізольоване, система втрачає здатність підтримувати багатоетапний сценарій, враховувати результати попередніх викликів і коректно продовжувати діалог після уточнювальних запитань. Саме

тому в межах запропонованого методу важливе місце займає підсистема управління контекстом, станом і пам’яттю сесії, яка забезпечує зв’язність оркестрації та узгодженість рішень на послідовних етапах виконання.

У межах запропонованого підходу доцільно розрізняти три взаємопов’язані, але не тотожні поняття: контекст, стан і пам’ять сесії. Контекстом є сукупність релевантних даних, які використовуються для прийняття рішення на поточному кроці планування. До нього можуть належати виявлені сутності запиту, встановлені ідентифікатори замовлень або відправлень, результати попередніх сервісних викликів, маркери відсутніх параметрів, а також ознаки наміру користувача, уточнені в попередніх репліках. Стан сесії відображає поточне положення сценарію взаємодії, тобто інформацію про те, на якому етапі перебуває оркестрація, які дії вже виконано, які параметри ще очікуються і чи досягнуто умов завершення. Пам’ять сесії виступає короткочасним операційним сховищем, у якому фіксуються структуровані факти і проміжні результати, необхідні для збереження зв’язності між окремими кроками сценарію.

Потреба в такій підсистемі особливо проявляється в багатоетапних сценаріях, коли один користувацький запит не може бути оброблений одним викликом інструмента. Наприклад, звернення щодо статусу останнього замовлення потребує щонайменше двох послідовних дій: спочатку необхідно визначити, яке саме замовлення є останнім у контексті поточного користувача, а потім виконати звернення до сервісу відстеження для отримання його актуального стану. Аналогічно, якщо користувач надає неповний запит і система змушена поставити уточнювальне запитання, подальша відповідь користувача повинна інтерпретуватися не ізольовано, а з урахуванням уже відомої інтенції та попередньо сформованого плану дій. Без збереження контексту така взаємодія або переривається, або змушує систему повторно проходити повний цикл інтерпретації.

Одним із базових принципів управління контекстом у запропонованому методі є його мінімізація. До підсистеми планування доцільно передавати не повну історію діалогу в сирому текстовому вигляді, а лише ті структуровані факти, які

мають значення для поточного кроку оркестрації. До них належать ідентифікатори замовлень, результати попередніх сервісних викликів, значення ролей користувача, ознаки наявності або відсутності критично важливих параметрів, а також службові маркери поточного етапу сценарію. Такий підхід дає змогу зменшити обсяг даних, які повинні враховуватися під час планування, підвищити відтворюваність рішень оркестратора та одночасно знизити ризики витоку надлишкової інформації. Практичне значення цього принципу полягає в тому, що в сесійній пам’яті доцільно зберігати саме структуровані факти та проміжні результати, а не всю історію повідомлень у необробленому вигляді.

Формально управління станом сесії можна подати як процес послідовного оновлення контексту на основі нових спостережень. На кожному кроці виконання новий стан формується шляхом інтеграції попереднього стану з новими даними, отриманими або з репліки користувача, або з результатів інструментального виклику. Це означає, що контекст не є статичним описом сесії, а змінюється в міру просування оркестрації. Завдяки цьому інтелектуальний шлюз переходить від ролі простого посередника між клієнтом і мікросервісами до ролі керованого координатора, здатного враховувати накопичений досвід поточної взаємодії при виборі наступного кроку. Саме така модель дозволяє адаптувати план дії після кожного нового спостереження та уникати механічного виконання фіксованого ланцюжка викликів.

Життєвий цикл пам’яті сесії охоплює кілька послідовних етапів. На початку взаємодії створюється початковий контекст, який містить базові відомості про користувача та виявлену інтенцію звернення. Після кожного кроку оркестрації цей контекст оновлюється новими фактами, зокрема результатами сервісних викликів, уточненими параметрами або інформацією про помилки та відхилені дії. У разі втрати актуальності окремих даних вони повинні вилучатися або позначатися як застарілі, щоб не впливати на наступні рішення планувальника. Завершальним етапом є очищення пам’яті сесії після досягнення цільового результату або після завершення встановленого часу життя контексту. Така організація дає змогу

підтримувати лише актуальний стан взаємодії та не перетворювати сесійну пам’ять на неконтрольоване довготривале сховище.

З практичної точки зору пам’ять сесії в інтелектуальному шлюзі не повинна розглядатися як повноцінна транзакційна база даних. Її призначення полягає не в тривалому зберіганні доменних даних, а в оперативному утриманні короткоживучого набору структурованих відомостей, необхідних для підтримки безперервності сценарію. До таких даних можуть належати ідентифікатори замовлень або відправлень, останні отримані стани, маркери очікуваних параметрів після уточнення, ознаки дозволеності або недопустимості окремих інструментів, а також допоміжні атрибути для відтворюваності логіки виконання. Відповідно, до такого сховища висуваються вимоги високої швидкодії, доступності з кількох екземплярів шлюзу в разі горизонтального масштабування, підтримки короткого часу життя даних і можливості автоматичного очищення неактуальних записів.

Окреме значення має ізоляція контексту в багатокористувацькому середовищі. Кожен об’єкт стану повинен бути жорстко пов’язаний з ідентифікатором користувача або конкретної сесії, отриманим у процесі автентифікації. Це дає змогу запобігти перетіканню контексту між різними користувачами та забезпечує коректне виконання персоналізованих сценаріїв. У випадку системи трекінгу замовлень така ізоляція є принципово важливою, оскільки навіть короткочасне змішування сесійних даних може призвести до некоректного звернення до сервісів, хибної інтерпретації наміру або розкриття інформації про чуже замовлення. Отже, управління станом сесії є не лише питанням функціональності, а й важливим елементом безпеки та контролю доступу.

Для прототипу інтелектуального шлюзу доцільно розглядати Redis як сховище Session State, у якому кожна сесія має окремий ключовий простір, прив’язаний до ідентифікатора користувача або сеансу, а записи автоматично підпорядковуються політикам експірації. Це дозволяє забезпечити швидкий доступ до контексту незалежно від того, який екземпляр шлюзу обробляє черговий крок сценарію, що особливо важливо в умовах масштабування застосунку.

Перевага Redis у задачі зберігання сесійного контексту полягає не лише у високій швидкодії, а й у підтримці механізму TTL (time-to-live) [28]. Завдяки TTL для кожного ключа можна задати час життя, після завершення якого відповідний запис автоматично видаляється. Для підсистеми пам’яті сесії це означає можливість природного завершення життєвого циклу контексту без ручного адміністрування, автоматичне очищення застарілих даних і зменшення ризику накопичення неактуальної інформації. Таким чином, механізм TTL безпосередньо відтворює вимогу короткочасної пам’яті, сформульовану на рівні методу, і дозволяє реалізувати контрольоване “забування” проміжного стану після завершення або переривання сценарію взаємодії.

Отже, управління контекстом, станом і пам’яттю сесії є невід’ємною складовою запропонованого методу інтелектуальної маршрутизації. Саме воно забезпечує збереження зв’язності багатоетапної взаємодії, дозволяє оркестратору адаптувати план дій до проміжних результатів, запобігає повторному запитуванню вже відомих параметрів і створює основу для контрольованого та безпечного виконання сервісних викликів. Практична реалізація цієї підсистеми через розподілений кеш з підтримкою TTL, зокрема на базі Redis, логічно випливає з вимог до короткочасної пам’яті, масштабованості, ізоляції та автоматичного очищення сесійних даних. Це, своєю чергою, формує підґрунтя для подальшого розгляду архітектурних рішень, орієнтованих на забезпечення продуктивності та масштабованості інтелектуального шлюзу.

2.5 Архітектурні рішення для забезпечення масштабованості та продуктивності

Запропонований інтелектуальний шлюз, на відміну від класичного API Gateway, додає обчислювально «важкий» когнітивний контур, який включає семантичний розбір, вибір інструментів, формування плану та ітеративне виконання сценаріїв. Тому забезпечення масштабованості та продуктивності розглядається як складова проектування методу, а не як другорядна деталь

реалізації. У даній роботі масштабованість інтерпретується як здатність системи зберігати прийнятні значення затримки та пропускної здатності при зростанні кількості запитів і розширенні каталогу інструментів, тоді як продуктивність — як здатність обробляти запити в межах обмеженого бюджету обчислень, зокрема з контрольованою кількістю звернень до когнітивної підсистеми.

Базовим архітектурним рішенням, що впливає на масштабування, є шарова декомпозиція шлюзу, визначена у підрозділі 2.1: транспортно-безпековий рівень обробляє максимальну частку трафіку детерміновано, а когнітивний рівень залучається лише для природномовних або сценарних запитів. Такий поділ дозволяє розвести навантаження за профілем: прості запити на читання стану виконуються найкоротшим шляхом і не залежать від когнітивного контуру, тоді як складні сценарії отримують окремий обчислювальний бюджет і не створюють деградації для решти користувачів. У результаті інтелектуальний режим виступає надбудовою над класичною маршрутизацією, а не її повною заміною.

Другим ключовим рішенням є обмеження вартості когнітивного контуру за рахунок редукції простору пошуку на етапі семантичного добору інструментів, описаного у підрозділі 2.2. Повний каталог операцій може бути значним, тому передавання всього набору інструментів у контекст планування збільшує затримку та підвищує ризик помилкового вибору. Використання семантичної близькості для відбору обмеженого набору кандидатів зменшує обсяг контексту і стабілізує поведінку системи, оскільки планування здійснюється в межах вузького кандидатного набору. При цьому у випадках низької релевантності кандидатів система переходить до уточнення замість спроби ініціювати дію на основі недостатньої визначеності, що одночасно зменшує ризик помилкових викликів і запобігає повторним витратам обчислень через некоректне виконання.

Окремим питанням є керування затримки як системною властивістю. Для систем трекінгу замовлень час відповіді визначає практичну корисність сервісу, тому проектування шлюзу доцільно здійснювати з урахуванням «бюджету» на обробку запиту. У детермінованому режимі цей бюджет має бути мінімальним і прогнозованим, а в інтелектуальному режимі необхідно фіксувати межі складності

сценарію, зокрема обмеження на кількість кроків плану та загальний час оркестрації. Такі межі потрібні для запобігання неконтрольованій багатоетапній обробці і для керованої деградації: якщо сценарій не може бути завершений у межах допустимого бюджету, коректною реакцією є завершення з поясненням або перехід до уточнення, а не спроба продовжувати виконання без гарантії результату.

Продуктивність виконуваного контуру залежить від того, як організовано багатоетапні звернення до мікросервісів. У типових сценаріях трекінгу є кроки, які мають бути послідовними через залежність від результатів попереднього кроку, однак у випадках відсутності залежностей може бути доцільною паралелізація викликів, щоб скоротити загальний час сценарію. Крім того, для зменшення зайвого навантаження важливо уникати повторних звернень до однакових операцій у межах одного сценарію, якщо результат уже отримано і збережено у стані сесії, описаному у підрозділі 2.4. Це узгоджується з ідеєю керованої оркестрації, де проміжні результати виступають спостереженнями та використовуються для наступних кроків планування.

У контексті масштабованості важливим є також зменшення повторюваних обчислень у когнітивній частині. Оскільки реєстр інструментів є стандартизованим описом доступних операцій, його артефакти повинні розглядатися як підготовлені дані, що використовуються багаторазово, а не як дані, які формуються заново для кожного запиту. Аналогічно, результати попередніх кроків у межах сесії повинні повторно використовуватися як частина контексту, щоб зменшувати кількість як внутрішніх викликів мікросервісів, так і повторних етапів семантичного розбору.

Нарешті, забезпечення масштабованості та продуктивності неможливе без вимірювальності процесу. Оскільки у постановці задачі задекларовано порівняння підходів за швидкодією, у проектних рішеннях доцільно передбачити метрики, які прямо відображають роботу методу: частку запитів, оброблених у детермінованому та інтелектуальному режимах; середню тривалість семантичного добору; середню кількість кроків у плані; частоту уточнень; частку успішно виконаних сценаріїв; а також розподіл часу між етапами розбору, планування та виконання. Такі показники дозволяють виявляти вузькі місця, підтверджувати ефективність

запропонованих рішень та формують основу для подальшого експериментального оцінювання.

У підсумку, архітектурні рішення для масштабованості та продуктивності в даній роботі ґрунтуються на керованому розділенні режимів обробки, редукції набору інструментів при семантичному доборі, введенні обмежень на складність оркестрації, оптимізації багатоетапних звернень через повторне використання проміжних результатів, а також на забезпеченні вимірювальності ключових етапів методу. Це узгоджується з побудованою у підрозділах 2.1–2.4 моделлю інтелектуального шлюзу й створює основу для розгляду механізмів безпеки та авторизації при виконанні інтелектуальних запитів у наступному підрозділі.

2.6 Механізми авторизації та безпеки при виконанні інтелектуальних запитів

Інтелектуальна маршрутизація запитів у мікросервісному середовищі створює додаткові вимоги до безпеки порівняно з традиційними API Gateway. Якщо в класичному шлюзі набір маршрутів, допустимих параметрів і сценаріїв взаємодії визначається переважно статично, то в інтелектуальному шлюзі частина логіки прийняття рішень переноситься на рівень інтерпретації природномовного запиту, вибору інструмента та формування послідовності дій. За таких умов безпека повинна забезпечуватися не лише на рівні мережевої взаємодії чи автентифікації користувача, а й безпосередньо на рівні механізмів планування, оркестрації та виконання інструментальних викликів.

Ключова особливість цієї задачі полягає в тому, що природномовний запит не може вважатися безпосередньою командою до виконання дії. Він виступає лише вхідним сигналом для побудови плану, який повинен пройти послідовність перевірок перед фактичним зверненням до мікросервісів. Саме тому в межах запропонованого методу авторизація і безпека розглядаються як багаторівнева система обмежень, що охоплює перевірку ідентичності користувача, визначення

ролей і прав доступу, фільтрацію дозволених інструментів, валідацію аргументів, контроль сесійного контексту та аналіз допустимості кожного кроку оркестрації.

Базовим рівнем захисту є автентифікація користувача та встановлення його цифрової ідентичності. У системі трекінгу замовлень це необхідно для однозначного пов’язування природномовного звернення з конкретним обліковим записом, набором доступних даних і політиками виконання дій. Результат автентифікації виступає вхідною умовою для всіх подальших рішень оркестратора, оскільки саме на його основі формується множина допустимих інструментів і визначається, які операції система може виконувати автоматично, а які вимагають обмеження або додаткової перевірки.

Наступним рівнем є авторизація як механізм перевірки допустимості дії для конкретного користувача. У межах інтелектуального шлюзу авторизація не повинна обмежуватися лише доступом до HTTP-маршруту. Вона має застосовуватися до кожного інструмента, який може бути обраний у процесі оркестрації, а також до конкретного типу операції, яку цей інструмент виконує. Це означає, що рішення про виклик сервісу повинно прийматися не лише на підставі семантичної відповідності інтенції користувача, а й з урахуванням ролі користувача, контексту поточної сесії, типу запитуваної дії та її потенційного впливу на доменні дані.

У задачі трекінгу замовлень доцільно розрізняти операції читання та операції модифікації. Запити, пов’язані з отриманням статусу замовлення, інформації про доставку, місцезнаходження відправлення або історії змін, зазвичай мають нижчий рівень ризику і можуть виконуватися в межах дозволеного набору read-only інструментів. Натомість дії, що стосуються зміни параметрів доставки, оновлення адреси, скасування замовлення, перенесення часу отримання або будь-яких інших змін стану системи, повинні розглядатися як потенційно ризикові та підлягати суворішим обмеженням. Таким чином, інтелектуальний шлюз не може бути універсальним виконавцем будь-якого наміру, вираженого природною мовою. Його автономність повинна бути обмежена множиною попередньо визначених дозволених дій.

Одним із центральних механізмів безпеки в такій архітектурі є модель дозволених інструментів. Її сутність полягає в тому, що оркестратор не має права вільно обирати будь-який сервіс або метод, доступний у внутрішньому середовищі. Натомість кожна сесія або кожен тип сценарію повинні бути пов’язані з фіксованим набором інструментів, які вважаються допустимими для відповідного користувача та конкретної інтенції. Усі інші інструменти мають бути недоступними для планування та виконання, навіть якщо формально існує семантична подібність між запитом користувача і відповідною операцією. Такий підхід суттєво зменшує ризик несанкціонованого звернення до внутрішніх сервісів і дає змогу контролювати межі автономної поведінки шлюзу.

Важливим компонентом захисту є також валідація аргументів інструментів. Навіть якщо вибраний інструмент входить до дозволеного набору, його виклик не може виконуватися без попередньої перевірки структури вхідних параметрів. Доцільно перевіряти повноту обов’язкових полів, коректність типів, допустимість форматів, межі значень, а також логічну узгодженість між параметрами. Наприклад, якщо користувач запитує інформацію про конкретне замовлення, система повинна переконатися, що вказаний ідентифікатор справді існує у дозволеному контексті поточного користувача, а не був довільно згенерований або підставлений із зовнішнього тексту. Завдяки цьому знижується ризик як випадкових помилок виконання, так і навмисного маніпулювання параметрами.

Окремого розгляду потребує проблема prompt injection, яка є однією з найбільш характерних загроз для систем, що використовують великі мовні моделі або моделі, сумісні з модельною інтерпретацією природної мови. Сутність цієї загрози полягає в тому, що користувацький запит або зовнішні текстові дані можуть містити спеціально сформульовані інструкції, спрямовані на зміну внутрішньої логіки планування, ігнорування політик безпеки або ініціювання недозволених викликів. У контексті інтелектуального шлюзу це може проявлятися як спроба нав’язати системі використання непризначеного інструмента, обійти перевірку ролей або змусити оркестратор інтерпретувати текст користувача не як дані, а як керівну команду. Саме тому природномовний вхід повинен трактуватися

виключно як джерело наміру та параметрів, але не як авторитетне джерело політик або інструкцій до внутрішніх механізмів виконання.

Протидія таким загрозам потребує чіткого розмежування між системними інструкціями, описами інструментів і користувацьким вмістом. Логіка дозволених дій, політики доступу та правила переходу між етапами оркестрації мають формуватися на рівні застосунку і не повинні змінюватися залежно від того, як сформульовано запит користувача. Іншими словами, модель або оркестратор можуть допомагати інтерпретувати інтенцію, але не повинні самостійно визначати межі допустимості дій. Це рішення є принциповим для побудови безпечного інтелектуального шлюзу, оскільки переносить критичні механізми контролю з рівня мовної інтерпретації на рівень формально визначених правил застосунку.

Значну роль у забезпеченні безпеки відіграє ізоляція контексту сесії. Як було показано в попередньому підрозділі, сесійна пам’ять використовується для збереження проміжних результатів, ідентифікаторів доменних об’єктів, маркерів поточного стану сценарію та інших структурованих фактів, необхідних для багатоетапної взаємодії. Проте така пам’ять несе ризик помилкового використання чужого контексту або повторного застосування застарілих даних. Тому кожен об’єкт сесійного стану повинен бути однозначно прив’язаний до ідентичності користувача або його сесії, а всі рішення про використання збережених фактів мають прийматися лише після перевірки їх актуальності та належності. У системі трекінгу замовлень це особливо важливо, оскільки навіть короткочасне змішування контексту між користувачами може призвести до розкриття інформації про чуже замовлення або до виконання помилкової дії на основі неактуальних даних.

Ще одним необхідним механізмом є обмеження автономності багатоетапної оркестрації. У процесі планування інтелектуальний шлюз може послідовно переходити між кількома кроками, використовуючи результати попередніх викликів для формування нових рішень. Без чітких обмежень така поведінка здатна призвести до надлишкових, циклічних або небезпечних послідовностей дій. Саме тому доцільно вводити максимальну кількість кроків оркестрації, розмежовувати сценарії, у яких дозволено лише read-only виклики, від сценаріїв із потенційно

змінюючими операціями, а також фіксувати причини завершення сценарію: успішне досягнення цілі, недостатність даних, порушення політики доступу або помилку зовнішнього сервісу. Такі обмеження роблять поведінку системи передбачуваною і спрощують аудит рішень, прийнятих у межах інтелектуального виконання.

Із безпекової точки зору важливим є й журналювання дій оркестратора. Для кожного кроку доцільно фіксувати, яка інтенція була визначена, який інструмент обрано, чи входив він до дозволеного набору, які аргументи передавалися після валідації та з якої причини сценарій був продовжений, уточнений, відхилений або завершений. Таке журналювання не лише полегшує налагодження прототипу, а й створює основу для подальшого аналізу безпеки, аудиту рішень і виявлення аномальної поведінки. Для систем, у яких частина рішень формується на основі модельної інтерпретації, наявність такої трасувальної інформації є критично важливою умовою керованості.

Отже, механізми авторизації та безпеки в інтелектуальному шлюзі повинні будуватися як багаторівнева система контролю, що охоплює автентифікацію користувача, рольову авторизацію, модель дозволених інструментів, валідацію аргументів, ізоляцію контексту, захист від prompt injection, обмеження автономності оркестрації та журналювання всіх критично важливих рішень. Лише за таких умов природномовний інтерфейс може бути інтегрований у мікросервісну архітектуру без втрати керованості та безпеки. У межах запропонованого методу безпека не є зовнішнім доповненням до оркестрації, а виступає її невід’ємною складовою, яка визначає межі допустимого виконання дій і забезпечує контрольоване перетворення наміру користувача на сервісний виклик.

2.7 Візуальна специфікація процесу інтелектуальної маршрутизації

У підрозділах 2.1–2.6 було сформовано архітектурну та методичну основу інтелектуального шлюзу: визначено багаторівневу структуру, формалізовано семантичний розбір запиту й модель метаданих інструментів, описано агентську

оркестрацію, керування станом сесії, а також вимоги до продуктивності та безпеки. Для того щоб ці рішення сприймалися як єдиний керований процес, у даному підрозділі наведено діаграми, які відображають логіку роботи системи у вигляді чітких потоків даних і взаємодії компонентів. Візуалізація виконує роль проектної верифікації: вона показує, як природномовний запит перетворюється на структуровані інструментальні виклики в межах дозволеної множини інструментів T, проходить формальну перевірку параметрів і виконується з урахуванням контексту сесії та проміжних результатів.

На рисунку 2.3 наведено діаграму потоку даних, яка відображає повний цикл обробки природномовного запиту користувача в інтелектуальному шлюзі — від моменту надходження запиту до формування відповіді та, за необхідності, переходу до наступної ітерації оркестрації. На вході система отримує запит p разом із даними автентифікації τ. Після перевірки доступу формується дозволена множина інструментів T, яка надалі виступає обмеженням для планування та унеможливлює ініціювання дій поза наданими правами.

Далі виконується підготовка контексту сесії: із компонента стану сесії отримується контекст s, що включає релевантні факти попередньої взаємодії та проміжні результати виконаних кроків. На основі (p,s,T) здійснюється семантичний добір інструментів із використанням реєстру інструментів, у якому зберігаються описи операцій та їх метадані. Результатом цього етапу є набір кандидатів $K \subseteq T$, який обмежує простір вибору для наступного кроку.

Після формування набору кандидатів система формує структурований виклик інструмента c у вигляді пари tool_id + args. Принципово, що на цьому етапі відбувається не виконання, а побудова формалізованого подання дії, придатного до перевірки. Перед допуском до виконання виклик проходить контрольну точку валідації та допуску: перевіряється належність обраного інструмента до множини T і відповідність параметрів схемі інструмента (JSON Schema) та правилам допуску дій. У випадках недостатності даних або неоднозначності система не переходить до виконання, а формує уточнююче питання користувачу, після чого процес може бути повторений із доповненим запитом.

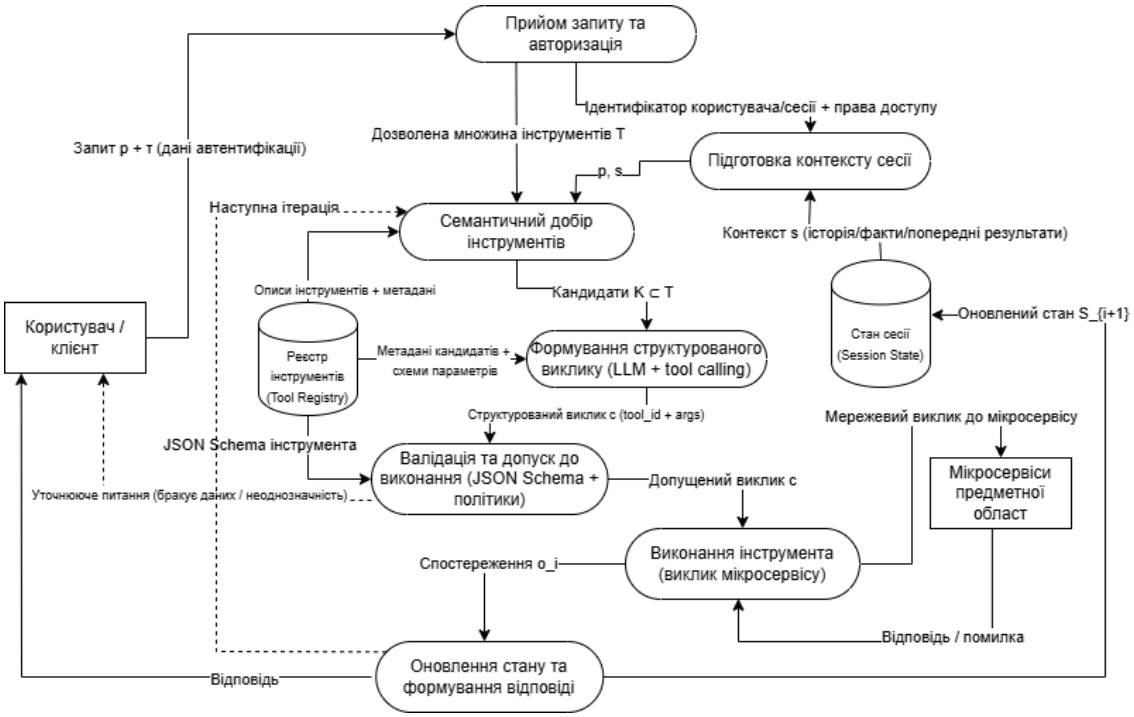

Рисунок 2.3 – Потік даних обробки запиту в інтелектуальному шлюзі

На рисунку 2.4 показано послідовність обробки запиту в детермінованому режимі інтелектуального шлюзу. У цьому сценарії клієнт надсилає структурований запит разом із даними автентифікації τ, після чого шлюз виконує перевірку автентифікації та авторизації, а також застосовує політики доступу. Якщо перевірка не пройдена, діаграма демонструє альтернативну гілку з поверненням помилки авторизації та завершенням обробки без звернення до внутрішніх сервісів.

У випадку успішної перевірки доступу шлюз визначає режим обробки як детермінований та перенаправляє запит до відповідного мікросервісу за наперед заданим маршрутом. Після отримання відповіді від мікросервісу шлюз формує підсумкову відповідь і повертає її клієнту. Таким чином, на рисунку 2.4 зафіксовано «короткий шлях» обробки запитів, у якому не залучаються семантичний добір інструментів, формування структурованих інструментальних викликів і керування контекстом сесії. Це узгоджується з положеннями підрозділів 2.1 і 2.5: інтелектуальний режим є надбудовою, яка застосовується лише для неструктурованих або сценарних запитів, тоді як типові технічні звернення можуть

оброблятися стандартними механізмами маршрутизації з прогнозованими показниками продуктивності.

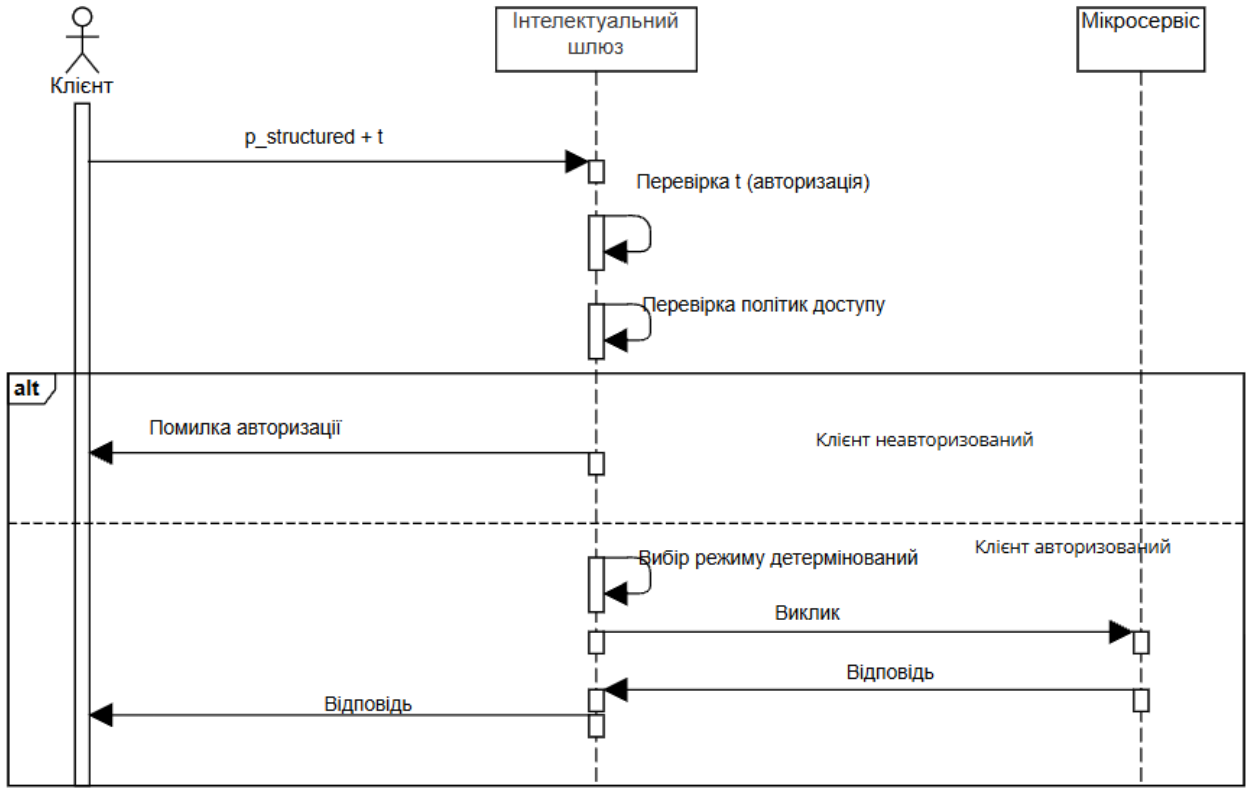

Рисунок 2.4 – Діаграма послідовності обробки запиту в детермінованому режимі

На рисунку 2.5 показано послідовність обробки природномовного запиту в інтелектуальному режимі інтелектуального шлюзу. Після надходження запиту p разом із даними автентифікації τ шлюз виконує перевірку доступу та формує дозволену множину інструментів T, яка надалі обмежує всі можливі дії в межах політик авторизації. Далі шлюз отримує контекст сесії s зі стану сесії та звертається до реєстру інструментів для отримання метаданих операцій і схем параметрів. На основі (p,s,T) виконується семантичний добір кандидатів $K \subseteq T$, після чого ці дані передаються LLM-оркестратору для формування структурованого виклику c у форматі tool_id + args.

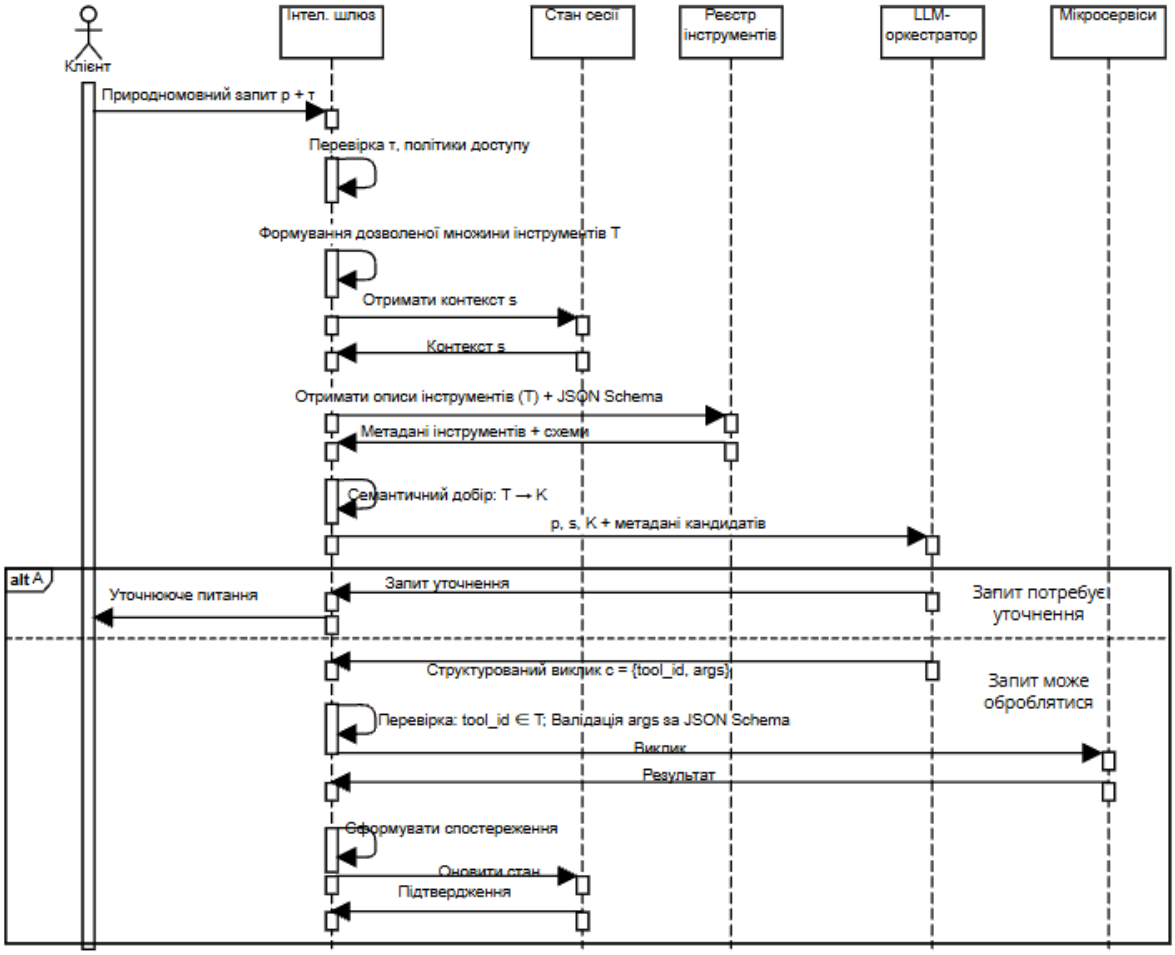

Рисунок 2.5 – Діаграма послідовності обробки запиту в інтелектуальному режимі

На діаграмі явно показано розгалуження для випадку недостатності даних: якщо сформувати коректний виклик неможливо або запит є неоднозначним, шлюз повертає користувачу уточнююче питання, не виконуючи жодної дії в мікросервісному середовищі. Якщо даних достатньо, сформований виклик проходить контроль допуску до виконання: перевіряється належність обраного інструмента до множини T та коректність параметрів відповідно до формального контракту інструмента (JSON Schema). Лише після успішної перевірки шлюз здійснює мережевий виклик до відповідного мікросервісу та отримує результат, який фіксується як спостереження o_i. Далі спостереження використовується для оновлення стану сесії та формування відповіді користувачу, а за потреби — для перехода до наступної ітерації планування у багатоетапному сценарії.

Керований життєвий цикл оркестрації сценарного запиту в інтелектуальному шлюзі у вигляді послідовності станів і переходів між ними показано на рисунку 2.6.

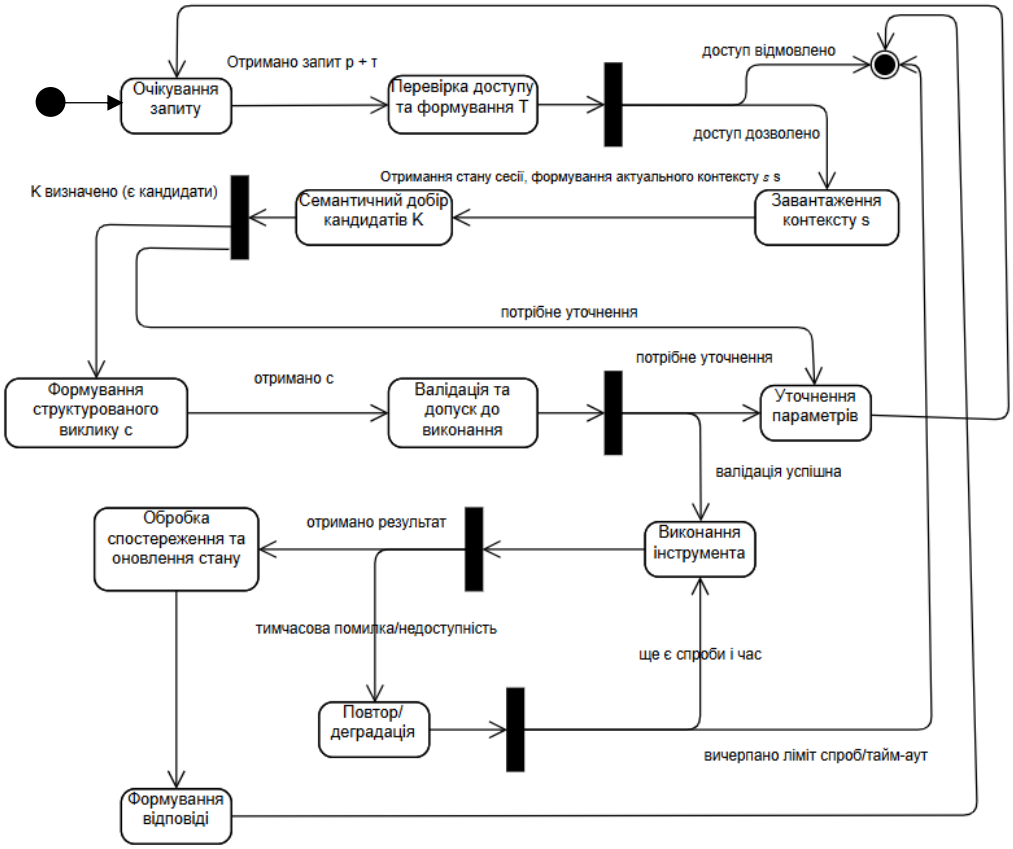

Рисунок 2.6 – Діаграма керованого процесу оркестрації

Процес стартує зі стану очікування запиту та переходить до перевірки доступу, де виконується автентифікація й формується дозволена множина інструментів T. Після отримання актуального контексту сесії s система виконує семантичний добір кандидатів K і формує структурований виклик c. Далі відбувається контроль допуску до виконання, який включає перевірку належності інструмента до T та валідацію параметрів за формальним контрактом (JSON Schema). Якщо даних недостатньо, процес переходить у стан уточнення параметрів і повертається до очікування наступного повідомлення користувача. У разі успішної валідації виконується інструментальний виклик до мікросервісу, після чого результат обробляється як спостереження o_i з оновленням стану сесії. Окремо відображено механізм деградації для тимчасових збоїв: повторні спроби виконання

допускаються лише в межах заданих обмежень за кількістю спроб і часом, після чого процес завершується з відповідним результатом.

3 ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЧНА БАЗА РЕАЛІЗАЦІЇ ІНТЕЛЕКТУАЛЬНОГО ШЛЮЗУ

3.1 Обґрунтування вибору платформи .NET та мови C#

Реалізація інтелектуального шлюзу для системи трекінгу замовлень висуває вимоги, характерні для високонавантажених прикладних компонентів: прогнозована затримка обробки запитів, надійна робота з мережевими протоколами, зручна побудова конвеєра обробки (middleware-підхід), керована інтеграція з зовнішніми сервісами та можливість стандартизовано реалізувати політики безпеки, валідації й логування. З огляду на ці вимоги, у роботі обрано платформу .NET (цільова версія .NET 8/9) та мову C# як технологічну основу прототипу інтелектуального шлюзу.

Платформа .NET забезпечує продуктивну та стабільну базу для побудови HTTP-сервісів завдяки ASP.NET Core та веб-серверу Kestrel, який є сервером за замовчуванням у шаблонах ASP.NET Core і підтримує кросплатформений запуск. Kestrel підтримує HTTPS, WebSockets, HTTP/2 та сценарії високопродуктивної роботи за зворотним проксі (зокрема із застосуванням Unix-sockets), що є релевантним для розгортання шлюзу в інфраструктурі з балансувальниками навантаження й контейнерними середовищами [29].

Окремою причиною вибору .NET є системна орієнтація платформи на підвищення продуктивності у нових версіях. Для ASP.NET Core 8 та .NET 9 Microsoft публікує окремі огляди оптимізацій продуктивності, що підтверджує пріоритетність цього напряму для платформи та є важливим для шлюзу, який обробляє значну кількість короткоживучих запитів і виконує додаткові кроки (семантичний добір, валідація, оркестрація) у інтелектуальному режимі.

Мова C# та екосистема .NET також дають практичні переваги для реалізації вимог, сформульованих у розділі 2. По-перше, сувора типізація й розвинені інструменти статичного аналізу полегшують реалізацію перевірюваних контрактів і інваріантів, що критично для контуру «формування структурованого виклику →

валідація → виконання». По-друге, в ASP.NET Core вбудовано механізм залежнісного впровадження (DI) та стандартний контейнер сервісів, що дозволяє уніфіковано конфігурувати та тестувати компоненти шлюзу: реєстр інструментів, модуль семантичного добору, клієнт LLM, модуль валідації, виконувальний контур та підсистеми спостережуваності [30].

Нарешті, кросплатформеність .NET спрощує відтворюваність середовища прототипу: шлюз може запускатися локально під час розробки та розгортатися у Linux-контейнерах у цільовій інфраструктурі без зміни коду, що є важливою умовою для інженерної перевірки продуктивності й стабільності в умовах, наближених до експлуатаційних [31].

У підсумку вибір .NET 8/9 і C# є обґрунтованим для задачі реалізації інтелектуального шлюзу: платформа забезпечує високопродуктивний кросплатформений веб-сервер, стандартизований конвеєр обробки запитів і вбудовані механізми DI, необхідні для модульної реалізації компонентів методу, описаного в розділі 2.

3.2 Інструменти інтеграції з великими мовними моделями в .NET: Microsoft.Extensions.AI та Semantic Kernel

Реалізація інтелектуального шлюзу, описаного в розділі 2, потребує двох взаємопов’язаних груп інструментів. Перша група відповідає за стандартизоване підключення до LLM-провайдерів та інфраструктурні можливості на кшталт керованого виклику інструментів, телеметрії, кешування й типового для .NET способу конфігурації через залежнісне впровадження. Друга група інструментів потрібна для побудови оркестрації: перетворення запиту p, контексту сесії s і множини дозволених інструментів T у послідовність керованих кроків з отриманням спостережень o_i, оновленням стану та переходами між кроками відповідно до правил допуску, валідації й уточнення. В екосистемі .NET ці потреби природно закривають два підходи: бібліотеки Microsoft.Extensions.AI як «базовий інфраструктурний шар» інтеграції generative AI у застосунки .NET, а також

Semantic Kernel як фреймворк, орієнтований на оркестрацію інструментів і агентні сценарії.

Microsoft.Extensions.AI — це набір бібліотек, які надають уніфікований рівень абстракцій для взаємодії з генеративними моделями та сервісами в типовому стилі .NET [32]. В офіційній документації Microsoft зазначено, що пакет дозволяє інтегрувати до застосунків такі компоненти, як автоматичне викликання інструментів, телеметрію та кешування, використовуючи знайомі розробникам патерни DI та middleware. У .NET-блозі Microsoft цей підхід описується як «foundational library» для generative AI у .NET, що переносить у домен AI ті самі практики, які вже стали стандартом для ASP.NET: конвеєри обробки, конфігурованість і замінність компонентів. Для задачі інтелектуального шлюзу це важливо, оскільки дозволяє реалізувати «LLM-контур» як керований сервіс застосунку: LLM-клієнт можна конфігурувати, обгортати middleware (наприклад, для логування чи обмеження поведінки), а також замінювати реалізацію провайдера без перепроєктування решти компонентів шлюзу.

Окремо варто підкреслити, що Microsoft.Extensions.AI містить практичні механізми для реалізації tool calling у прикладному коді. Зокрема, у репозиторії dotnet/extensions наведено приклади, де поверх IChatClient будуються обгортки з middleware-підходом і підключається автоматичне виконання функцій через UseFunctionInvocation(), а самі інструменти задаються як структуровані описи у ChatOptions.Tools [33]. Це безпосередньо узгоджується з вимогами у підрозділах 2.2 і 2.6: модель повертає не довільний текст, а структурований виклик tool_id + args, який далі проходить перевірки до моменту виконання. Офіційна документація Microsoft щодо tool calling підкреслює, що застосунок описує інструменти моделі як частину «розмови», модель може вибрати інструмент, після чого застосунок виконує його й повертає результат назад моделі для формування відповіді. У термінах розділу 2 цей набір можливостей відповідає «інфраструктурному контуру»: стандартизоване формування запитів до LLM, представлення інструментів і прийом структурованих результатів.

Semantic Kernel доповнює цей інфраструктурний рівень тим, що надає вищий рівень абстракції для оркестрації інструментів і моделювання агентних сценаріїв [34]. У документації Semantic Kernel «плагін» визначається як група функцій, які можуть бути «експоновані» AI-застосунку, і ці функції можуть бути оркестровані для виконання запитів користувача; при цьому Semantic Kernel підтримує автоматичне викликання функцій через механізми function calling. Окремо в екосистемі Semantic Kernel доступний Agent Framework, який позиціонується як платформа для створення AI-агентів та інтеграції agentic-патернів у застосунки, що відповідає логіці підрозділу 2.3 (керований цикл «планування–виконання–спостереження» з переходами між кроками). Для задачі інтелектуального шлюзу Semantic Kernel зручний тим, що дозволяє описувати інструменти як плагіни/функції та будувати поверх них процес оркестрації: визначати, які інструменти можуть бути використані в конкретному сценарії, як передавати контекст і як обробляти проміжні результати [35].

У межах даної роботи доцільно розглядати Microsoft.Extensions.AI та Semantic Kernel як взаємодоповнювальні інструменти, що відповідають різним рівням архітектури з розділу 2. Microsoft.Extensions.AI закриває базові потреби інтеграції generative AI у .NET-застосунок: уніфікований клієнтський інтерфейс, middleware-підхід, інструментальні виклики, телеметрію й стандартний спосіб підключення через DI. Semantic Kernel, зі свого боку, корисний як фреймворк для опису й виконання оркестрації: він надає поняття плагінів/функцій як інструментів, а також підходи до агентних сценаріїв, де рішення про наступний крок приймається з урахуванням контексту та результатів попередніх викликів [36]. Такий розподіл ролей узгоджується з методичними вимогами: LLM-частина має бути інтегрована в застосунок керовано, а логіка оркестрації повинна залишатися під контролем шлюзу через обмеження множини T, перевірювані контракти параметрів (JSON Schema), правила допуску дій та механізм уточнення. Саме тому в подальших підрозділах розділу 3 інструменти інтеграції з LLM розглядаються разом із засобами контрактної валідації та зберігання стану сесії, які забезпечують детермінованість і безпеку виконання інтелектуальних запитів.

3.3 Вибір провайдера великих мовних моделей та стратегії їх розгортання

У межах реалізації інтелектуального шлюзу (розділ 2) вибір провайдера LLM визначається не лише якістю генерації, а насамперед вимогами до керованості, безпеки, стабільності доступу та відтворюваності експериментів. Оскільки мовна модель вбудовується в контур «семантичний добір → формування структурованого виклику → валідація → виконання», критичними є можливість стандартизовано викликати модель через API, контролювати параметри запиту, отримувати структуровані відповіді для tool calling, а також інтегрувати виклики моделі в загальну систему спостережуваності й аудиту. З практичної точки зору доцільно розглядати три підходи: використання керованого хмарного сервісу корпоративного класу, використання прямого API провайдера моделей, а також локальне розгортання моделей у середовищі виконання прототипу.

Перший підхід ґрунтується на використанні Azure OpenAI, який надає доступ до моделей OpenAI в інфраструктурі Microsoft Azure та позиціонується як сервіс для побудови «enterprise-ready» рішень [37]. У документації підкреслюється сумісність API та можливість використання моделей OpenAI в контурі безпеки Azure, а також наводяться матеріали щодо політик безпеки й процедурних рекомендацій (security baseline) для Azure OpenAI [38]. Це важливо саме для шлюзу, оскільки модель бере участь у прийнятті рішення про інструментальний виклик і тому повинна бути включена в контрольований середовище експлуатації: з керуванням доступом, політиками, журналюванням і моніторингом. Крім того, для корпоративних сценаріїв суттєвими є питання обробки даних і конфіденційності, для яких Microsoft публікує окремі роз’яснення щодо data privacy та обробки даних в Azure-контурі [39]. У підсумку, використання Azure OpenAI є обґрунтованим для прототипу, який моделює реалістичні умови експлуатації системи трекінгу в організації, де важливі керованість, відповідність вимогам безпеки та інтеграція з інструментами спостережуваності.

Другий підхід передбачає використання OpenAI API як прямого каналу доступу до моделей через HTTP-інтерфейс [40]. Офіційна документація OpenAI

надає опис REST-викликів, режимів потокової відповіді та загальні принципи використання платформи. Для дослідницької частини роботи це дає перевагу простоти підключення та незалежності від конкретної хмарної платформи. Водночас, у межах розгортання прототипу як «інтелектуального шлюзу» важливо, щоб контур безпеки й управління доступом був узгоджений із загальними вимогами системи; тому вибір між Azure OpenAI та OpenAI API доцільно здійснювати за критеріями організаційної інфраструктури, політик безпеки, вимог до розміщення та обробки даних, а також доступних механізмів спостережуваності й управління [41]. У контексті даної роботи цей варіант може розглядатися як технічно придатний альтернативний канал інтеграції з моделями для прототипування та порівняння підходів, за умови дотримання тих самих проектних обмежень на рівні шлюзу (множина T, схематична валідація, правила допуску дій).

Третій підхід полягає у використанні локальних моделей (on-premise / local inference), коли інференс виконується в середовищі прототипу без викликів зовнішніх API. У .NET-екосистемі як приклад такого підходу може розглядатися бібліотека LLamaSharp, яка позиціонується як кросплатформена C#/.NET-обгортка для запуску моделей локально на CPU або GPU на основі llama.cpp [42]. Локальне розгортання потенційно зменшує залежність від зовнішнього сервісу та спрощує контроль середовища виконання, однак накладає вимоги до апаратних ресурсів і керування моделями (розмір, швидкодія, якість, оновлення) [43]. Тому для прототипу інтелектуального шлюзу локальні моделі доцільно трактувати як альтернативний режим розгортання, який може бути корисним у сценаріях, де критичними є автономність або вимоги до локальної обробки, але який потребує окремого інженерного обґрунтування з позиції продуктивності та стабільності.

Таким чином, стратегія вибору провайдера LLM у даній роботі визначається балансом між керованістю, вимогами безпеки, стабільністю доступу та відтворюваністю експериментів. Для прототипу, орієнтованого на реалістичне мікросервісне середовище, обґрунтованим є використання керованого сервісу корпоративного класу, при цьому прямий API-доступ і локальне розгортання

розглядаються як альтернативні варіанти, які можуть бути використані залежно від вимог інфраструктури та обмежень середовища.

3.4 Контракти інструментів і валідація параметрів: JSON Schema та бібліотеки .NET

Ключова вимога методу, сформульована у підрозділах 2.2 та 2.6, полягає в тому, що результат семантичного розбору не може залишатися довільним текстом: він має бути перетворений у структурований виклик інструмента tool_id + args, який підлягає машинній перевірці до моменту виконання. Для цього необхідно мати формальний опис параметрів кожного інструмента та механізм валідації, який відсікає некоректні або потенційно небезпечні виклики. У даній роботі як формат контрактів параметрів використовується JSON Schema, оскільки він стандартизовано описує структуру JSON-даних, типи, обов'язковість полів, обмеження значень, вкладені об'єкти та інші правила, що дозволяє будувати перевірюваний бар'єр між імовірнісним етапом планування та детермінованим етапом виконання.

У сучасній практиці найбільш актуальною є специфікація JSON Schema Draft 2020-12, яка визначає поточну версію стандарту на сайті проєкту JSON Schema і розділяє специфікацію на частини Core та Validation. Для прототипу інтелектуального шлюзу це важливо з двох причин. По-перше, наявність поля $schema дозволяє однозначно зафіксувати діалект, уникаючи неоднозначностей при інтерпретації схем різними валідаторами. По-друге, сама специфікація чітко формалізує «словник» валідації (validation vocabulary), який використовується для перевірки входів інструментів, тобто для встановлення того, що виклик відповідає контракту і може бути безпечно виконаний. У рамках розроблюваного підходу кожен інструмент у реєстрі описується метаданими, де inputSchema (або еквівалентне поле) містить JSON Schema для параметрів, а оркестратор формує args так, щоб вони відповідали цій схемі.

З погляду реалізації в .NET потрібні два рівні підтримки JSON-контрактів. Перший рівень — це базові засоби роботи з JSON для побудови/серіалізації структурованого виклику та для передачі даних між компонентами шлюзу. У .NET ці задачі закриває System.Text.Json, який забезпечує серіалізацію/десеріалізацію та роботу з JSON-поданням даних у продуктивний спосіб і природно інтегрується з ASP.NET Core. Другий рівень — це власне валідація JSON за схемою, для якої у .NET найчастіше застосовуються спеціалізовані бібліотеки, оскільки стандартна бібліотека System.Text.Json не є валідатором JSON Schema.

У практичній частині методу (валідаційний бар'єр у підрозділах 2.2 та 2.6) доцільно опиратися на одну з двох зрілих реалізацій JSON Schema для .NET:

— Json.NET Schema (Newtonsoft.Json.Schema) — окремий продукт/пакет екосистеми Newtonsoft, який позиціонується як повноцінний фреймворк JSON Schema для .NET і надає об'єктну модель схеми та механізми валідації. Його перевага для прототипу полягає у зрілості, широкому використанні та наявності документації/прикладів.

— NJsonSchema — бібліотека .NET для читання, генерації та валідації JSON Schema (draft v4+), яка також підтримує сценарії генерації схем та перевірки JSON-даних на відповідність схемі. Перевагою є зручність для роботи зі схемами як артефактами реєстру інструментів і можливість використовувати її як незалежний валідатор.

В контексті інтелектуального шлюзу обидва варіанти можуть бути використані як реалізація одного й того ж принципу: після того як LLM сформувала c = {tool_id, args}, шлюз виконує детерміновану перевірку args за JSON Schema інструмента. Якщо валідація не проходить, виклик не виконується, а система переходить до режиму уточнення або відхиляє дію згідно з політикою (наприклад, якщо виклик суперечить контракту або виходить за межі дозволеної множини T). Таким чином, JSON Schema виступає не просто «документацією параметрів», а формальним механізмом забезпечення коректності та безпеки, який відокремлює етап планування (ймовірнісний) від етапу виконання (детермінований) і робить поведінку системи відтворюваною в тестуванні.

3.5 Спостережуваність, журналювання та метрики оцінювання роботи інтелектуального шлюзу

Оскільки інтелектуальний шлюз поєднує детермінований контур обробки запитів із когнітивним контуром (семантичний добір, формування інструментального виклику, валідація, виконання), критично важливо забезпечити спостережуваність системи. У межах даної роботи спостережуваність розглядається як сукупність механізмів журналювання, трасування та метрик, які дозволяють, по-перше, пояснити і відтворити послідовність інструментальних кроків a_i , по-друге, контролювати якість і надійність виконання (успіх/помилки/повтори), і, по-третє, отримати кількісні показники для експериментальної оцінки методу в розділі 4. У контексті LLM-інтеграції це особливо важливо, оскільки поведінка когнітивного контуру є імовірнісною, а отже потребує прозорого контролю на рівні детермінованих артефактів: які інструменти обрано, які параметри сформовано, чи пройшли вони валідацію, який результат повернув сервіс і як оновився стан сесії.

Базовим елементом спостережуваності є структуроване журналювання подій оркестрації. На рівні шлюзу доцільно фіксувати: ідентифікатор запиту/кореляції, обраний режим (детермінований або інтелектуальний), множину кандидатів K (у скороченому вигляді), обраний інструмент tool_id, параметри args після валідації, факт проходження перевірок (належність до T, відповідність JSON Schema), результат виконання інструмента (успіх/помилка) та час виконання кроку. Окремо слід журналювати переходи в режим уточнення, оскільки частота уточнень є прямим індикатором якості семантичного розбору та достатності контексту. Такий журнал потрібен не лише для налагодження, а й для аудиту та відтворюваності сценаріїв [44].

Другим компонентом є трасування запитів (distributed tracing), яке дозволяє зібрати в один «ланцюжок» події шлюзу та виклики мікросервісів. Для мікросервісних систем це важливо, оскільки один сценарний запит може породжувати кілька звернень до різних сервісів, а затримка користувацької

відповіді визначається сумою латентностей окремих етапів. Використання кореляційного ідентифікатора і стандартних підходів до трасування дозволяє локалізувати «вузьке місце»: чи затримка виникає на етапі звернення до LLM, на етапі валідації/пост-обробки, чи в мережевих викликах до внутрішніх сервісів.

Третій компонент — метрики, які формалізують ефективність методу та слугують основою для розділу 4. З огляду на поставлену задачу доцільно збирати метрики трьох груп. Перша група характеризує продуктивність: загальна latency запиту, розподіл часу за етапами (підготовка контексту, семантичний добір, LLM-виклик, валідація, виконання інструментів), а також кількість кроків сценарію. Друга група характеризує надійність: частка успішних сценаріїв, частота помилок валідації (некоректні параметри), частота помилок виконання інструментів, кількість повторних спроб і частка сценаріїв, які завершилися деградацією. Третя група характеризує «якість взаємодії»: частка запитів, що потребували уточнення, середня кількість уточнень на сценарій, а також частка випадків, коли система завершила сценарій без виконання дії через доменні обмеження або політики допуску. Якщо у прототипі доступні дані про витрати токенів для LLM-викликів, їх також доцільно враховувати як окрему метрику «вартість когнітивного контуру», оскільки вона напряму впливає на масштабованість і вартість експлуатації.

У підсумку, підсистема спостережуваності для інтелектуального шлюзу повинна забезпечити прозорість усього контуру «семантичний розбір → структурований виклик → валідація → виконання → оновлення стану» та надати достатні кількісні дані для експериментальної оцінки. Саме тому на рівні прототипу доцільно фіксувати структуровані події оркестрації, забезпечувати трасування між компонентами та збирати метрики, що відображають продуктивність, надійність і частоту уточнень. Це безпосередньо підготує основу для розділу 4, де буде виконано оцінювання запропонованого методу на типових сценаріях системи трекінгу замовлень.

4 РЕАЛІЗАЦІЯ ПРОТОТИПУ ТА ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА МЕТОДУ

4.1 Архітектура прототипу та реалізовані компоненти

Для перевірки запропонованого підходу було реалізовано прототип інтелектуального шлюзу у вигляді окремого програмного рішення. Прототип призначений для приймання природномовних запитів користувача, визначення допустимих інструментів, перевірки параметрів виклику, звернення до мікросервісів та формування підсумкової відповіді разом із трасувальною інформацією. Загальна побудова рішення відповідає ідеї інтелектуального API Gateway, у якому інтерпретація запиту відокремлена від фактичного виконання сервісних дій. Весь вихідний код знаходиться в Додатку Б.

Архітектура реалізованого прототипу побудована таким чином, щоб відобразити основні етапи запропонованого методу інтелектуальної маршрутизації, розглянутого в розділі 2. Зокрема, у структурі рішення окремо виділено рівень приймання запиту, рівень прикладної оркестрації, підсистему керування станом сесії, механізми політик доступу та валідації параметрів, а також рівень виконання інструментальних викликів. Це дозволяє не лише реалізувати працездатний MVP-прототип, а й простежити відповідність між концептуальною моделлю методу та її програмною реалізацією. Такий підхід дозволяє простежити зв’язок між основними етапами запропонованого методу інтелектуальної маршрутизації та їх програмною реалізацією в межах прототипу.

Структурно прототип складається з п’яти проєктів. Проєкт IntelligentGateway.Api реалізує HTTP-рівень взаємодії із зовнішніми клієнтами та містить точки входу до системи. Проєкт IntelligentGateway.Application відповідає за прикладну логіку обробки користувацьких запитів. У проєкті IntelligentGateway.Domain визначено основні моделі, що описують інструменти, стан сесії, результати виконання та структуру відповіді. Проєкт IntelligentGateway.Infrastructure містить технічні реалізації реєстру інструментів,

політик доступу, валідаторів аргументів, адаптерів до сервісів та сховищ сесій. Проєкт IntelligentGateway.Tests використовується для інтеграційної перевірки ключових сценаріїв роботи прототипу.

Узагальнену взаємодію основних компонентів реалізованого прототипу наведено на рисунку 4.1.

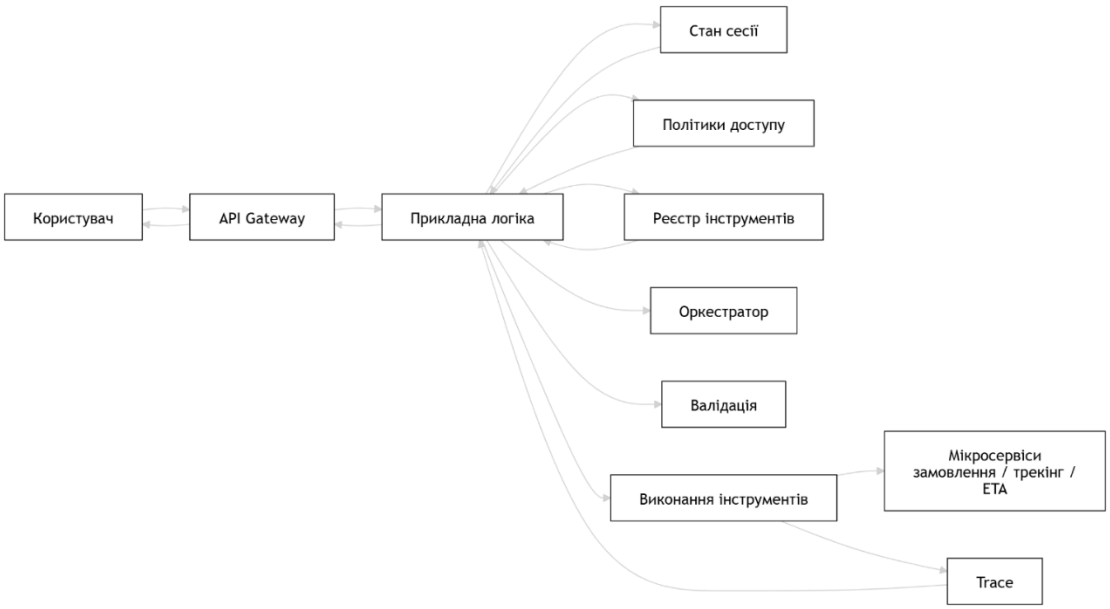

Рисунок 4.1 – Архітектура прототипу інтелектуального шлюзу

Як видно зі схеми, прикладний рівень прототипу виконує центральну координаційну роль: після отримання природномовного запиту він звертається до сховища стану сесії, визначає набір допустимих інструментів, формує план дій, перевіряє коректність аргументів і лише після цього ініціює виконання сервісного виклику. Така організація відповідає принципу контрольованого виконання, у якому інтерпретація користувацького наміру відокремлена від фактичного доступу до внутрішніх сервісів.

Зовнішня взаємодія з прототипом організована через набір HTTP endpoint'ів. Основною точкою входу є POST /api/query, через яку клієнт передає ідентифікатор сесії, ідентифікатор користувача та текст запиту природною мовою. Додатково реалізовано endpoint GET /api/tools для отримання переліку доступних інструментів, GET /api/sessions/{sessionId} (лістинг 4.1) для перегляду поточного

стану конкретної сесії та GET /health для перевірки працездатності сервісу. Така організація дозволяє використовувати шлюз як єдину точку доступу до функціональності мікросервісної системи та водночас забезпечує зручність тестування і налагодження.

Лістинг 4.1 – Кінцева точка для перегляду поточного стану конкретної сесії

```
app.MapGet("/api/sessions/{sessionId}",
           async (string sessionId, ISessionStore sessionStore,
CancellationToken cancellationToken) =>
{
    var session = await sessionStore.GetAsync(sessionId,
cancellationToken);
    return session is null ? Results.NotFound() : Results.Ok(session);
});
```

Ключовим архітектурним елементом прототипу є механізм роботи з інструментами. У системі використовується централізований реєстр інструментів, у якому кожна операція описується уніфіковано: через ідентифікатор, назву, текстовий опис, набір policy tags, схему вхідних параметрів у форматі JSON Schema, опис вихідних даних, а також технічні параметри виконання (лістинг 4.2). Такий підхід дозволяє розглядати звернення до сервісів не як довільні виклики endpoint’ів, а як контрольовані інструменти, придатні до перевірки перед виконанням. У реалізованому прототипі для демонстрації підтримуються інструменти отримання замовлення за ідентифікатором, пошуку останнього замовлення користувача, отримання статусу трекінгу та оцінки орієнтовного часу доставки.

Лістинг 4.2 – Приклад опису інструмента в реєстрі з JSON Schema для валідації вхідних параметрів

```
new ToolDefinition(
   "get_order_by_id",
   "Get order by id",
   "Retrieves order details by orderId.",
   ["orders.read"],
```

```
JsonDocument.Parse("""
  {
    "type":"object",
    "required":["orderId"],
    "properties":{
    "orderId": { "type":"string", "minLength":1 }
   }
  }
"""),
"Order summary containing id, customerId, status and createdAt.",
"order-service",
TimeSpan.FromSeconds(5),
1)
```

Центральна логіка обробки користувацького запиту зосереджена в компоненті UserQueryHandler. Після надходження запиту цей компонент завантажує або створює сесію, додає повідомлення користувача до короткотривалого контексту, отримує з реєстру перелік доступних інструментів і передає його до модуля політик доступу. Далі виконується побудова плану дії: система може сформувати уточнювальне запитання, безпосередню фінальну відповідь або структурований виклик певного інструмента. Якщо обрано інструмент, його аргументи перевіряються на відповідність JSON Schema, після чого здійснюється виклик відповідного адаптера мікросервісу. Результат виконання додається до сесії та може використовуватися на наступних кроках обробки. У прототипі реалізовано обмежений цикл виконання до трьох ітерацій, що забезпечує керованість, відтворюваність і прозорість поведінки системи.

Окремим функціональним блоком прототипу є підсистема керування станом сесії. У доменній моделі SessionContext зберігаються ідентифікатор сесії, ідентифікатор користувача, нещодавні повідомлення, результати викликів інструментів та витягнуті сутності, зокрема orderId. Це дозволяє підтримувати багатоетапні сценарії, у яких результат попереднього кроку використовується на наступному. Завдяки такому підходу шлюз може послідовно виконувати зв’язані

дії, наприклад спочатку визначити останнє замовлення користувача, а потім автоматично отримати для нього актуальний статус трекінгу.

Для забезпечення контрольованості та подальшого аналізу в прототипі реалізовано засоби трасування і базового метрикованого моніторингу. Під час обробки запиту формується кореляційний ідентифікатор, а у відповідь повертається об’єкт trace, що містить тривалість виконання, перелік кроків, кількість викликів інструментів, число помилок валідації та кількість уточнювальних взаємодій. На рівні API також реалізовано фіксацію загальної тривалості обробки запиту та інших базових метрик.

Для наочності в таблиці 4.1 подано відповідність між основними елементами запропонованого методу інтелектуальної маршрутизації, компонентами реалізованого прототипу та аспектами їх подальшої експериментальної перевірки.

Таблиця 4.1 – Відповідність елементів методу компонентам прототипу

Елемент методу	Компонент прототипу	Артефакт реалізації	Що перевіряється
Приймання природномовного запиту	API-рівень	`POST /api/query`	Коректність приймання запиту та повернення відповіді
Оркестрація та планування дій	Прикладний рівень	`UserQueryHandler`, orchestrator	Single-step, multi-step, clarification
Формалізація інструментів	Реєстр інструментів	`ToolDefinition`, `InMemoryToolRegistry`	Коректність вибору інструмента та опису контракту
Валідація параметрів	Валідатор аргументів	JSON Schema validation	Відхилення некоректних аргументів
Політики доступу	Сервіс політик	allowlist / policy filtering	Відмова в недозволених викликах
Стан сесії та пам’ять	Session store	`SessionContext`, `ISessionStore`	Follow-up сценарії, повторне використання контексту
Виконання інструментів	Виконавчий рівень	tool executor / adapters	Успішне виконання, деградація
Трасування виконання	Trace subsystem	`ExecutionTrace`	Кількість кроків, tool calls, validation failures, clarifications

Водночас реалізований прототип має характер MVP, орієнтованого на перевірку працездатності базових механізмів оркестрації, валідації, політик доступу та керування контекстом, а не на повномасштабне відтворення всіх можливих сценаріїв виробничої експлуатації.

Таким чином, реалізована архітектура прототипу поєднує зовнішній API-рівень, прикладну логіку оркестрації, доменні моделі, реєстр інструментів, політики доступу, механізми валідації параметрів та збереження контексту сесії. У результаті шлюз виконує роль окремого координаційного шару, який пов’язує природномовний користувацький запит із послідовністю формалізованих сервісних дій у мікросервісній системі.

4.2 Алгоритм роботи прототипу інтелектуального шлюзу

Далі розглянуто порядок обробки користувацького запиту в межах реалізованого прототипу інтелектуального шлюзу. Послідовність роботи системи охоплює використання контексту сесії, вибір допустимих інструментів, перевірку параметрів виклику та формування підсумкової відповіді.

Центральна логіка обробки користувацького запиту зосереджена в компоненті UserQueryHandler. Після надходження запиту система завантажує або створює сесію, додає повідомлення користувача до короткотривалого контексту, отримує з реєстру перелік доступних інструментів і передає його до модуля політик доступу. Після цього формується план дії, який залежно від змісту запиту та поточного контексту може привести до уточнення відсутніх даних, безпосереднього формування фінальної відповіді або вибору інструмента для подальшого виконання.

У випадку, коли для виконання сценарію обирається інструмент, система перевіряє допустимість цього виклику в межах визначених політик доступу, а також виконує валідацію аргументів на відповідність формальному опису. Лише після успішного проходження цих перевірок виконується звернення до відповідного адаптера сервісу. Отриманий результат зберігається в контексті сесії

та може бути використаний на наступних кроках обробки. Така організація дозволяє підтримувати не лише одноетапні, а й багатоетапні сценарії взаємодії.

Послідовність роботи прототипу при обробці користувацького запиту:

1) отримати природномовний запит користувача;
2) завантажити або створити сесію;
3) додати поточне повідомлення до контексту сесії;
4) отримати перелік доступних інструментів;
5) відібрати інструменти, дозволені для поточного користувача;
6) сформувати план дії на основі запиту та контексту;
7) якщо потрібне уточнення, сформувати уточнювальну відповідь;
8) якщо обрано інструмент, перевірити коректність його аргументів;
9) виконати інструментальний виклик;
10) оновити контекст сесії результатами виконання;
11) додати трасувальну інформацію;
12) повернути фінальну відповідь користувачу.

Для забезпечення керованості та передбачуваності поведінки системи в прототипі реалізовано обмежений цикл обробки до трьох ітерацій. Це означає, що в межах одного запиту шлюз може послідовно виконати кілька кроків планування і виклику інструментів, але лише в заздалегідь визначених межах. Такий підхід дозволяє уникнути неконтрольованих переходів між станами, спрощує аналіз роботи прототипу та підвищує відтворюваність результатів. Приклад реалізації керованого циклу обробки запиту в прикладному рівні прототипу наведено в лістингу 4.3.

Лістинг 4.3 – Фрагмент реалізації циклу обробки запиту в UserQueryHandler

```
for (var iteration = 0; iteration < 3; iteration++)
{
    var plan = orchestrator.BuildPlan(request, session, allowedTools);

    if (plan.RequiresClarification)
        return CreateClarificationResponse(plan);
```

```
    if (plan.IsFinal)
        return CreateFinalResponse(plan);

    validator.Validate(plan.ToolName, plan.Arguments);
    var result = await executor.ExecuteAsync(plan.ToolName, plan.Arguments,
cancellationToken);

    session.Apply(result);
}
```

Окрему роль у роботі прототипу відіграє контекст сесії. У ньому зберігаються ідентифікатор сесії, ідентифікатор користувача, нещодавні повідомлення, результати попередніх викликів інструментів та витягнуті сутності. Завдяки цьому результат одного кроку може бути використаний на наступному, що особливо важливо для follow-up сценаріїв, коли наступний запит користувача спирається на вже визначені дані, наприклад на знайдений ідентифікатор замовлення.

Для подальшого аналізу роботи шлюзу в прототипі реалізовано механізм трасування виконання. Під час обробки запиту накопичуються дані про тривалість роботи, кількість кроків, число викликів інструментів, кількість уточнювальних взаємодій і помилок валідації. У результаті користувачу повертається не лише відповідь, а й супровідна трасувальна інформація, що дозволяє оцінювати перебіг сценарію та використовувати ці дані для подальшої експериментальної перевірки.

Таким чином, у межах прототипу алгоритм обробки користувацького запиту реалізовано як послідовність контрольованих кроків, що охоплюють завантаження контексту сесії, відбір дозволених інструментів, формування плану дії, перевірку аргументів, виконання інструментального виклику та формування підсумкової відповіді з трасувальною інформацією. Така організація забезпечує керованість роботи прототипу і створює основу для подальшої експериментальної перевірки основних сценаріїв його функціонування.

4.3 Тестові сценарії та методика перевірки прототипу

Метою перевірки реалізованого прототипу є оцінка працездатності основних механізмів запропонованого методу інтелектуальної маршрутизації в межах керованих інтеграційних сценаріїв. На відміну від функціонального тестування окремих компонентів, у цьому підрозділі увага зосереджується на перевірці повного ланцюжка обробки запиту: від надходження природномовного звернення до формування плану дій, виконання інструментального виклику, оновлення сесійного контексту та повернення підсумкової відповіді з трасувальною інформацією. Такий підхід дозволяє оцінити не лише коректність окремих операцій, а й узгодженість роботи прототипу як цілісного координаційного шару мікросервісної системи.

Для перевірки було сформовано набір типових користувацьких інтенцій, що охоплюють як коректні сценарії звернення до системи, так і випадки, у яких виконання повинно бути відхилене або обмежене. До базового набору сценаріїв доцільно віднести: отримання інформації про замовлення за ідентифікатором, визначення статусу останнього замовлення користувача, оцінку орієнтовного часу доставки, уточнення відсутніх параметрів, відхилення недозволеного інструментального виклику, відхилення виклику через некоректні аргументи, а також сценарій контрольованої деградації у випадку помилки зовнішнього сервісу. Окремо до перевірки включено follow-up сценарії, у яких повторний запит у межах тієї самої сесії використовує контекст, сформований на попередньому кроці.

У межах експериментальної перевірки для кожного сценарію встановлюються критерії успішності. По-перше, система повинна коректно визначити тип дії, тобто сформувати правильний варіант плану: уточнення, завершення без виклику інструмента або допустимий інструментальний виклик. По-друге, у випадку інструментального виконання аргументи повинні відповідати формальному опису інструмента та пройти валідацію. По-третє, у разі наявності політик доступу виклик повинен бути або дозволений, або відхилений відповідно до встановлених правил. По-четверте, у багатоетапних сценаріях має

забезпечуватися коректне оновлення і повторне використання сесійного контексту. По-п’яте, результат сценарію повинен супроводжуватися трасувальною інформацією, достатньою для аналізу перебігу виконання.

Для оцінки роботи прототипу доцільно використовувати набір базових метрик, які характеризують коректність, керованість і продуктивність виконання. До першої групи належать метрики правильності: частка сценаріїв, у яких система обрала очікуваний інструмент або коректний тип завершення; частка викликів, що успішно пройшли валідацію аргументів; а також частка сценаріїв, у яких політики доступу були застосовані належним чином. До другої групи належать метрики взаємодії: кількість уточнювальних звернень, середня кількість кроків сценарію та частка запитів, у яких було використано контекст попередньої сесійної взаємодії. До третьої групи належать метрики продуктивності, зокрема загальна тривалість обробки запиту та розподіл часу за окремими сценаріями. Такий вибір метрик узгоджується з теоретичними положеннями, закладеними в розділі 2, і з наявною трасувальною інформацією, яку повертає прототип.

У практичній реалізації перевірка здійснюється на основі інтеграційних сценаріїв, які моделюють типовий користувацький запит до шлюзу та дозволяють спостерігати його повне проходження через основні етапи прикладної логіки. Для кожного сценарію фіксуються очікуваний ланцюжок дій, фактичний результат виконання, кількість ітерацій, обрані інструменти, факт уточнення, наявність помилок валідації та тип завершення сценарію. Саме така організація перевірки дозволяє трактувати її не лише як демонстрацію працездатності окремих endpoint’ів, а як оцінку поведінки прототипу в межах заданого набору користувацьких інтенцій.

Для систематизації перевірки в таблиці 4.2 наведено основні сценарії, використані для оцінювання роботи прототипу інтелектуального шлюзу. Вони охоплюють як коректні одноетапні та багатоетапні звернення, так і випадки, у яких система повинна виконати уточнення, відхилити недозволену дію, зафіксувати помилку валідації або коректно завершити сценарій в умовах контрольованої деградації.

Таблиця 4.2 – Основні сценарії перевірки прототипу

Позначення	Назва сценарію	Очікувана поведінка системи
S1	Отримання інформації про замовлення за заданим ідентифікатором	Коректне формування інструментального виклику та повернення результату
S2	Визначення статусу останнього замовлення користувача	Багатоетапне виконання з використанням проміжного результату
S3	Уточнення відсутніх параметрів користувацького запиту	Формування уточнювального повідомлення без виконання інструмента
S4	Відхилення недозволеного інструментального виклику	Блокування дії відповідно до політик доступу
S5	Виявлення некоректних аргументів інструмента	Відмова у виконанні після етапу валідації
S6	Контрольоване завершення сценарію в умовах помилки зовнішнього сервісу	Коректна обробка збою без порушення загальної логіки роботи
S7	Повторне використання контексту в межах однієї сесії	Використання збережених даних попереднього кроку під час follow-up запиту

Наведені сценарії формують базовий набір перевірки, достатній для оцінювання працездатності основних механізмів прототипу: оркестрації, валідації параметрів, застосування політик доступу, використання сесійного контексту та трасування виконання.

Таким чином, методика перевірки прототипу ґрунтується на наборі інтеграційних сценаріїв, які охоплюють одноетапні та багатоетапні звернення, уточнювальні взаємодії, перевірку політик доступу, валідацію аргументів, контрольовану деградацію та повторне використання сесійного контексту.

4.4 Результати перевірки та аналіз роботи прототипу

Результати перевірки реалізованого прототипу доцільно аналізувати з погляду того, наскільки коректно система реалізує основні механізми запропонованого методу інтелектуальної маршрутизації. До таких механізмів належать формування плану дії на основі природномовного запиту, перевірка допустимості інструментального виклику, валідація аргументів, підтримка багатоетапної взаємодії в межах сесії та формування трасувальної інформації для подальшого аналізу. У межах дослідження результати оцінювалися на основі

інтеграційних сценаріїв, визначених у підрозділі 4.3, що дозволило простежити поведінку прототипу не лише на рівні кінцевої відповіді, а й на рівні внутрішнього ланцюжка виконання.

Для систематизації результатів у таблиці 4.3 наведено узагальнені дані щодо виконання основних сценаріїв перевірки. Для кожного сценарію зафіксовано очікуваний ланцюжок дій, фактичну поведінку системи, кількість ітерацій, задіяні інструменти та тип завершення сценарію. Такий формат подання дозволяє порівняти теоретично очікуваний перебіг сценарію з реальною поведінкою прототипу, відображеною в трасувальних даних.

Таблиця 4.3 – Результати виконання основних сценаріїв перевірки

Сценарій	Очікуваний результат	Фактичний результат	Ітерації	Інструменти	Тип завершення
S1	Коректний виклик інструмента отримання замовлення за ідентифікатором	Сценарій завершено успішно, результат повернуто	1	`get_order_by_id`	Success
S2	Багатоетапне визначення статусу останнього замовлення	Сценарій завершено успішно з використанням проміжного результату	2	`get_latest_user_order` → `get_tracking_status`	Success
S3	Формування уточнювального повідомлення за відсутності параметрів	Система сформувала уточнення без виклику інструмента	1	—	Clarification Needed
S4	Відхилення недозволеного виклику	Виклик відхилено відповідно до політик доступу	1	—	Forbidden
S5	Відхилення некоректних аргументів	Виклик зупинено на етапі валідації	1	—	Invalid Arguments
S6	Контрольована деградація при помилці сервісу	Помилку оброблено без порушення загального сценарію	1–2	залежно від сценарію	Degraded Fallback
S7	Повторне використання контексту в межах сесії	Повторний запит використав збережені дані попереднього кроку	2	`get_latest_user_order` → `get_delivery_eta` або `get_tracking_status`	Success

З наведених результатів видно, що прототип коректно обробляє як одноетапні, так і багатоетапні сценарії. Для одноетапних звернень характерною є одна ітерація з прямим переходом до інструментального виклику або до уточнення, тоді як багатоетапні сценарії передбачають збереження проміжного результату та його подальше використання в межах тієї самої сесії. Це підтверджує узгодженість між оркестрацією запиту, інструментальним виконанням і використанням контексту сесії. Також результати демонструють, що прототип розрізняє ситуації успішного виконання, уточнення, політичного відхилення, валідаційної відмови та контрольованої деградації, що важливо для забезпечення керованості системи.

Для наочнішого подання результатів перевірки доцільно окремо розглянути розподіл базових сценаріїв за типом завершення (рис. 4.2). Такий підхід дає змогу оцінити не лише кількість успішно виконаних звернень, а й здатність системи коректно формувати уточнення, відхиляти недопустимі дії, фіксувати помилки валідації та завершувати сценарії в режимі контрольованої деградації

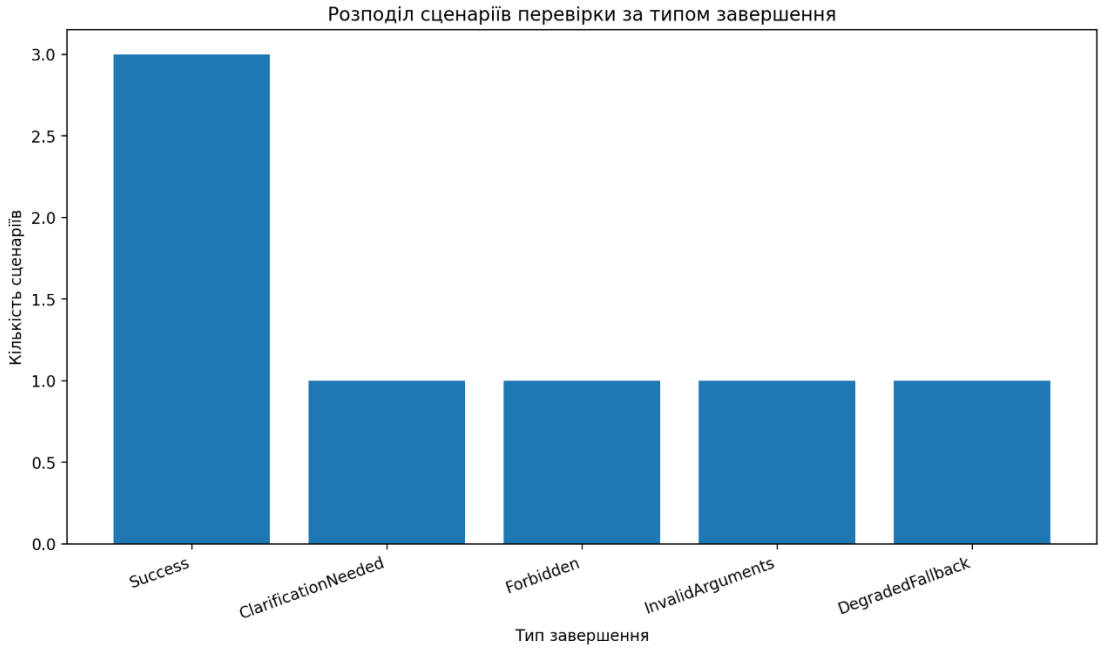

Рисунок 4.2 – Графік розподілу сценаріїв перевірки за типом завершення

Як видно з рисунка 4.2, найбільшу частку становлять сценарії, що завершилися успішним виконанням. Водночас у межах перевірки зафіксовано

також окремі випадки уточнення, відхилення за політиками доступу, валідаційної відмови та контрольованої деградації. Це свідчить про те, що реалізований прототип забезпечує не лише успішне виконання допустимих запитів, а й кероване завершення сценаріїв у ситуаціях, коли запит є неповним, недозволеним або містить некоректні параметри.

Для узагальнення результатів доцільно також розглянути агреговані показники, отримані на основі трасувальної інформації, що формується під час виконання сценаріїв. У поточній реалізації trace містить щонайменше тривалість обробки запиту, кількість кроків сценарію, число викликів інструментів, кількість уточнювальних взаємодій та число помилок валідації. Саме ці показники дозволяють перейти від якісного опису сценаріїв до більш структурованого аналізу роботи прототипу.

Таблиця 4.4 – Узагальнені показники перевірки прототипу

Показник	Характеристика результату
Частка сценаріїв із коректним типом завершення	В усіх базових сценаріях зафіксовано очікуваний тип завершення
Частка сценаріїв з успішним інструментальним виконанням	Усі допустимі сценарії з інструментальним викликом завершено коректно
Частка сценаріїв з уточненням	У сценаріях неповного запиту система формувала уточнення без передчасного виклику інструмента
Частка політичних відхилень	Недозволені виклики блокувалися до етапу виконання
Частка валідаційних відмов	Некоректні аргументи виявлялися до звернення до адаптера сервісу
Підтримка follow-up сценаріїв	Повторні звернення в межах сесії використовували збережений контекст
Трасувальна прозорість	Для кожного сценарію формувалися дані про кроки, виклики, уточнення та валідацію

Аналіз узагальнених показників дає підстави стверджувати, що реалізований прототип забезпечує працездатність базового контрольного контуру інтелектуального шлюзу. Зокрема, система коректно виконує відбір допустимих інструментів, перевіряє аргументи перед викликом, підтримує сценарії з

уточненням, використовує сесійний контекст у follow-up взаємодії та накопичує трасувальну інформацію, придатну для подальшого аналізу. Це означає, що запропонований підхід є практично придатним як архітектурна основа для побудови інтелектуального координаційного шару в мікросервісному середовищі.

Окрім типу завершення сценаріїв, важливо проаналізувати й особливості їх виконання. Для цього доцільно розглянути кількість ітерацій і кількість викликів інструментів у межах кожного базового сценарію перевірки. Такі показники дозволяють наочно відобразити різницю між одноетапними зверненнями, багатоетапними сценаріями та сценаріями, що використовують сесійний контекст.

Як видно з рисунка 4.3, одноетапні сценарії S1, S3, S4, S5 і S6 виконуються в межах однієї ітерації, тоді як сценарії S2 і S7 мають багатоетапний характер. У сценарії S2 багатоетапність зумовлена необхідністю послідовного використання кількох інструментів, а в сценарії S7 — повторним використанням контексту попереднього звернення в межах однієї сесії. Це підтверджує працездатність механізмів оркестрації та керування сесійним контекстом у реалізованому прототипі.

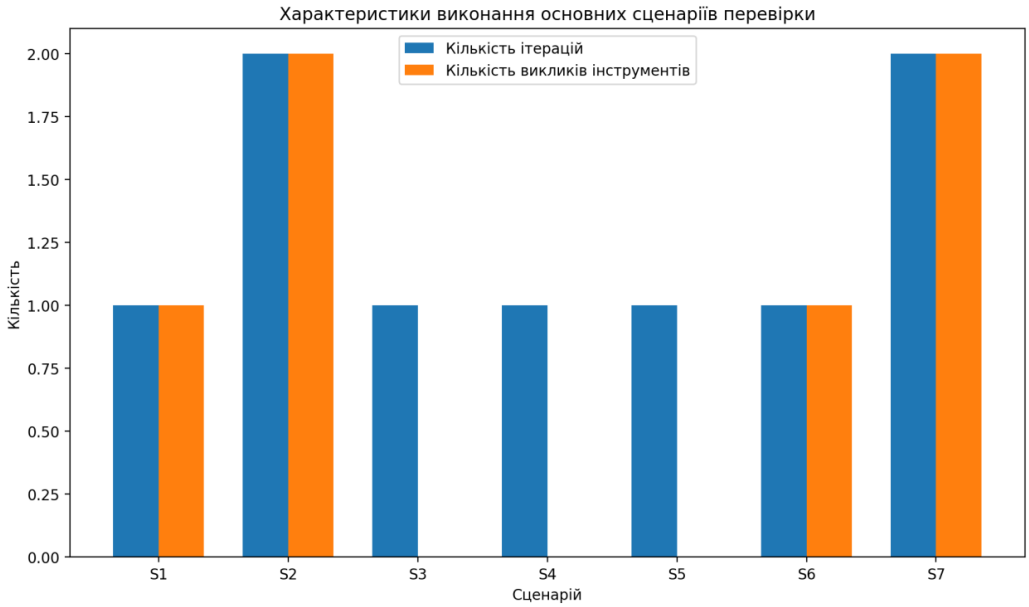

Рисунок 4.3 – Графік кількості ітерацій і викликів інструментів у базових сценаріях перевірки

Водночас отримані результати слід інтерпретувати з урахуванням поточних обмежень MVP-прототипу. Частина сценаріїв спирається на контрольовані або синтетичні умови виконання, а окремі інтеграції можуть використовувати placeholder- чи mock-реалізації. Крім того, проведена перевірка має передусім інтеграційно-функціональний характер і не охоплює повномасштабного статистичного чи навантажувального оцінювання. Тому результати цього підрозділу доцільно розглядати як підтвердження працездатності архітектури, оркестрації, валідації, політик доступу та механізму керування контекстом, а не як остаточне кількісне доведення ефективності промислового рішення.

Таким чином, результати перевірки показують, що реалізований прототип здатний коректно обробляти основні типи користувацьких звернень, пов’язані з пошуком інформації про замовлення, багатоетапним визначенням статусу, уточненням неповних даних, застосуванням політик доступу, валідацією аргументів та повторним використанням сесійного контексту. Це підтверджує працездатність запропонованого підходу на рівні MVP-прототипу та створює основу для його подальшого розширення і поглибленого кількісного оцінювання.

4.5 Обмеження реалізованого прототипу та напрями подальшого розвитку

Отримані результати перевірки підтверджують працездатність реалізованого прототипу інтелектуального шлюзу в межах визначених сценаріїв, однак поточне рішення має низку обмежень, які слід враховувати при інтерпретації результатів. Ці обмеження пов’язані як із внутрішньою архітектурною спрощеністю MVP-прототипу, так і з характером проведеної експериментальної перевірки.

До внутрішніх обмежень реалізованого прототипу належить насамперед використання детермінованого rule-based orchestrator замість повноцінного модельного інтерпретатора, що обмежує гнучкість розпізнавання складних або нетипових природномовних запитів. Крім того, у поточній реалізації цикл обробки обмежено трьома ітераціями, що забезпечує керованість і відтворюваність сценаріїв, але водночас звужує можливість виконання довших ланцюжків

оркестрації. Ще одним обмеженням є відносно невеликий набір реалізованих інструментів, орієнтований передусім на демонстрацію базових механізмів роботи шлюзу, а не на охоплення повного спектра можливих дій у системі трекінгу замовлень.

До зовнішніх обмежень належить характер експериментальної перевірки, яка в поточній роботі має переважно інтеграційно-функціональне спрямування. Набір сценаріїв перевірки є відносно компактним і орієнтованим на покриття типових та контрольованих випадків роботи системи. Частина інтеграцій у межах MVP-прототипу спирається на placeholder- або mock-реалізації, зокрема для окремих сценаріїв визначення останнього замовлення користувача та оцінки часу доставки. Крім того, у межах поточного дослідження не виконувалося повномасштабне навантажувальне тестування, а також не проводилося статистично репрезентативне оцінювання якості інтерпретації великого корпусу природномовних запитів. Саме тому результати доцільно трактувати як підтвердження працездатності архітектурного контуру й базових механізмів, а не як остаточну оцінку ефективності завершеного промислового рішення.

Подальший розвиток запропонованого підходу доцільно здійснювати в кількох взаємопов’язаних напрямах. Першим напрямом є інтеграція повноцінного модельного інтерпретатора, здатного працювати з ширшим спектром природномовних формулювань і формувати більш адаптивні плани дій. Другим напрямом є посилення безпеки інтелектуальної оркестрації, зокрема в частині захисту від некоректного використання інструментів, маніпулятивних запитів і ризиків, пов’язаних із model-driven інтерпретацією. Третім напрямом є розширення засобів спостережуваності, включно з глибшим трасуванням сценаріїв, деталізацією метрик продуктивності та накопиченням даних для подальшого кількісного аналізу. Четвертим напрямом є масштабування прототипу шляхом підключення реальних сервісних інтеграцій, розширення набору інструментів і перевірки поведінки системи в умовах складніших та більш різноманітних сценаріїв.

Таким чином, реалізований прототип слід розглядати як працездатний MVP, що підтверджує можливість побудови інтелектуального координаційного шару між природномовним користувацьким запитом і формалізованими сервісними діями в мікросервісному середовищі. Водночас подальше розширення функціональності, підключення повнішого модельного інтерпретатора, поглиблення безпекових механізмів і розширення експериментальної бази є необхідними умовами для переходу від MVP-прототипу до більш зрілого прикладного рішення.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне завдання підвищення гнучкості та керованості обробки природномовних запитів у мікросервісному середовищі на прикладі системи трекінгу замовлень. Актуальність теми зумовлена тим, що традиційні API Gateway ефективно виконують маршрутизацію та агрегацію запитів, але є обмеженими у випадках, коли звернення користувача має природномовну форму, є неповним, неоднозначним або потребує багатоетапної обробки.

У ході дослідження проаналізовано особливості мікросервісної архітектури, підходи до побудови API Gateway, засоби семантичної інтерпретації природномовних запитів, а також механізми оркестрації, керування контекстом і контролю виконання. На цій основі обґрунтовано доцільність побудови інтелектуального координаційного шару між користувачем і мікросервісами.

Основним результатом роботи є запропонований метод інтелектуальної маршрутизації запитів у мікросервісному середовищі, який поєднує семантичну інтерпретацію природномовного звернення з формальними механізмами вибору інструментів, валідації параметрів, політик доступу та контролю виконання. У межах методу передбачено багатоетапну оркестрацію, підтримку уточнення відсутніх даних і використання сесійного контексту для повторного застосування проміжних результатів.

У практичній частині реалізовано прототип інтелектуального шлюзу, архітектура якого охоплює API-рівень, прикладну логіку оркестрації, реєстр інструментів, механізми валідації, політики доступу, підсистему керування станом сесії та засоби трасування виконання. Реалізований прототип виконує роль координаційного шару між природномовним користувацьким запитом і формалізованими сервісними діями.

У межах перевірки сформовано набір інтеграційних сценаріїв, що охоплюють одноетапні й багатоетапні звернення, уточнення, валідаційні відмови, політичні обмеження, контрольовану деградацію та повторне використання сесійного

контексту. Отримані результати підтвердили працездатність базових механізмів запропонованого підходу на рівні MVP-прототипу, зокрема оркестрації, валідації, застосування політик доступу, роботи з контекстом і формування трасувальної інформації.

Практичне значення одержаних результатів полягає в тому, що запропонований підхід і створений прототип можуть бути використані як основа для побудови інтелектуальних інтерфейсів доступу до мікросервісних систем, у яких користувацькі запити мають природномовний характер і потребують контрольованої багатоетапної обробки.

Водночас результати слід інтерпретувати з урахуванням обмежень MVP-прототипу. Поточна реалізація орієнтована передусім на перевірку архітектурного контуру та базових механізмів роботи шлюзу, а частина інтеграцій виконується в контрольованих умовах. Проведена перевірка має переважно інтеграційно-функціональний характер і не охоплює повномасштабного статистичного чи навантажувального оцінювання.

Перспективи подальших досліджень полягають у розширенні набору підтримуваних інструментів, підключенні реальних сервісних інтеграцій, розвитку безпекових механізмів і засобів спостережуваності, а також у переході до повнішого модельного інтерпретатора для обробки складніших природномовних сценаріїв.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Безуглий Н. С. Інтелектуалізація шлюзу API для системи трекінгу замовлень: підхід на основі великих мовних моделей. // 30-й Міжнародний молодіжний форум «Радіоелектроника та молодь XXI столітті». Зб. матеріалів форуму. Т.6. Конференція «Інформаційні інтелектуальні системи» – Харків: ХНУРЕ. 2026.

2. Безуглий Н. С. Інтелектуальна оркестрація сервісних викликів у мікросервісній системі трекінгу замовлень. // 2-га Міжнародна науково-практична конференція «СУЧАСНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ MIT&AIS-2026». Зб. матеріалів конференції. Ч.1. – Харків: ХНУРЕ. 2026.

3. Christopher M. Logistics & Supply Chain Management. 5th ed. Harlow : Pearson UK, 2016. 310 p.

4. Rushton A., Croucher P., Baker P. The Handbook of Logistics and Distribution Management. 7th ed. London : Kogan Page, 2022. 824 p.

5. Taniguchi E., Thompson R. G. City Logistics 2: Modeling and Planning Initiatives. Hoboken : Wiley, 2018. 402 p.

6. Bowersox D., Closs D., Cooper M. B., Bowersox J. Supply Chain Logistics Management. 5th ed. New York : McGraw-Hill Education, 2019. 480 p.

7. Wang Y., Pettit S. E-Logistics: A Guide to Supply Chain Information Systems and Technology. London : Kogan Page, 2016. 536 p.

8. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O’Reilly Media, 2021. 616 p.

9. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island : Manning Publications, 2018. 520 p.

10. API gateways – Azure Architecture Center. URL: https://learn.microsoft.com/en-us/azure/architecture/microservices/design/gateway (дата звернення: 22.03.2026).

11. Microservices: a definition of this new architectural term. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 22.03.2026).

12. Indrasiri K., Siriwardena P. Microservices for the Enterprise: Designing, Developing, and Deploying. Berkeley : Apress, 2018. 422 p.

13. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol : O'Reilly Media, 2017. 616 p.

14. Ocelot Gateway Documentation. URL: https://ocelot.readthedocs.io/ (дата звернення: 22.03.2026).

15. Azure API Management – Overview and Key Concepts. URL: https://learn.microsoft.com/en-us/azure/api-management/api-management-key-concepts (дата звернення: 22.03.2026).

16. API gateway in Azure API Management. URL: https://learn.microsoft.com/en-us/azure/api-management/api-management-gateways-overview (дата звернення: 22.03.2026).

17. Kong Gateway Documentation. URL: https://docs.konghq.com/ (дата звернення: 22.03.2026).

18. Jurafsky D., Martin J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2nd ed. Upper Saddle River : Prentice Hall, 2008. 1024 p.

19. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge : MIT Press, 2016. 800 p.

20. Tunstall L., von Werra L., Wolf T. Natural Language Processing with Transformers. Sebastopol : O'Reilly Media, 2022. 408 p.

21. Manning C. D., Schütze H. Foundations of Statistical Natural Language Processing. Cambridge : MIT Press, 1999. 718 p.

22. Burkov A. The Hundred-Page Machine Learning Book. [S. l.] : Andriy Burkov, 2019. 345 p.

23. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol : O'Reilly Media, 2020. 432 p.

24. Jin B., Sahni S., Shevat A. Designing Web APIs: Building APIs That Developers Love. Sebastopol : O'Reilly Media, 2018. 230 p.

25. Lauret A. The Design of Web APIs. 2nd ed. Shelter Island : Manning Publications, 2024. 235 p.

26. Marrs T. JSON at Work: Practical Data Integration for the Web. Sebastopol : O'Reilly Media, 2017. 376 p.

27. Coleman J. Introducing Speech and Language Processing. Cambridge : Cambridge University Press, 2005. 534 p.

28. Carlson J. L. Redis in Action. Shelter Island : Manning Publications, 2013. 325 p.

29. Freeman A. Pro ASP.NET Core 7. 9th ed. New York : Apress, 2023. 1056 p.

30. Seemann M. Dependency Injection in .NET. Shelter Island : Manning Publications, 2018. 552 p.

31. Price M. C# 10 and .NET 6 – Modern Cross-Platform Development. 6th ed. Birmingham : Packt Publishing, 2021. 800 p.

32. Troelsen A., Japikse P. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. New York : Apress, 2022. 1000 p.

33. Shukla S., Karmakar S. Hands-On Machine Learning with C#. Birmingham : Packt Publishing, 2020. 350 p.

34. Freeman A. Pro ASP.NET Core MVC 2. 7th ed. New York : Apress, 2017. 832 p.

35. Richards M. Software Architecture Patterns. Sebastopol : O'Reilly Media, 2015. 250 p.

36. Newman S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. Sebastopol : O'Reilly Media, 2020. 272 p.

37. Kerr B. API Design Patterns. Shelter Island : Manning Publications, 2021. 424 p.

38. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. Irvine : University of California, 2000. 162 p.

39. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston : Addison-Wesley, 2003. 735 p.

40. Microsoft Azure Documentation. URL: https://learn.microsoft.com/azure/ (дата звернення: 22.03.2026).

41. Microsoft Azure OpenAI Service Documentation. URL: https://learn.microsoft.com/azure/ai-services/openai/ (дата звернення: 22.03.2026).

42. LLamaSharp Documentation. URL: https://scisharp.github.io/LLamaSharp/ (дата звернення: 22.03.2026).

43. SciSharp. LLamaSharp GitHub Repository. URL: https://github.com/SciSharp/LLamaSharp(дата звернення: 22.03.2026).

44. Reis J., Housley M. Fundamentals of Data Engineering. Sebastopol : O’Reilly Media, 2022. 446 p.