

**МАГІСТЕРСЬКА РОБОТА**  
**МР. ШМ - 69.00.00.000 ПЗ**

**Група ШМ-24-3**

**Римар Богдан**

**2025**

Міністерство освіти і науки України  
Івано-Франківський національний технічний університет нафти і газу  
Факультет інформаційних технологій  
Кафедра інженерії програмного забезпечення  
Римар Богдан Миколайович

УДК 004.8

## МАГІСТЕРСЬКА РОБОТА

Моделі, методи та алгоритми оптимізації організації навчального процесу за допомогою чат-ботів на основі LLM

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 – Інженерія програмного забезпечення

(шифр і назва спеціальності)

Римар Б. М.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Григорчук Любомир Іванович доцент, к. п. н.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В. В.

(посада) (підпис) (дата) (ініціал та прізвище)

Нормоконтроль

доц. Вовк Р. Б.

(посада) (підпис) (дата) (ініціал та прізвище)

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

Івано-Франківськ - 2025



## 6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Перевірка на плагіат	доц. к. т. н. Вовк Р. Б.		

7. Дата видачі завдання: \_\_\_\_\_

**Керівник**

\_\_\_\_\_ (підпис)

**Завдання прийняв до виконання**

\_\_\_\_\_ (підпис)

## КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів	Термін виконання етапів роботи	Відмітка про виконання
1	Аналіз предметної області та постановка задачі	24.10.25	виконано
2	Проектування архітектури та вибір технологій	04.11.25	виконано
3	Налаштування програмного середовища та підготовка даних	07.11.25	виконано
4	Розробка підсистеми семантичного пошуку	20.11.25	виконано
5	Реалізація логіки роутера між LLM	01.12.25	виконано
6	Закінчення реалізації необхідних компонентів, тестування та валідація MVP	03.12.25	виконано
7	Оформлення роботи та підготовка до захисту	09.12.25	виконано

**Студент-магістр**

\_\_\_\_\_ (підпис)

**Керівник роботи**

\_\_\_\_\_ (підпис)

## АНОТАЦІЯ

**Магістерська робота:** 79 с., 23 рисунки, 4 табл., 40 джерел, 3 додатки.

**Тема:** моделі, методи та алгоритми оптимізації організації навчального процесу за допомогою чат-ботів на основі LLM

**Об'єкт дослідження:** моделі та алгоритми, що використовуються в чат-боті на основі LLM для полегшення навчальної діяльності.

**Мета роботи:** імплементувати систему для удосконалення навчального процесу, розробка програмного продукту з використанням ШІ для оптимізації організації навчального процесу

**Предмет дослідження:** освітня сфера та шляхи покращення її компонентів, таких як навчального процесу, програмних засобів та платформ за допомогою архітектурних рішень штучного інтелекту.

**Результати дослідження:**

Виконано аналіз існуючих підходів до організації навчання та на їх основі запропоновано власний чат-бот з використанням ШІ.

**Висновок:**

В результаті пророблених досліджень та імплементованих алгоритмів отримано систему комунікації, яка базується на даних, що отримані з освітніх платформ, що вирішує проблему оптимальної організації навчального процесу.

ЧАТ-БОТ, ШТУЧНИЙ ІНТЕЛЕКТ, РОУТЕР, ОСВІТНІ ТЕХНОЛОГІЇ,  
НАВЧАЛЬНИЙ ПРОЦЕС, ГЕНЕРАТИВНИЙ ШТУЧНИЙ ІНТЕЛЕКТ, ПРОМПТ-  
ІНЖИНІРИНГ

## ANNOTATION

**Master's thesis:** 79 pages, 23 figures, 4 tables, 40 sources, 2 appendices.

**Topic:** models, methods and algorithms for optimizing the organization of the educational process using chatbots based on LLM

**Object of research:** models and algorithms used in an LLM-based chatbot to facilitate learning activities.

**Purpose:** implement a system to improve the educational process, develop a software product using AI to optimize the organization of the educational process

**Subject of research:** the educational sphere and ways to improve its components, such as the learning process, software tools, and platforms, using artificial intelligence architectural solutions.

**Research results:**

An analysis of existing approaches to organizing training was performed and, based on them, a custom chatbot using AI was proposed.

**Conclusion:**

As a result of the research and implemented algorithms, a communication system was obtained, which is based on data obtained from educational platforms, which solves the problem of optimal organization of the educational process.

CHAT-BOT, ARTIFICIAL INTELLIGENCE, ROUTER, EDUCATIONAL TECHNOLOGIES, LEARNING PROCESS, GENERATIVE ARTIFICIAL INTELLIGENCE, PROMPT ENGINEERING

## ЗМІСТ

	Стр.
<b>ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ СКОРОЧЕНЬ І ТЕРМІНІВ .....</b>	<b>9</b>
<b>ВСТУП.....</b>	<b>10</b>
<b>РОЗДІЛ 1. ТЕОРЕТИЧНЕ ТА БІЗНЕС-ОБҐРУНТУВАННЯ АГЕНТНИХ СИСТЕМ У СФЕРІ ОСВІТИ.....</b>	<b>16</b>
1.1 Освіта в епоху інновацій.....	16
1.2 Еволюція системи освіти .....	17
1.3 Дослідження чат-ботів та інших систем на основі штучного інтелекту як засобу покращення освітнього процесу.....	19
1.4. Аналіз існуючих засобів та програмних середовищ для реалізації ШІ чат-ботів .....	20
1.5. Висновки до розділу .....	22
<b>РОЗДІЛ 2 СТВОРЕННЯ БАЗОВИХ ЕЛЕМЕНТІВ АРХІТЕКТУРИ ЧАТ-БОТУ .....</b>	<b>24</b>
2.1 Дослідження моделей оркестрації та диспетчеризації запитів в LLM-системах (на прикладі Router Model).....	24
2.2 Визначення функціональних вимог до Generic Model Proxu для обробки стандартних запитів .....	27
2.3 Визначення функціональних вимог до Thinking Tool Model Proxu для обробки комплексних запитів .....	30
2.4 Висновки до розділу.....	33
<b>РОЗДІЛ 3 АНАЛІЗ, ПРОЄКТУВАННЯ ТА РОЗРОБКА АЛГОРИТМІВ КЕРУВАННЯ ПОТОКОМ ІНФОРМАЦІЇ ТА ОБРОБКИ ЗАПИТІВ У NOODLE AI .....</b>	<b>35</b>

3.1 Алгоритм семантичної маршрутизації (Router Model) на основі класифікації запиту.....	35
3.2 Аналіз вимог та розробка сервісу отримання актуальної інформації (WebSearch).....	38
3.3 Аналіз вимог та інтеграція сервісу розкладу .....	39
3.4 Обґрунтування застосування Model Context Protocol (MCP) для взаємодії з Moodle MCP Server.....	45
3.5 Висновки до розділу.....	47
<b>РОЗДІЛ 4 РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ.....</b>	<b>48</b>
4.1 Архітектура та стек технологій реалізації серверної частини системи.....	48
4.2 Програмна реалізація проксі-сервісів: Router, Generic, Thinking.....	49
4.3 Розробка алгоритмів роботи системи.....	56
4.4 Програмна реалізація системи моніторингу використання ресурсів .....	59
4.5 Створення бенчмарку та аналіз результату .....	62
4.6 Висновки до розділу .....	66
<b>ВИСНОВКИ .....</b>	<b>68</b>
<b>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....</b>	<b>69</b>
<b>ДОДАТКИ.....</b>	<b>72</b>

## **ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ СКОРОЧЕНЬ І ТЕРМІНІВ**

- API – інтерфейс прикладного програмування (Application Programming Interface)
- CoT – ланцюжок міркувань (Chain-of-Thought)
- DTO – об'єкт передачі даних (Data Transfer Object)
- FAQ – поширені запитання (Frequently Asked Questions)
- GMP – проксі-сервіс узагальненої моделі (Generic Model Proxy)
- HTTP – протокол передачі гіпертексту (HyperText Transfer Protocol)
- HTTPS – захищений протокол передачі гіпертексту (HyperText Transfer Protocol Secure)
- JSON – нотація об'єктів JavaScript (JavaScript Object Notation)
- LLM – велика мовна модель (Large Language Model)
- LMS – система управління навчанням (Learning Management System)
- MCP – протокол контексту моделі (Model Context Protocol)
- MVP – мінімально життєздатний продукт (Minimum Viable Product)
- RAG – генерація з доповненням пошуком (Retrieval-Augmented Generation)
- ReAct – міркування та дія (Reasoning and Acting)
- REST – передача репрезентативного стану (Representational State Transfer)
- SSE – події, що надсилаються сервером (Server-Sent Events)
- TTMP – проксі-сервіс моделі мислення та використання інструментів (Thinking & Tool Use Model Proxy)
- UI – інтерфейс користувача (User Interface)
- URL – уніфікований покажчик ресурсу (Uniform Resource Locator)
- UUID – універсальний унікальний ідентифікатор (Universally Unique Identifier)
- ВНЗ – вищий навчальний заклад
- ІТ – інформаційні технології
- ІІ (AI) – штучний інтелект (Artificial Intelligence)

## ВСТУП

### **Актуальність роботи.**

Сучасний навчальний процес характеризується високим рівнем цифровізації, але водночас — фрагментацією інформаційних ресурсів. Розклад занять, навчальні матеріали, оцінювання, журнали, комунікація зі студентами та викладачами зберігаються та відбуваються у різних системах (LMS, календарях, чатах, корпоративних платформах), що створює значне когнітивне навантаження та ускладнює доступ до необхідної інформації. Студент змушений перемикатися між декількома сервісами, а викладач витрачає час на пошук і узгодження даних із різних джерел.

Поширені нині LLM-чат-боти для освіти частково вирішують проблему пошуку та менеджменту інформації, однак здебільшого є монолітними системами, що не мають гнучкої архітектури, не забезпечують глибоку інтеграцію з LMS і часто використовують єдину модель для всіх типів запитів. У результаті такі рішення є дорогими в експлуатації, маломасштабованими і не пристосованими до обробки структурованих даних навчального процесу (курсів, модулів, оцінок, розкладу).

Виникає потреба у новому типі інтелектуальних систем, які б поєднували можливості великих мовних моделей із чіткою багатовартовою архітектурою, що дозволяє оптимізувати вартість обчислень, забезпечувати маршрутизацію запитів, підключати зовнішні інструменти та стандартизовано працювати з даними LMS.

У цьому контексті актуальним є створення агентної системи Noodle AI, яка реалізує модульний підхід до обробки запитів, забезпечує інтеграцію з Moodle, розкладом занять та іншими сервісами університетської цифрової інфраструктури. Така система не тільки підвищує ефективність доступу студентів і викладачів до інформації, але й відповідає сучасним викликам та вимогам персоналізованої освіти.

### **Порівняння роботи з відомими розв'язаннями проблеми**

У більшості сучасних рішень для підтримки навчального процесу з використанням ШІ можна виділити кілька типових підходів:

1. Інтегровані чат-боти всередині LMS. Це рішення, коли до Moodle або іншої LMS додається простий чат з LLM, який:

- працює як «розумні підказки» до навчальних матеріалів;
- взаємодіє з LMS переважно через готові API або прості плагіни.

Такі боти зазвичай не мають чітко виділених LLM-моделей на кшталт Router / Generic / Thinking, не підтримують єдиний проксі-доступ до зовнішніх сервісів (розклад, пошук, аналітика) і не розв'язують задачу оптимізації використання ресурсів LLM. У запропонованій системі Noodle AI, навпаки, маршрутизація є центральним компонентом: Router Model приймає рішення, чи достатньо швидкою «дешевою» відповіді Generic-моделі, чи потрібно залучати reasoning-агента з інструментами.

2. Монолітні LLM-асистенти університетського рівня. Частина робіт і проєктів для реалізації задач пропонує єдиного «університетського асистента», який:

- реалізований як один великий сервіс, що безпосередньо звертається до LLM;
- виконує і прості, і складні запити в межах однієї логіки;

Недоліком такого підходу є низька модульність та складність масштабування: додати новий інструмент (нову LMS, окрему БД із розкладом, зовнішній аналітичний сервіс) важко без глибоких змін коду. У Noodle AI замість цього використано проксі-архітектуру з окремими сервісами Generic Model Proxy, Thinking & Tool Use Proxy, Schedule Service та Moodle MCP Server. Це дозволяє підключати нові інструменти як окремі MCP-сервіси без зміни основної логіки Router Model.

3. Горизонтальні корпоративні AI-асистенти. Комерційні рішення яких на кшталт офісних «копілотів» орієнтовані на:

- роботу з документами, поштою, календарями;
- автоматизацію загальних бізнес-процесів;
- інтеграцію з хмарними сервісами офісних пакетів.

Хоча такі системи можуть опосередковано підтримувати навчальний процес (аналіз текстів, генерація завдань, резюме лекцій), вони не інтегровані глибоко в конкретну LMS й, як правило, не працюють із структурованими даними типу

академічних груп, пар, журналів, рейтингів. На відміну від них, Noodle AI спеціально спроектовано під університетське середовище:

- Schedule Service оперує саме розкладом занять і пов'язаними сутностями;
- Moodle MCP Server забезпечує стандартизований доступ до курсів, завдань і балів через MCP-протокол.

Основні відмінності запропонованої системи від наявних підходів можна сформулювати так:

- агентно-проксі архітектура: чітке розділення на Router, Generic та Thinking & Tool Use моделі, що дозволяє оптимізувати вартість запитів і підвищувати відмовостійкість системи;
- стандартизована інтеграція з LMS через Moodle MCP Server, який може бути використаний і в інших LLM-рішеннях без прив'язки лише до Noodle AI;
- спеціалізовані сервісні компоненти (Schedule Service, WebSearch Proxy), орієнтовані саме на задачі освітнього середовища, а не загальне «запитання-відповідь»;
- можливість подальшого розширення: додавання нових MCP-сервісів або інструментів не потребує зміни основної логіки Router Model, що спрощує підтримку й розвиток системи.

Порівняно з типовими інтегрованими чат-ботами в LMS та монолітними університетськими асистентами, запропонована система має більш гнучку, модульну архітектуру й забезпечує глибшу інтеграцію з навчальним процесом при кращому контролі використання обчислювальних ресурсів.

### **Мета і задачі дослідження**

Метою роботи є імплементація системи для удосконалення навчального процесу, а також розробка програмного продукту з використанням ШІ для оптимізації організації навчального процесу.

Досягнення цієї мети вимагає не лише технічної розробки, але й охоплення повного життєвого циклу проєкту, включаючи аналіз вимог, бізнес-аналіз, управління проєктом та методологію тестування ключових компонентів системи.

Для досягнення поставленої мети визначено наступні завдання:

- провести теоретичний аналіз архітектур LLM-агентів, патернів Tool Use та концепції Model Context Protocol (MCP) для обґрунтування вибору мікросервісної структури;
- розробити алгоритм семантичної маршрутизації (Router Model) для ефективного розподілу запитів; створити алгоритми Generic Model та адаптери для інтеграції WebSearch та Schedule Service;
- здійснити тестування розроблених проксі-сервісів (Router, Generic, Thinking) та інструментів, а також проаналізувати метрики їхньої ефективності.

**Об'єктом дослідження** є моделі та алгоритми, що використовуються в чат-боті на основі LLM для полегшення навчальної діяльності.

**Предметом дослідження** є освітня сфера та шляхи покращення її компонентів, таких як навчального процесу, програмних засобів та платформ за допомогою архітектурних рішень штучного інтелекту.

**Методи дослідження:** системний аналіз – для вивчення мікросервісних патернів, LLM-агентів та протоколів інтеграції; теорія алгоритмів – для розробки алгоритмів Router Model, Generic Model та логіки роботи Moodle MCP Server та Schedule Service; бізнес-аналіз та управління проєктами – для визначення вимог, планування етапів розробки та оцінки практичної цінності рішення; імітаційне моделювання та експериментальне тестування – для верифікації точності класифікації інтену в Router Model та оцінки швидкодії проксі-сервісів.

### **Наукова новизна одержаних результатів**

- Запропоновано модульну агентно-проксі архітектуру системи Noodle AI, яка поєднує Router Model, Generic Model Proxy та Thinking & Tool Use Proxy. Така архітектура забезпечує чітке розділення функцій, контроль вартості обчислень і можливість підключення нових інструментів без зміни логіки ядра.

- Розроблено алгоритм семантичної маршрутизації запитів, який автоматично обирає оптимальний шлях обробки (Generic / Thinking / WebSearch / Schedule / Moodle). Запропонований підхід мінімізує використання дорогих reasoning-моделей та підвищує продуктивність системи.
- Запропоновано сервісну модель інтеграції зовнішніх освітніх даних, зокрема Schedule Service та WebSearch Proxy, яка перевершує існуючі вузькоспеціалізовані інтегровані боти завдяки можливості централізованого керування доступом до структурованих даних (розкладу, курсів, завдань, оцінок).

### **Практичне значення одержаних результатів**

Розроблено систему Noodle AI, що є функціональним програмним продуктом, готовим до впровадження у навчальних закладах для оптимізації організації навчального процесу та надання миттєвого доступу до персоналізованих даних.

Реалізовано Router Model Proxy, Generic/Thinking Model Proxy, що дозволяють використовувати дорогі LLM-ресурси виключно за необхідності. Розроблені модулі Schedule Service та Moodle MCP Server можуть бути використані як універсальні мікросервіси для інтеграції будь-якої іншої LLM-системи з відповідними освітніми платформами. Методологія тестування, використана в роботі, може слугувати зразком для подальшого масштабування та розвитку подібних інтелектуальних рішень.

### **Особистий внесок**

1. Спроектовано та розроблено алгоритми, а також програмно реалізовано проксі-сервіси Router Model Proxy та Generic/Thinking Model Proxy для ефективної семантичної маршрутизації та обробки узагальнених запитів.

2. Розроблено та реалізовано адаптери для зовнішніх інструментів: Schedule Service (для парсингу та надання даних розкладу) та WebSearch (для актуалізації знань системи). Спроектовано та реалізовано Moodle MCP Server для стандартизованої та безпечної інтеграції з навчальною платформою.

3. Розроблено план, методологію та проведено функціональне тестування ключових модулів (Router Model Proxy, Generic Model Proxy, Thinking Model Proxy), а також здійснено аналіз метрик ефективності алгоритмів маршрутизації та швидкості відгуку системи.

### **Апробація результатів магістерської роботи**

Основні концепції, що пропрацьовані в магістерській роботі, були представлені та обговорені на XXXVII Науково-Технічній конференції студентів 2024/2025 н. р. в ІФНТУНГ.

### **Публікації**

За результатами наукових досліджень, проведених у магістерській роботі, опубліковано статтю у фаховому виданні України “Вісник Хмельницького національного університету”. Серія: Технічні науки. Тема статті: “Штучний інтелект в освіті: можливості та виклики” (<https://doi.org/10.31891/2307-5732-2025-349-55>).

### **Структура магістерської роботи.**

Магістерська робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків (40 найменувань). Повний обсяг роботи становить 79 сторінок.

# РОЗДІЛ 1.

## ТЕОРЕТИЧНЕ ТА БІЗНЕС-ОБҐРУНТУВАННЯ АГЕНТНИХ СИСТЕМ У СФЕРІ ОСВІТИ

### 1.1 Освіта в епоху інновацій

Сучасний світ характеризується експоненційним розвитком технологій, що сприяє форсованій трансформації та цифровізації всіх сфер життя, особливо освіти. Те, що ще декілька років тому вважалося педагогічною новинкою та сумнівною практикою, як-от використання електронних навчальних платформ чи інтерактивних симуляцій, наразі є стандартною практикою та невід'ємним елементом сучасного освітнього процесу.

Особливо яскраво ця трансформація проявилася у сфері дистанційної та гібридної освіти. Нещодавно очне навчання видавалось незамінною парадигмою, проте глобальні виклики - від епідеміологічних криз до, зокрема, в Україні, військового фактору - виявили критичну вразливість традиційних моделей. Онлайн-навчання стрімко перетворилося на розповсюджену, а часто й безальтернативну, практику, що вимагає від освітніх закладів не лише технічної адаптації, а й методичної інновації.

Цей перехід потягнув за собою не тільки зміну формату, але й кардинальну зміну вимог до персоналізації навчального контенту та оптимізації адміністративних процесів. Збільшення обсягу освітнього контенту, необхідність синхронізації розкладів у віддалених групах та потреба в індивідуальній підтримці кожного студента створили потужний попит на автоматизовані та інтелектуальні рішення. Традиційні системи управління навчанням (LMS), хоч і є основою, виявилися недостатньо гнучкими для оперативного управління цим багатофакторним освітнім середовищем.

Саме в цьому контексті зростає значення штучного інтелекту (ШІ), зокрема великих мовних моделей (LLM). ШІ переходить від ролі пасивного інструменту до ролі активного інтелектуального агента, здатного не просто зберігати інформацію, а

й аналізувати, маршрутизувати запити та взаємодіяти з різними динамічними освітніми сервісами. Таким чином, сучасна освіта вступає в еру, де інновації та технології є не доповненням, а фундаментом ефективної та стійкої організації навчального процесу.

## 1.2 Еволюція системи освіти

Історія освітніх моделей відображає постійну адаптацію до соціально-економічних змін, технологічних інновацій та еволюції педагогічних теорій. Сучасна освіта є результатом трансформації, що пройшла п'ять ключових етапів - від Education 1.0 до актуальної моделі Education 5.0[1].

Початковий етап, Education 1.0, сформувався в епоху промислової революції на початку ХХ століття. Ця модель була орієнтована на вчителя (Teacher-Centered) і функціонувала за принципом "фабричного конвеєра" (Factory Model), основною метою якого було підготувати робочу силу для заводів. Основними рисами були стандартизоване навчання, засноване на підручниках і лекціях (Chalk-and-Talk Method), а також пасивне сприйняття інформації учнями. Основний фокус ставився на зубріння та повторення (Rote Memorization), з оцінюванням за допомогою стандартизованих паперово-олівцевих тестів, що вимірювали здатність до запам'ятовування.

Наступний етап, Education 2.0 (середина–кінець ХХ століття), зберіг вчительську спрямованість, але розпочав інтеграцію освітніх технологій. Ця модель була відзначена появою стандартизованого тестування (Standardized Testing) та розширенням відстеження успішності учнів (Student Monitoring). Завдяки інфузії Ed tech (Ed tech infusion) почалося використання комп'ютерів та Інтернету в класах, що дозволило робити перші спроби диференціації навчання (Differentiation Efforts) та використання даних для формування освітньої політики (Data-Driven Decisions).

Зі зміною тисячоліть, під впливом теорій конструктивізму та соціальних зрушень, відбувся перехід до Education 3.0. Тут акцент змістився на навчання, орієнтоване на учня (Learner-Focused Approach), та технологічну індивідуалізацію.

Характерними рисами стали гіпер-з'єднане навчання (Hyper-connected Learning), проєктний підхід (Project-based Learning) та розвиток креативності. Роль вчителя змінилася на роль консультанта та гіда (Teacher as Guide), який направляє, а не лише надає знання.

Education 4.0 (початок ХХІ століття) зберегла елементи технологічної індивідуалізації, але наголосила на командній роботі та вирішенні реальних проблем. Ця модель акцентує увагу на Problem-based Learning (PBL) (застосуванні знань для вирішення реальних проблем), розвитку емоційного інтелекту (Emotional Intelligence Focus) та міждисциплінарному навчанні (Interdisciplinary Learning). Вчитель перетворився на фасилітатора, який керує спільним навчанням та надає зворотний зв'язок.

Сучасна парадигма, Education 5.0[1] (виникає на початку 2010-х років), є людиноцентричною (Human-Centered) та інноваційно-орієнтованою. Вона ставить на чільне місце розвиток людських навичок (Human Skills) (наприклад, креативність, інновації, критичне мислення, комплексне вирішення проблем), які є життєво важливими для навігації у світі невизначеності.

Розглянемо ключові особливості Education 5.0, які є критичними для архітектури системи:

- персоналізоване навчання, орієнтоване на індивідуальні інтереси та мотивацію;
- використання штучного інтелекту для посилення та персоналізації навчального досвіду;
- роль вчителя як куратора, що направляє студентів до кастомізованих ресурсів.
- акцент на безперервному навчанні.

Таким чином, еволюція освіти від Education 1.0 до Education 5.0 демонструє постійний рух до гіпер-персоналізації та зростання залежності від інтелектуальних технологій. Це створює платформу для подальшого аналізу ролі LLM-систем як агентів-посередників, здатних використовувати ШІ для маршрутизації та забезпечення індивідуалізованого доступу до різномірних освітніх даних, що є основою даної магістерської роботи.

### 1.3 Дослідження чат-ботів та інших систем на основі штучного інтелекту як засобу покращення освітнього процесу

Еволюція освітніх моделей до парадигми Education 5.0 вимагає відходу від уніфікованого підходу до навчання на користь гіпер-персоналізації та розвитку людиноцентричних навичок (Human Skills), таких як креативність, критичне мислення та комплексне вирішення проблем. Для реалізації цих завдань освітній процес дедалі більше покладається на штучний інтелект (ШІ), що розглядається як інструмент для посилення та індивідуалізації навчального досвіду (AI Leverage).

Інтеграція ШІ в освіту відбувається за декількома основними напрямками, які безпосередньо покращують освітній процес, адаптуючи його до вимог Суспільства 5.0:

- персоналізоване навчання та адаптація контенту (Personalized Learning): ШІ використовується для аналізу індивідуальних мотивацій та пристрастей студентів, що дозволяє Tailor education. Це значно виходить за межі простої диференціації, характерної для Education 2.0, і забезпечує глибоку кастомізацію освіти навколо інтересів. ШІ виступає в ролі вчителя-куратора, який скеровує студентів до найбільш релевантних ресурсів, заохочуючи їх до Meta Learning;
- підтримка міждисциплінарного підходу: ШІ-системи можуть автоматично ідентифікувати та поєднувати концепції, інтегруючи ідеї з різних предметних областей, що є ключовою вимогою Education 5.0;
- автоматизація адміністративних та рутинних завдань: системи ШІ значно знижують навантаження на викладачів, автоматизуючи рутинні процеси, що дозволяє їм перетворитися з "постачальників знань" на фасилітаторів та кураторів.

Чат-боти на основі LLM є найбільш динамічним інструментом ШІ в освіті. Вони еволюціонували із простих систем Education 2.0 до складних інтелектуальних агентів, здатних до комплексних взаємодій. Проте, саме тут виникає архітектурна складність, яка є об'єктом даного дослідження. Для функціонування в реальному

освітньому середовищі, як-от Noodle AI, чат-боти стикаються з проблемою множинності та динамічності джерел даних. Користувач може запитати:

- загальну інформацію, яка вимагає використання Generic Model.
- актуальну інформацію, що вимагає використання інструменту Thinking model та конкретного інструменту Schedule Service.
- персоналізовану інформацію про навчання, що вимагає безпечної авторизованої взаємодії з LMS Moodle через Moodle MCP Server.

Для ефективного вирішення цих завдань традиційні підходи недостатні.

Необхідна агентна архітектура, яка повинна забезпечити:

- семантичну маршрутизацію: розробка Router Model для миттєвого та точного визначення інтенту запиту і його перенаправлення до правильного модуля (LLM, RAG, Tool, WebSearch) для забезпечення швидкості та економічності.
- інтеграцію з зовнішніми інструментами: використання Model Context Protocol (MCP) для стандартизованого підключення адаптерів живих даних (Schedule Service), що гарантує актуальність та достовірність персоналізованих даних, на відміну від статичних баз знань.

Таким чином, для повноцінного переходу освіти до моделі 5.0 та забезпечення гіпер-персоналізованого досвіду, чат-боти повинні стати архітектурно-диференційованими агентами, здатними оркеструвати взаємодію з різними освітніми платформами. Дослідження та розробка цих механізмів маршрутизації та інтеграції становить наукову новизну даної роботи.

#### **1.4. Аналіз існуючих засобів та програмних середовищ для реалізації ШІ чат-ботів**

Сучасна екосистема розробки інтелектуальних чат-ботів значно еволюціонувала від використання простих скриптів з правилами (rule-based) до складних агентних фреймворків, що дозволяють оркеструвати роботу великих мовних моделей (LLM). Для реалізації системи типу Noodle AI, яка вимагає семантичної маршрутизації та використання зовнішніх інструментів, необхідно

проаналізувати ключові програмні середовища за трьома напрямками: фреймворки оркестрації, сервіси виконання моделей (Inference) та протоколи інтеграції даних.

Основним викликом при створенні агентів є необхідність поєднувати генеративні можливості моделі з програмною логікою. На ринку існують такі ключові рішення:

- LangChain: Це найпопулярніший фреймворк з відкритим кодом для створення застосунків на основі LLM. Його ключовою перевагою є абстракція "Ланцюжків" (Chains) та "Агентів" (Agents). LangChain надає вбудовані механізми для реалізації патернів ReAct (Reasoning and Acting) та CoT (Chain of Thought), що є критично важливим для реалізації Thinking & Tool Use Model. Крім того, він має широку бібліотеку інтеграцій (Tools), що спрощує підключення пошукових систем та API;
- LlamaIndex: Цей фреймворк фокусується насамперед на роботі з даними — індексації, пошуку та RAG (Retrieval-Augmented Generation). Хоча він ефективний для створення баз знань, його можливості щодо керування складною агентною логікою та маршрутизацією дещо поступаються LangChain;
- Microsoft Semantic Kernel: SDK, що дозволяє інтегрувати LLM з існуючим кодом. Він є потужним, але орієнтованим переважно на екосистему .NET/C#, хоча має і Python-версію.

Для задач Noodle AI, що включають складну маршрутизацію та мульти-агентну взаємодію, LangChain виглядає найбільш оптимальним вибором завдяки гнучкості та розвинутій екосистемі Python.

Чат-боти вимагають високої швидкодії та підтримки асинхронних з'єднань для стрімінгу тексту (token streaming):

- FastAPI: Сучасний веб-фреймворк для Python, що базується на стандартних підказках типів (type hints). Його ключовою особливістю є нативна підтримка асинхронності (async/await) та протоколів ASGI. Це робить його ідеальним для реалізації мікросервісів-проксі, які повинні обробляти велику кількість

одночасних запитів до LLM та підтримувати протокол Server-Sent Events (SSE) для стрімінгу;

- Flask/Django: Традиційні фреймворки, які, хоч і є надійними, мають обмеження в обробці асинхронних подій та веб-сокетів порівняно з FastAPI, що може створювати затримки (latency) при роботі з LLM.

Важливим аспектом є спосіб доступу до самих моделей.

- OpenAI API: Де-факто галузевий стандарт інтерфейсу для чат-компліцій (/v1/chat/completions). Використання цього стандарту дозволяє легко замінювати моделі без зміни коду клієнта;
- LM Studio / Ollama: Програмні середовища для локального запуску відкритих моделей (Llama 3, Mistral, Gemma). Вони надають можливість розгорнути локальний сервер, що імітує OpenAI API. Це критично важливо для розробки та тестування систем без витрат на хмарні токени, а також для забезпечення приватності даних.

Новітнім напрямком у розвитку чат-ботів є стандартизація підключення до зовнішніх даних. Model Context Protocol[30] (MCP) — це відкритий стандарт, який дозволяє розробляти універсальні конектори до джерел даних (наприклад, LMS Moodle)[29]. На відміну від створення кастомних API-обгортки, MCP дозволяє моделям "розуміти" структуру даних та доступні дії (Tools) стандартизованим чином, що значно спрощує масштабування системи.

## 1.5. Висновки до розділу

Проведено комплексне теоретичне дослідження сучасної освітньої парадигми та визначено місце й роль інтелектуальних систем на основі LLM в її трансформації.

На основі аналізу встановлено, що сучасна освіта еволюціонувала від жорсткої, орієнтованої на вчителя моделі Education 1.0 до людиноцентричної парадигми Education 5.0. Цей перехід був каталізований суспільними та економічними змінами, а також прискорений технологічними інноваціями.

Встановлено, що ключовим інструментом для реалізації вимог Education 5.0 є чат-боти на основі LLM, які функціонують як інтелектуальні агенти. Агент повинен бути здатний не лише генерувати відповіді, але й оркеструвати взаємодію із зовнішніми сервісами. Це вимагає відходу від монолітних рішень до складних агентних архітектур з використанням механізмів Tool Calling та семантичної маршрутизації (Routing).

## РОЗДІЛ 2

### СТВОРЕННЯ БАЗОВИХ ЕЛЕМЕНТІВ АРХІТЕКТУРИ ЧАТ-БОТУ

#### 2.1 Дослідження моделей оркестрації та диспетчеризації запитів в LLM-системах (на прикладі Router Model)

Проектування ефективної системи на основі великих мовних моделей (LLM) вимагає стратегічного розподілу обчислювальних ресурсів. У сучасних інтелектуальних агентних архітектурах прийнято розрізняти два основні функціональні режими або типи моделей за їхньою здатністю до міркування (Reasoning). Це розмежування є ключовим для оптимізації швидкості відгуку, точності та експлуатаційних витрат.

Узагальнені (Non-Reasoning) моделі – це типові реалізації LLM, сфокусовані на прямому та контекстно-залежному генеруванні відповіді без необхідності складного планування дій чи залучення зовнішніх ресурсів. Вони виконують роль інформаційного ядра системи:

- принцип роботи: ці моделі оперують знаннями, на яких вони були навчені, або інформацією, що міститься у короткостроковій пам'яті поточної сесії. Їхня функція – швидко та якісно відповідати на загальнопізнавальні або контекстно-зв'язані запити;
- сфера застосування: обробка типових запитів, що не вимагають динамічної взаємодії з реальним світом: надання визначень, пояснення концепцій, генерація тексту, узагальнення інформації;
- переваги: висока швидкість обробки та низьке ресурсоспоживання. Це робить їх економічно вигідними для обробки великої кількості простих, але частих запитів;
- обмеження: нездатність до виконання операційних завдань, що вимагають верифікації актуальних даних або складного покрокового виконання.

Міркувальні (Reasoning) моделі – це архітектури або агенти, що використовують LLM як планувальник для виконання складних, багатоетапних

завдань. Їхня основна функція - не просто відповісти, а виконати завдання в віртуальному середовищі:

- механізм роботи: вони застосовують техніки, як-от Chain-of-Thought (CoT) або ReAct, для формування внутрішнього плану міркування (планування), динамічного виклику необхідних інструментів (Tool Calling), аналізу отриманих результатів (Observation) та інтеграції їх у фінальну відповідь. На рисунку 2.1 продемонстровано різницю між Generic та Reasoning моделями:

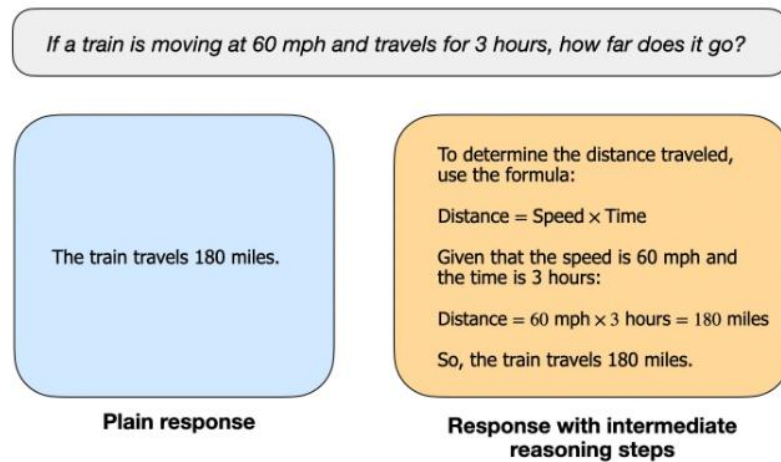


Рис. 2.1. Демонстрація різниці між звичайною відповіддю та з відповідями, де видно кроки Reasoning

- сфера застосування: виконання операційних завдань, що вимагають доступу до динамічних або персоналізованих даних (наприклад, перевірити розклад студента, знайти актуальну інформацію в Інтернеті, надіслати запит до зовнішньої бази даних);
- переваги: висока точність для операційних завдань, оскільки відповідь ґрунтується на актуальних, верифікованих даних, а не лише на статичних знаннях моделі;
- вимоги: більша кількість токенів для формування плану міркування і часто залучають потужніші LLM, що робить їх дорожчими та повільнішими порівняно з узагальненими моделями. Наведемо приклад (рис. 2.2).
- оцінка результатів: тестові запити можуть оцінити точність моделі

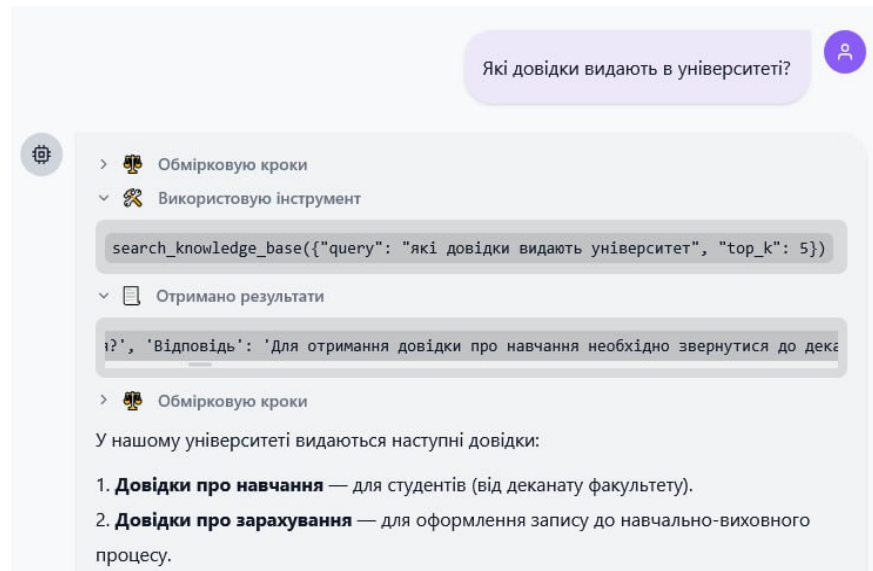


Рис. 2.2. Reasoning в Noodle AI

Імплементация цих двох типів моделей в одну систему призводить до необхідності створення інтелектуального маршрутизатора (Router). Його завдання полягає у семантичній диспетчеризації запитів:

- якщо інтент запиту є загальним і не вимагає зовнішніх інструментів, запит направляється до Non-Reasoning моделі для швидкої та економічної обробки;
- якщо запит вимагає виконання дії, доступу до динамічних даних або складного міркування, він направляється до Reasoning-агента та його інструментів.

Наведено відповідний приклад (рис. 2.3)

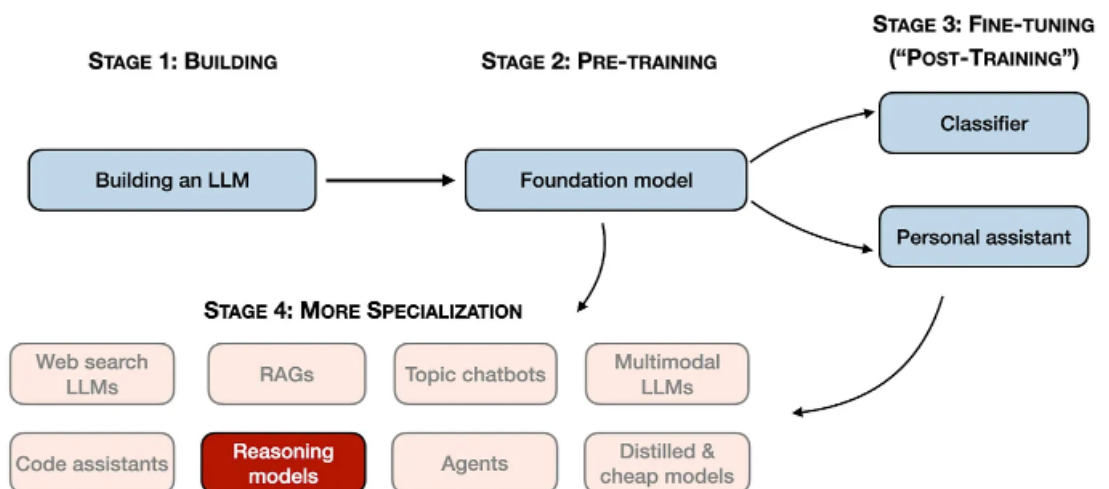


Рис. 2.3. Приклад побудови ШІ системи персонального асистента та місце в ній Reasoning Model

Таким чином, розмежування між Non-Reasoning та Reasoning компонентами є фундаментальною архітектурною моделлю, яка забезпечує оптимізацію ресурсів, масштабованість та функціональну гнучкість інтелектуальної системи.

## **2.2 Визначення функціональних вимог до Generic Model Proxy для обробки стандартних запитів**

Generic Model Proxy (GMP) є критично важливим компонентом архітектури Noodle AI і слугує точкою входу для узагальнених, інформаційних (Non-Reasoning) запитів, які не вимагають доступу до динамічних зовнішніх інструментів. Основне завдання GMP - забезпечити швидку, економічну та якісну обробку більшості стандартних комунікацій. Вимоги до GMP поділяються на функціональні (що система повинна робити) та нефункціональні (як система повинна працювати).

Функціональні вимоги до GMP визначають його основні можливості в обробці запитів:

- обробка узагальнених запитів: GMP повинен приймати запити, класифіковані Router Model як "Generic" або "Informational", і генерувати відповіді на основі знань, вбудованих у модель або наданих через контекст. Приклади: надання визначень, пояснення термінів, генерація коротких текстів, відповіді на загальноосвітні питання;
- управління контекстом розмови (Chat Memory): GMP повинен інтегруватися з модулем Chat Memory для збереження та використання історії попередніх запитів та відповідей у поточній сесії. Мета: підтримувати зв'язність діалогу та забезпечувати контекстну релевантність відповідей;
- форматування вихідних даних: GMP повинен формувати відповіді, отримані від LLM, у єдиний стандартизований вихідний формат (наприклад, JSON, що містить текст, відформатований у Markdown), зрозумілий для Chatbot UI;
- стійкість до нецільових запитів: GMP повинен бути запрограмований (через системний промт, або "гайдрейл") на відхилення або перенаправлення запитів, які, незважаючи на класифікацію Router Model, вимагають Tool Use

(наприклад, "Який мій розклад?"). У таких випадках GMP має генерувати попереджувальне повідомлення та перенаправляти на Thinking & Tool Use Model Proxy;

- сумісність з API: GMP повинен функціонувати як проксі-сервіс, підтримуючи OpenAI API-сумісний інтерфейс для безшовної інтеграції з локальними (наприклад, LM Studio) або хмарними LLM-провайдерами.

Нефункціональні вимоги визначають якість та ефективність роботи GMP, що є ключовим для оптимізації освітнього процесу та впливають на реальний комфорт взаємодії з ботом:

- швидкодія (Latency): час відгуку GMP для стандартних запитів (P90) повинен бути мінімальним (наприклад, не перевищувати 3-5 секунд), оскільки ці запити становлять основний обсяг комунікації. Обґрунтування: висока швидкість забезпечує якість користувацького досвіду і сприяє активнішій взаємодії;
- ресурсоефективність: GMP повинен бути налаштований на використання менш ресурсоемних LLM або на мінімальне споживання токенів, оскільки його функціонал є "дешевим" і не вимагає складного reasoning. Обґрунтування: зниження експлуатаційних витрат;
- масштабованість: GMP повинен підтримувати горизонтальне масштабування, дозволяючи обробляти велику кількість одночасних сесій без значного збільшення часу відгуку, що критично для університетського середовища;
- надійність: GMP повинен коректно обробляти помилки з'єднання з LLM-ядром, надаючи користувачу зрозуміле повідомлення про помилку.

Таким чином, Generic Model Proxy є основою для економічної та швидкої комунікації системи, дозволяючи Router Model ефективно ізолювати дорогі та повільні операції Reasoning від масової інформаційної підтримки. Код, що реалізовує Generic Model Proxy, розподілений на певні компоненти. Їх перелік наведено в таблиці 2.1 та включає в себе компоненти з таких файлів, як llm\_service.py, v1.py, chat.py, config.py. Усі вони разом складають основну логіку взаємодії з користувачем та описують необхідні функції.

Таблиця 2.1

## Компоненти Generic Model Proxy

Компонент	Призначення та функція
LLMService	Це сервісний шар, що містить логіку взаємодії з LLM. Використовує LangChain (ChatOpenAI) для підключення до LLM через змінні середовища (OPENAI_API_URL, OPENAI_API_KEY, MODEL). Ключова особливість: ін'єкція SYSTEM_PROMPT як першого повідомлення до кожного запиту, що гарантує дотримання моделі її ролі
API Endpoint	Встановлює endpoint /chat/completions (OpenAI-сумісний). Використовує Dependency Injection для отримання екземпляра LLMService.
Обробка запитів	Обробляє два типи запитів: звичайна відповідь (non-stream) та стрімінг (stream).
Стрімінг	Функція stream_generator реалізує логіку формування Server-Sent Events (SSE). Вона асинхронно ітерує по шматках відповіді від LLM (llm_service.get_chat_completion_stream), формуючи OpenAI-сумісні чанки (ChatCompletionStreamResponse) та надсилаючи їх клієнту. Це забезпечує відображення відповіді користувачу в реальному часі.
Обробка без стрімінгу	Функція create_chat_completion для non-stream запитів викликає llm_service.get_chat_completion і формує повну JSON-відповідь з метаданими (ID, час створення, usage-інформація про токени).
Моделі даних	Використовує Pydantic для стандартизації форматів запитів (ChatCompletionRequest) та відповідей (ChatCompletionResponse, ChatCompletionStreamResponse), наслідуючи структуру OpenAI Chat Completions API
Конфігурація	Забезпечує завантаження всіх конфігураційних параметрів (включаючи URL, ключ, назву моделі та системний промт) із файлу .env для гнучкості розгортання.

## 2.3 Визначення функціональних вимог до Thinking Tool Model Proxy для обробки комплексних запитів

Thinking & Tool Use Model Proxy (TTMP) є міркувальним агентом (Reasoning Agent) системи Noodle AI і відповідає за обробку комплексних, багатоетапних запитів, які вимагають взаємодії з динамічними даними. Його архітектура є значно складнішою, ніж у Generic Model Proxy, оскільки включає ядро LangChain AgentExecutor.

Функціональні вимоги Thinking & Tool Use Model Proxy визначають його здатність до виконання складних операцій та взаємодії з іншими мікросервісами:

- підтримка складних агентних механізмів: Thinking & Tool Use Model Proxy повинен бути ініціалізований як LangChain Agent (`create_openai_tools_agent`) і використовувати `AgentExecutor` для керування процесом міркування та виконання завдань;
- інтеграція зовнішніх інструментів (Tool Calling): Thinking & Tool Use Model Proxy повинен мати можливість динамічно викликати зареєстровані інструменти (`all_tools`), які відповідають за функціонал WebSearch, Schedule Service, Moodle MCP Server та інші;
- перетворення повідомлень для агента: Thinking & Tool Use Model Proxy повинен конвертувати вхідні повідомлення (`List[Message]`) з API-формату у формат, зрозумілий для LangChain (`HumanMessage`, `AIMessage`), і правильно ініціювати агент з останнім запитом (`user_input`);
- управління контекстом і історією: Thinking & Tool Use Model Proxy повинен використовувати `MessagesPlaceholder("chat_history")` у своєму промті для забезпечення контекстуальної пам'яті (пам'яті сесії) та `MessagesPlaceholder("agent_scratchpad")` для внутрішнього міркування агента;
- обробка стрімінгу (SSE Transformation): на відміну від Generic Model Proxy, Thinking & Tool Use Model Proxy повинен трансформувати внутрішні події

агента (`agent_executor.astream_events`) у стандартизований потік SSE. Це включає: виділення шматків тексту (`event_type == "chunk"`); ідентифікацію та інкапсуляцію команд виклику інструментів (`event_type == "tool_call"`) у спеціальні теги (`<tool_call>... </tool_call>`) для подальшої обробки клієнтом;

- обробка результатів Tool Observation: Thinking & Tool Use Model Proxy повинен трансформувати вихідні дані інструментів (`event_name == "on_tool_end"`) у спеціальні теги (`<tool_output>... </tool_output>`) для включення їх у потік міркування агента.

Для обробки комплексних завдань розглянемо нефункціональні вимоги які визначають якість роботи:

- точність міркування: Thinking & Tool Use Model Proxy повинен демонструвати високу точність у виборі інструментів та послідовності кроків (що залежить від `SYSTEM_PROMPT` та навченості базової моделі), оскільки невірний вибір інструменту призводить до невірної відповіді;
- наскрізна асинхронність: Thinking & Tool Use Model Proxy повинен використовувати асинхронні механізми (`ainvoke`, `astream_events`) на всіх рівнях (від FastAPI до LangChain та викликів інструментів) для мінімізації часу очікування, особливо під час звернення до повільних зовнішніх API (WebSearch, Moodle);
- конфігурованість зовнішніх інструментів: параметри підключення до всіх зовнішніх сервісів (`MOODLE_API_URL`, `SCHEDULE_SERVICE_URL`, `EMBEDDING_STORE_URL`) повинні бути винесені у конфігураційний файл (`config.py`) для забезпечення гнучкості розгортання та безпеки;
- рольова ізоляція: Thinking & Tool Use Model Proxy повинен використовувати System Prompt (`SYSTEM_PROMPT`) для чіткого визначення своєї ролі як агента-планувальника, що забезпечує ізоляцію функціоналу від Generic Model Proxy і запобігає нецільовому використанню дорогих Reasoning-ресурсів.

Усі ці вимоги реалізовані в наступних компонентах:

Таблиця 2.2

## Компоненти Thinking Model Proxy

Компонент	Призначення та функція
LLMService	Сервісний шар
Agent Executor	Включає AgentExecutor – ядро, яке керує багатоетапним процесом: отримує запит, формує план, викликає необхідні інструменти (all_tools) і генерує фінальну відповідь.
Системний Промпт & Історія	Агент ініціалізується з ChatPromptTemplate, що містить SystemMessage (SYSTEM_PROMPT), плейсхолдер для історії чату (MessagesPlaceholder("chat_history")) та плейсхолдер для внутрішнього міркування (MessagesPlaceholder("agent_scratchpad")). Це забезпечує зв'язність діалогу та контроль над логікою міркування.
Динамічні Інструменти	Файл config.py містить змінні середовища для підключення до зовнішніх сервісів: Google Contacts, MOODLE_API_URL, SCHEDULE_SERVICE_URL, EMBEDDING_STORE_URL. Сам агент (llm_service.py) імпортує їх через all_tools.
API Endpoint	Підтримує OpenAI-сумісний endpoint /chat/completions.
Обробка Стрімінгу (SSE)	Функція stream_generator є складною: вона не просто передає текст, а трансформує LangChain-події (events) (agent_executor.astream_events) у OpenAI-сумісний формат та структурований текст. Вона використовує спеціальну логіку для інкапсуляції викликів інструментів (<tool_call>... </tool_call>) у потік, дозволяючи клієнту візуалізувати процес міркування агента.
Моделі даних	Розширені Pydantic-моделі підтримують поля tool_calls (ToolCall, FunctionCall), які є обов'язковими для передачі команд від агента.

Також, ключові функціональні вимоги Thinking & Tool Use Model Proxy виконуються шляхом інтеграції наступних спеціалізованих інструментів.

Таблиця 2.3

## Інструменти, що використовує Thinking Model

Інструмент	Функція в системі Noodle AI	Роль у Reasoning
moodle_integration	Доступ до LMS Moodle для отримання персоналізованої інформації (read-only).	Вирішує завдання персоналізації (Education 5.0). Використовується для запитів: get_grades, get_user_courses, get_course_materials, get_user_info.
get_schedule	Доступ до сервісу розкладів для отримання актуального навчального графіку.	Вирішує завдання організації. Використовується для запитів: get_schedule, get_groups, get_teachers.
search_knowledge_base	Пошук у внутрішній базі знань (RAG Indexer, Embedding Store).	Забезпечує контекстне обґрунтування. Використовується, коли інформація занадто стара для Web Search, але занадто специфічна для Generic Model.
web_search_tool	Виконує пошук у реальному часі (DuckDuckGo).	Забезпечує актуальність. Використовується для запитів про поточні події, новини чи загальні знання, яких немає у внутрішній базі.

## 2.4 Висновки до розділу

Виконано аналіз архітектурних рішень та проектування ключових компонентів системи Noodle AI, що є необхідною передумовою для подальшої алгоритмізації. Визначено та обґрунтовано необхідність використання багаторівневої агентної архітектури для оптимізації ресурсів та забезпечення функціональної гнучкості. Встановлено, що для забезпечення економічності та швидкодії системи

життєво необхідним є чітке розмежування функціоналу між узагальненими (Generic Model Proxy), та міркувальними (Reasoning Thinking & Tool Use Model Proxy) моделями.

Успішне визначення функціональних вимог до Generic Model Proxy та Thinking & Tool Use Model Proxy, а також проектування їхньої взаємодії, закладає надійний архітектурний фундамент для системи Noodle AI. На основі проведеного аналізу та проектування в наступному розділі буде розроблено детальні алгоритми роботи ключових компонентів. Зокрема, увага буде зосереджена на алгоритмі семантичної маршрутизації (Router Model), який є критичним для забезпечення ефективного перенаправлення запитів між спроектованими проксі-моделями (Generic Model Proxy та Thinking & Tool Use Model Proxy) та їхніми інструментами.

## РОЗДІЛ 3

### АНАЛІЗ, ПРОЄКТУВАННЯ ТА РОЗРОБКА АЛГОРИТМІВ КЕРУВАННЯ ПОТОКОМ ІНФОРМАЦІЇ ТА ОБРОБКИ ЗАПИТІВ У NOODLE AI

#### 3.1 Алгоритм семантичної маршрутизації (Router Model) на основі класифікації запиту

Алгоритм семантичної маршрутизації є критично важливим шаром управління в системі Noodle AI, забезпечуючи необхідну ефективність, економічність та швидкість обробки запитів. Цей алгоритм, реалізований у компоненті Router Model Proxu, функціонує як інтелектуальний диспетчер, що визначає оптимальний шлях обробки для кожного вхідного запиту. Його основна роль полягає у семантичній класифікації наміру користувача (інтенту) для подальшого перенаправлення до відповідного проксі-сервісу.

Маршрутизатор розв'язує фундаментальну архітектурну проблему, розділяючи запити на дві категорії: узагальнені (Non-Reasoning), які є частими, але дешевими, та комплексні, які є рідшими, але вимагають дорогих обчислювальних ресурсів та Tool Use. Ядро маршрутизатора реалізовано у класі RouterService як мультикласовий класифікатор, що використовує окрему Router LLM. Для забезпечення надійного та передбачуваного результату класифікації, LLM налаштована на температуру  $T=0.0$ , що робить процес детермінованим і мінімізує імовірність креативних, але невірних рішень. Ось приклад маршрутизатора (рис. 3.1):



Рис. 3.1. Принцип роботи роутера LLM

Router Model приймає рішення, вибираючи один із двох класів інтенту: `GENERIC` – запит, що перенаправляється до Generic Model Proxy (GMP) для швидкої обробки, або `THINKING_TOOL_USE` – запит, що вимагає Reasoning та Tool Calling, який перенаправляється до Thinking & Tool Use Model Proxy (TTMP). Це архітектурне рішення значно знижує експлуатаційні витрати та підвищує загальну швидкодію системи. Алгоритм, реалізований у методі `route_and_proxy`, виконує чітку послідовність кроків для кожного вхідного запиту.

Спочатку відбувається визначення запиту, де вилучається останній запит користувача (`user_query`) із послідовності повідомлень. Цей запит, що відображає поточний намір, є єдиним вхідним параметром для подальшої класифікації.

Далі відбувається класифікація інтенту шляхом виклику `_determine_route(user_query)`. Router LLM повертає рішення у вигляді `GENERIC` або `THINKING_TOOL_USE`. У випадку, якщо модель повертає непередбачений результат, або при відсутності запиту, алгоритм реалізує механізм безпечного `fallback`, автоматично встановлюючи інтент `GENERIC` та логуючи попередження.

Наступним кроком є вибір цільового проксі та проксіювання. На основі рішення класифікатора встановлюється цільовий URL (`GENERIC_MODEL_PROXY_URL` або `THINKING_TOOL_MODEL_PROXY_URL`), які беруться з конфігураційного файлу. Використовуючи асинхронний HTTP-клієнт `httpx.AsyncClient`, метод `_proxy_request` асинхронно передає запит до відповідного проксі-сервісу. Цей проміжний шар проксіювання забезпечує єдиний вхідний пункт (`/chat/completions`) і стандартизовану обробку даних.

Router Model Proxy також підтримує наскрізний стрімінг (`route_and_proxy_stream`), що є критичним для забезпечення високої якості користувацького досвіду, особливо при обробці складних запитів TTMP, що використовує багатопотоковий стрімінг із тегами Tool Use. Router не аналізує вміст чанків, а просто асинхронно передає байти відповіді від цільового проксі клієнту, мінімізуючи затримку (`latency`).

Крім того, алгоритм має вбудовану систему обробки помилок, що є обов'язковим елементом мікросервісної архітектури. Використання `try...except` та

методів `httpx.HTTPStatusError` дозволяє коректно перехоплювати помилки, що виникають на бекенд-проксі-сервісах (GMP або TTMP), та перетворювати їх на зрозумілі HTTP-помилки з відповідними статусами (наприклад, 400 Bad Request або 503 Service Unavailable). Це підвищує загальну надійність та стійкість системи Noodle AI. В таблиці 3.1 описано, як саме працює роутер.

Таблиця 3.1

## Кроки, по яким працює Router

Крок	Дія	Опис та Обґрунтування
Визначення Запиту	Вилучення останнього запиту користувача ( <code>user_query</code> ).	Router оперує лише поточним наміром користувача, ігноруючи історію для класифікації, що забезпечує швидкість.
Класифікація Інтенту	Виклик <code>_determine_route(user_query)</code> .	Router LLM повертає <code>GENERIC</code> або <code>THINKING_TOOL_USE</code> . При несподіваному результаті встановлюється <code>GENERIC</code> за замовчуванням (безпечний <code>fallback</code> ).
Вибір Цільового Проксі	Встановлення <code>target_proxu</code> (URL GMP або URL TTMP) на основі рішення класифікатора.	Пряме перенаправлення запиту до відповідного мікросервісу.
Підготовка Запиту	Створення нового об'єкта <code>proxu_request</code> .	Забезпечується інжекція необхідних системних повідомлень, гарантуючи, що цільовий проксі отримає всі необхідні інструкції.
Проксіювання	Виклик <code>_proxu_request(target_url, proxu_request)</code> за допомогою <code>httpx.AsyncClient</code> .	Асинхронна передача запиту до цільового проксі-сервісу та очікування (або стрімінг) відповіді.
Обробка Помилки	Використання <code>try...except</code> та <code>response.raise_for_status()</code> .	Використання <code>try...except</code> та <code>response.raise_for_status()</code> .

### 3.2 Аналіз вимог та розробка сервісу отримання актуальної інформації (WebSearch)

Сервіс WebSearch є невід'ємним компонентом архітектури Noodle AI, що забезпечує міркувальний агент (Thinking & Tool Use Model Proxy, TTMP) здатністю оперувати актуальною інформацією та знаннями, які не були інтегровані у статичні бази даних (Embedding Store) чи внутрішні сховища. Ця функціональність критично важлива для відповідності вимогам Education 5.0, де важлива не лише наявність знань, а й здатність їх критично оцінювати та оновлювати у реальному часі.

Аналіз вимог до сервісу WebSearch виділяє ключові функціональні та нефункціональні аспекти. Основна функціональна вимога - забезпечення пошуку в реальному часі для запитів, що стосуються поточних подій, останніх наукових розробок або інформації, залежної від часу. Сервіс має приймати вхідний запит у вигляді простого текстового рядка, який генерується TTMP, та повертати обмежену кількість найбільш релевантних результатів у стислому текстовому форматі, придатному для подальшої обробки LLM. Серед нефункціональних вимог пріоритетними є швидкість виконання, оскільки час відгуку безпосередньо впливає на загальну Latency TTMP, та надійність джерела інформації. Крім того, сервіс повинен бути реалізований як ізольований інструмент, що мінімізує залежність від LLM-ядра.

Для реалізації цих вимог розроблено функцію `web_search_tool`, яка інтегрована до Thinking & Tool Use Model Proxy як інструмент LangChain. Алгоритм роботи інструменту включає наступні кроки.

Початково TTMP, визначивши, що для відповіді потрібна актуальна інформація, викликає `web_search_tool(query: str)`, формуючи чіткий пошуковий запит. На цьому етапі відбувається ініціалізація провайдера: у коді використовується пошуковий провайдер `DuckDuckGoSearchResults`, що забезпечує анонімність та ефективність для агрегації текстових сніпетів, які легко обробляються LLM. Встановлюються параметри для обмеження результатів, зокрема `num_results=5`, для контролю над обсягом даних, що подаються LLM. Далі інструмент виконує

пошуковий запит і автоматично агрегує результати, об'єднуючи їх у єдиний рядок, де кожен сніпет відокремлений символом нового рядка. Фінальний, концентрований текстовий блок із  $K$  найбільш релевантних результатів (Observation) повертається до TTMP.

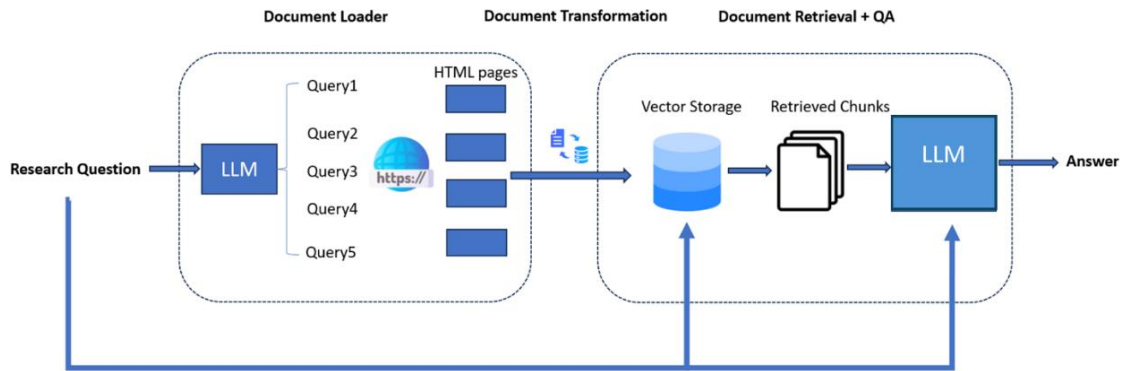


Рис. 3.2. Схема, що демонструє LLM з функцією WebSearch

На рисунку 3.2 наведено схему WebSearch, що виконує критичну роль у процесі Reasoning агента. Коли Router Model класифікує запит як THINKING\_TOOL\_USE, TTMP може вирішити використовувати WebSearch у сценаріях, коли необхідна валідація інформації, що суперечить внутрішнім знанням LLM, або для отримання часово-залежної інформації, яка вийшла після дати навчання LLM. WebSearch також може бути використаний як альтернатива або доповнення до внутрішнього search\_knowledge\_base. Таким чином, розроблений WebSearch Tool є високоефективним, ізольованим мікросервісом, що забезпечує Noodle AI динамічним доступом до актуальної інформації, що значно підвищує якість та надійність відповідей на комплексні запити, реалізуючи принцип "живих даних" у навчальній підтримці.

### 3.3 Аналіз вимог та інтеграція сервісу розкладу

Schedule Service є одним із критично важливих інструментів для Thinking & Tool Use Model Proxy (TTMP), оскільки він забезпечує доступ до динамічних,

персоналізованих даних про навчальний графік. Розробка цього сервісу є прикладом створення адаптера для інтеграції застарілих або нестандартизованих зовнішніх систем, що є типовим завданням в архітектурі підприємства.

Основна функціональна вимога полягає у наданні агенту актуальної інформації про розклад груп та викладачів, яка є життєво необхідною для організації навчального процесу та персоналізованої підтримки.

- Нестандартизоване джерело: зовнішній API (наприклад, <https://dekanat.nung.edu.ua/cgi-bin/timetable.cgi>) має жорстку, нестандартизовану структуру URL-адрес (?n=701&lev=141), що ускладнює пряму взаємодію з TTMP. Прямий виклик такого API агентом був би складним і ненадійним.
- Проблема кодування: зовнішній сервіс, імовірно, використовує застаріле кодування (наприклад, windows-1251), що вимагає спеціальної обробки даних перед їх подачею у внутрішню систему або LLM.
- Необхідність структуризації: відповіді зовнішнього сервісу, найімовірніше, мають неструктурований формат (HTML або неформатований текст), тоді як TTMP вимагає чистий JSON або структурований текст.

Парсинг даних відбувається за допомогою бібліотеки Jsoup, яка використовується для роботи з HTML-документами. Основний процес парсингу можна побачити у класі Schedule.

Формування HTTP-запиту: метод Jsoup.connect() створює з'єднання з URL-адресою, яка визначена в константі URL. До запиту додаються параметри (sdate, edate, teacher, group) через метод .data().

Встановлення кодування: метод postDataCharset(CHARSET.name()) задає кодування для запиту (використовується кодування windows-1251, визначене в TimetableFeignConfig).

Відправка запиту: метод .post() виконує HTTP POST-запит і повертає HTML-документ у вигляді об'єкта Document.

Обробка винятків: якщо виникає помилка під час виконання запиту, вона логуються через log.error.

Ось основний фрагмент коду, який відповідає за парсинг:

### Лістинг 3.1. Код для парсингу

```
var connection = Jsoup.connect(URL + "?n=700")
    .data("sdate", Optional.ofNullable(startDate).map(FORMATTER::format).orElse(""))
    .data("edate", Optional.ofNullable(endDate).map(FORMATTER::format).orElse(""));
if (notEmpty(teacher))
    connection.data("teacher", teacher);
if (notEmpty(group))
    connection.data("group", group);
try {
    return connection.postDataCharset(CHARSET.name()).post();
} catch (IOException e) {
    log.error("Unable to get the requested schedule", e);
    return null;
}
```

Для вирішення цих викликів спроектовано та реалізовано мікросервіс-адаптер Schedule Service (на основі Spring Framework, як видно з коду), який діє як посередник.

- Інтерфейс TimetableClient використовує FeignClient для абстрагування взаємодії із зовнішнім API ([https://dekanat.nung.edu.ua/...](https://dekanat.nung.edu.ua/)).
- У цьому інтерфейсі нестандартизовані URL-адреси (?n=701&lev=141) та параметри кодування інкапсульовані у двох чистих методах: searchTeachers(String query) та searchGroups(String query).

Обробка кодування та медіа-типів:

- Клас TimetableFeignConfig є критичним для забезпечення сумісності. Тут визначається та підтримується кодування windows-1251 (CHARSET) та медіа-тип (MediaType.TEXT\_HTML), що дозволяє коректно декодувати відповіді від зовнішнього сервісу.
- Використання методу String HTTP Message Converter та MappingJackson2HttpMessageConverter забезпечує, щоб відповідь була коректно перетворена в об'єкт QueryResponseDto.

Розробка внутрішнього API:

- GroupController реалізує чистий, стандартизований внутрішній API `/api/v1/groups` та `/api/v1/groups/search`.
- Це дозволяє TTMP (через Schedule Tool) викликати сервіс за чистими, передбачуваними URL-адресами, не знаючи про складнощі зовнішнього API.
- Логіка GroupService (яка використовує FeignClient) перекладає внутрішній запит на зовнішній нестандартизований формат.

Інтегрований Schedule Service перетворюється на Tool у TTMP, дозволяючи агенту:

- Диспетчеризація: коли Router Model позначає запит як `THINKING_TOOL_USE`, TTMP використовує Schedule Tool для запитів на кшталт "Який розклад у групи ПМ-24-2?" або "Чи є пари у професора Н. за розкладом?".
- Форматування виводу: сервіс гарантує, що дані, отримані від зовнішнього джерела (навіть якщо вони були в `windows-1251`), будуть представлені TTMP у чистому та зрозумілому LLM-форматі (JSON/String), що дозволяє агенту сформулювати фінальну, точну відповідь.
- Обробка Помилки (НФВ): клас `ControllerAdvice` забезпечує централізовану обробку помилок (наприклад, `IllegalArgumentException` як `INPUT_ERROR` або загальні помилки як `SERVER_ERROR`), повертаючи TTMP зрозумілі коди стану. Це захищає LLM від необроблених винятків і підвищує надійність системи.

Таким чином, розроблений Schedule Service виконує роль критичного адаптера, ізолюючи основну агентну логіку від складнощів зовнішнього середовища та забезпечуючи Noodle AI актуальними даними розкладу для виконання персоналізованих запитів.

Для використання сервісу розкладу використано Docker контейнер. Контейнеризація – це технологія пакування програмного забезпечення з усіма його залежностями (бібліотеки, конфігурації, файли) в ізольований, самодостатній пакет (контейнер), що дозволяє програмі працювати однаково на будь-якій системі,

незалежно від її операційної системи чи середовища, вирішуючи проблеми несумісності та спрощуючи розгортання.

Для створення контейнеру та поміщення туди сервісу необхідно зробити наступні основні кроки для контейнеризації Java-додатка:

Створення Dockerfile: у кореневій директорії проєкту створюється файл Dockerfile, який описує, як створити образ контейнера

```
# Використання базового образу з JDK
```

```
FROM openjdk:17-jdk-slim
```

```
# Встановлення робочої директорії
```

```
WORKDIR /app
```

```
# Копіювання файлів проєкту
```

```
COPY target/schedule-service.jar app.jar
```

```
# Вказання команди для запуску додатка
```

```
ENTRYPOINT ["java", "-jar", "app.jar"]
```

- Збірка JAR-файлу: використовується команда Maven для створення JAR-файлу: `mvn clean package`
- Створення Docker-образу: `build image` (рис. 3.3) та `compose` (Рис. 3.4) `docker build -t schedule-service`

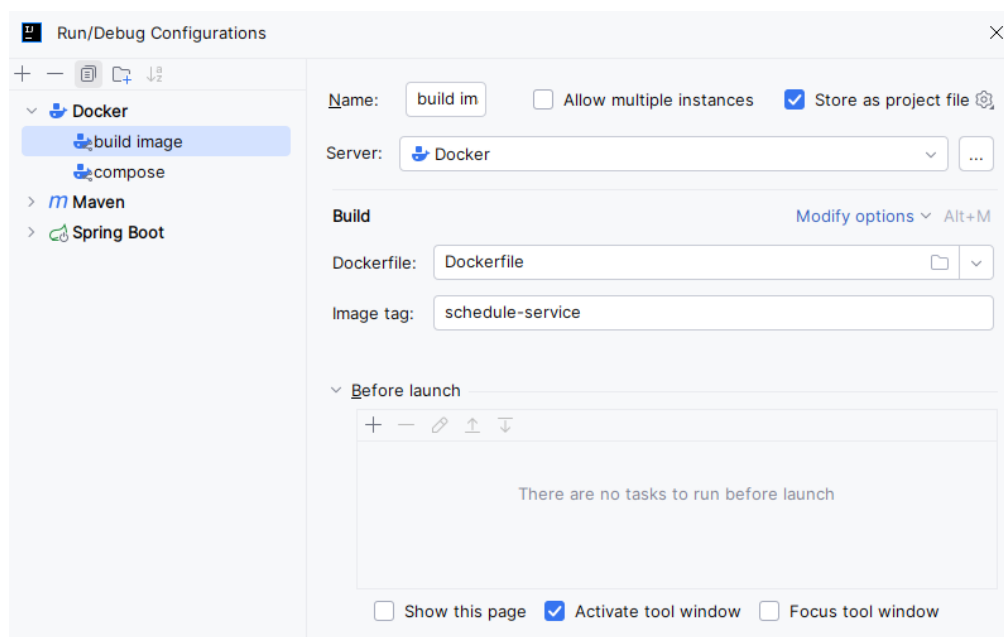


Рис. 3.3. Конфігурація для build docker image

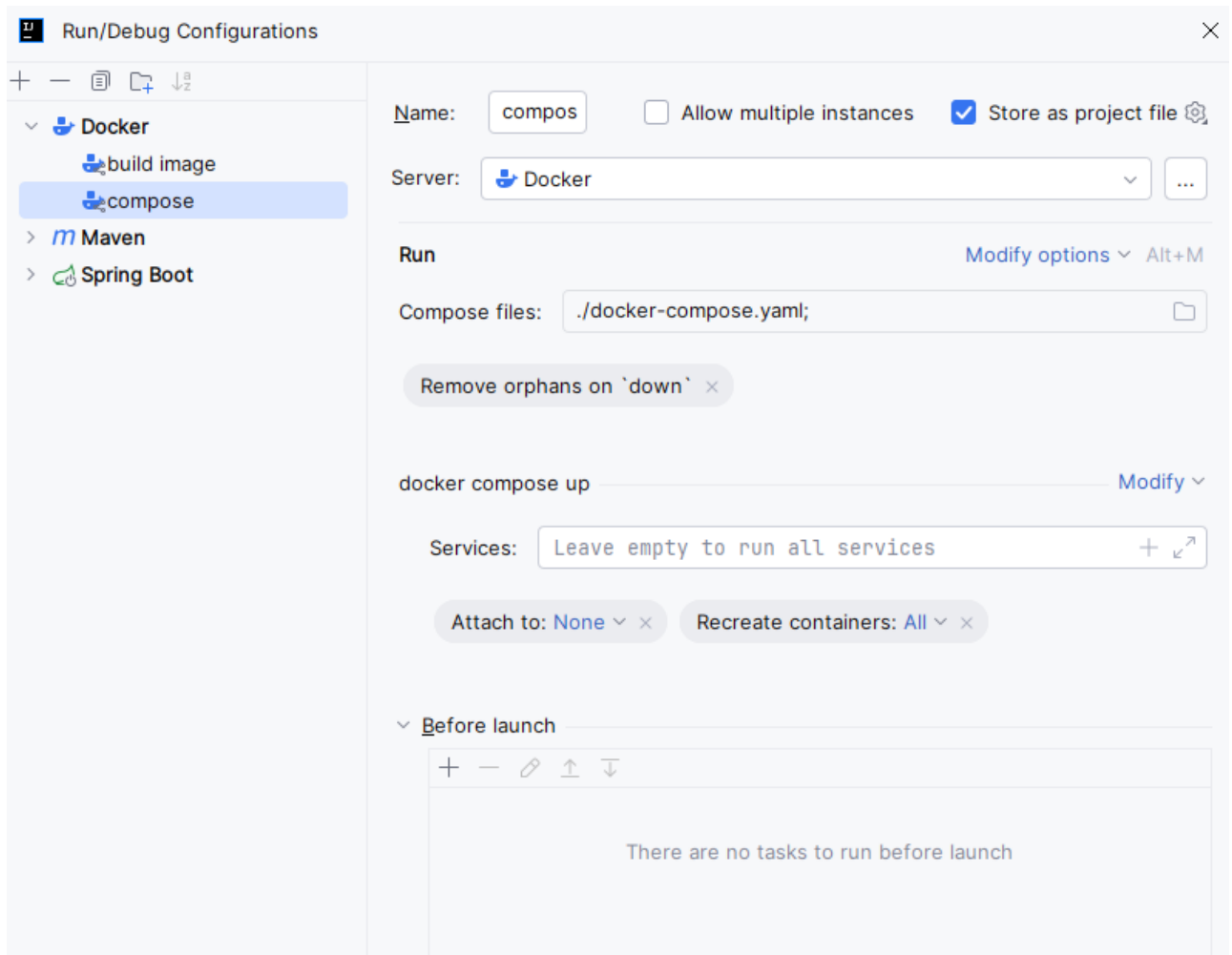


Рис. 3.4. Конфігурація для docker compose

- Запуск контейнера: `docker run -p 8080:8080 schedule-service`. Бачимо наступні логи (рис. 3.5):

```

:: Spring Boot ::                (v3.5.7)

2025-12-07T12:16:16.147Z INFO 1 --- [main] c.m.n.schedule.ScheduleApplication : Starting ScheduleApplication v0.0.1-SNAPSHOT using Java 25.0.1 with PID 1 (/app.jar started by root in /)
2025-12-07T12:16:16.149Z INFO 1 --- [main] c.m.n.schedule.ScheduleApplication : No active profile set, falling back to 1 default profile: "default"
2025-12-07T12:16:16.847Z INFO 1 --- [main] o.s.c.cloud.context.scope.GenericScope : BeanFactory id=Defab3fd-5e8d-30a2-980c-531b2ee61308
2025-12-07T12:16:17.060Z INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8101 (http)
2025-12-07T12:16:17.072Z INFO 1 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-12-07T12:16:17.073Z INFO 1 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.48]
2025-12-07T12:16:17.092Z INFO 1 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-12-07T12:16:17.092Z INFO 1 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 906 ms
2025-12-07T12:16:17.789Z INFO 1 --- [main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 2 endpoints beneath base path '/actuator'
2025-12-07T12:16:17.842Z INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8101 (http) with context path '/'
2025-12-07T12:16:17.854Z INFO 1 --- [main] c.m.n.schedule.ScheduleApplication : Started ScheduleApplication in 2.098 seconds (process running for 2.553)
2025-12-07T12:16:08.640Z INFO 1 --- [nio-8101-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-12-07T12:16:08.641Z INFO 1 --- [nio-8101-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-12-07T12:16:08.642Z INFO 1 --- [nio-8101-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2025-12-07T12:16:08.684Z INFO 1 --- [nio-8101-exec-1] c.m.n.s.controller.ScheduleController : Received request to get schedule for teacher null or group IIM-24-3 in range 2025-10-13-2025-10-13
2025-12-07T12:16:08.719Z INFO 1 --- [nio-8101-exec-1] c.m.n.s.controller.ScheduleController : Returning 1 days, 2 rows, 2 sessions
2025-12-07T12:41:26.252Z INFO 1 --- [nio-8101-exec-3] c.m.n.s.controller.ScheduleController : Received request to get schedule for teacher null or group IIM-24-3 in range 2025-10-12-2025-10-12
2025-12-07T12:41:27.070Z INFO 1 --- [nio-8101-exec-3] c.m.n.s.controller.ScheduleController : Returning 0 days, 0 rows, 0 sessions

```

Рис. 3.5. Логи з консолі Docker про запуск контейнера

Тепер ми отримаємо можливість взаємодіяти з сервісом. Ось приклад результату (рис. 3.6):

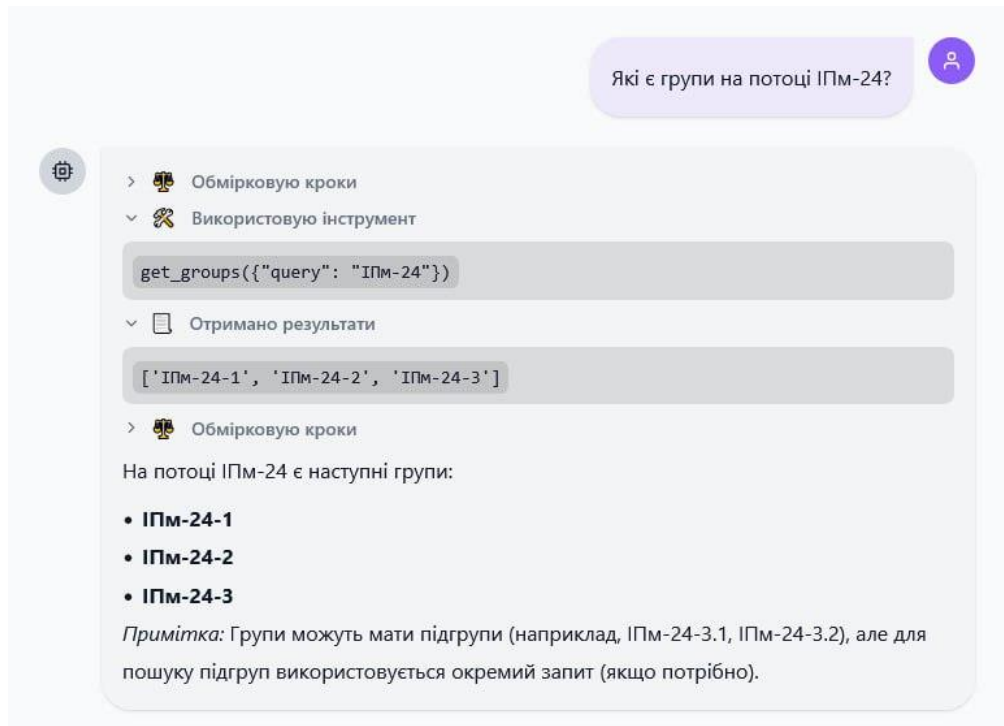


Рис. 3.6. Приклад взаємодії з розкладом через ШІ-асистента

### 3.4 Обґрунтування застосування Model Context Protocol (MCP) для взаємодії з Moodle MCP Server

У рамках проєкту Noodle AI взаємодія міркувального агента (Thinking & Tool Use Model Proxy, TTMP) з критично важливою освітньою платформою LMS Moodle вимагає не просто технічного підключення, а стратегічного архітектурного рішення, здатного забезпечити безпеку, стандартизацію та контекстну релевантність даних. Таким рішенням є впровадження спеціалізованого адаптера - Moodle MCP Server, який працює на основі Model Context Protocol (MCP).

Прямий доступ LLM-агента до традиційного Moodle Web Service API стикається з трьома основними проблемами:

- складність автентифікації та безпеки: прямий доступ вимагає передачі токенів авторизації Moodle безпосередньо в контекст LLM, що створює високий ризик

безпеки. Крім того, LLM-агент не має вбудованих механізмів для керування сесіями та правами доступу, що є критичним для роботи з персоналізованими даними (оцінки, профілі студентів);

- неефективність LLM: Moodle API повертає великі, надлишкові масиви даних (надлишковий JSON), які вимагають значної кількості токенів LLM для парсингу та фільтрації. Це підвищує експлуатаційні витрати та знижує швидкість міркування;
- відсутність контекстної релевантності: традиційний API не розуміє освітнього контексту. Наприклад, агент повинен сам знати, який саме `wsfunction` Moodle викликати для отримання "оцінок з останнього курсу", тоді як LLM краще працює з семантичними командами.

Model Context Protocol (MCP) - це архітектурний шаблон, що стандартизує інтерфейс взаємодії між інтелектуальними моделями та зовнішніми сервісами. MCP вирішує вищезазначені проблеми шляхом створення проміжного сервісу (Moodle MCP Server), який виконує роль "перекладача" та "охоронця" даних:

- ізоляція та безпека: Moodle MCP Server інкапсулює API-токен Moodle та обробляє логіку автентифікації. TTMP взаємодіє лише з MCP Server, а не з Moodle напряму. Це забезпечує ізоляцію критичних облікових даних від LLM-контексту, значно підвищуючи безпеку роботи з персоналізованою інформацією;
- контекстна фільтрація та агрегація: MCP Server виконує функцію попередньої обробки даних. Він приймає від TTMP чіткий запит (наприклад, "get\_grades"), викликає необхідні Moodle `wsfunction` та фільтрує надлишкову інформацію, повертаючи агенту лише концентрований, релевантний JSON-фрагмент. Це знижує токенове навантаження на TTMP і покращує якість його фінальної відповіді;
- стандартизований інтерфейс Tool Use: замість того, щоб навчати LLM десяткам специфічних Moodle `wsfunction`, TTMP взаємодіє з єдиним, стандартизованим інструментом (`moodle_integration`), що інкапсулює

всю складність Moodle API. TTMP передає MCP Server лише команду (**action**) та необхідні параметри (**params**), що спрощує логіку міркування агента;

- еволюційна стійкість: якщо Moodle оновлює свій API, потрібно змінити лише логіку Moodle MCP Server, тоді як інтерфейс, з яким взаємодіє LLM залишається незмінним, забезпечуючи стійкість архітектури.

Таким чином, застосування Model Context Protocol та реалізація Moodle MCP Server є стратегічним вибором, який перетворює складну інтеграцію на безпечний, еконий та ефективний канал обміну контекстними даними, що є необхідною умовою для якісної персоналізації навчального процесу в системі Moodle AI.

### 3.5 Висновки до розділу

В цьому розділі виконано проектування та розробку методологічної та алгоритмічної основи системи Moodle AI, що відповідає за керування потоком інформації та інтеграцію зовнішніх освітніх платформ. У результаті сформовано надійний технічний план для подальшої програмної реалізації.

Розроблено алгоритм семантичної маршрутизації (Router Model): створено алгоритм, реалізований у Router Model Proxy, який використовує LLM-класифікатор з низькою температурою ( $T=0.0$ ) для детермінованого розмежування запитів на класи GENERIC та THINKING\_TOOL\_USE. Це забезпечує економію ресурсів шляхом перенаправлення простих запитів до дешевого Generic Model Proxy та підвищує швидкість обробки.

Визначено вимоги та функціонал проксі-моделей. Generic Model Proxy (GMP): спроектовано "інформаційне ядро" з акцентом на швидкості. Thinking & Tool Use Model Proxy (TTMP): - здатний до багатоетапного Tool Calling та візуалізації процесу міркування клієнту.

Спроектовано та інтегровано адаптери динамічних даних та визначено функціональні вимоги до TTMP – WebSearch, Schedule Service Tool.

## РОЗДІЛ 4

### РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ

#### 4.1 Архітектура та стек технологій реалізації серверної частини системи

Для забезпечення вимог гнучкості, масштабованості та ізоляції компонентів, сформульованих у попередніх розділах, програмна реалізація системи Noodle AI виконана на основі мікросервісної архітектури. Система складається з набору незалежних сервісів, що взаємодіють через протокол HTTP (REST API) та асинхронні потоки подій (Server-Sent Events - SSE).

Розглянемо технологічний стек серверної частини:

- мова програмування Python 3.10+: використана для реалізації ядра інтелектуальної обробки (Router, Generic, TTMP) завдяки широкій підтримці бібліотек для роботи зі штучним інтелектом;
- web-фреймворк FastAPI: вибраний як основний каркас для мікросервісів-проксі. Його асинхронна природа (async/await) дозволяє ефективно обробляти довготривалі з'єднання, необхідні для стрімінгу відповідей від LLM, мінімізуючи блокування потоків;
- мова програмування Java 17 та Spring Boot: використана для реалізації Schedule Service. Вибір обумовлений надійністю екосистеми Spring для побудови Enterprise-адаптерів та наявністю потужних інструментів (Feign Client) для інтеграції з застарілими системами;
- orchestration framework LangChain: використаний у Thinking & Tool Use Model Проху для керування логікою агента, ланцюжками міркувань (CoT) та викликом інструментів (Tool Calling);
- LLM-провайдер: реалізація підтримує роботу як з хмарними моделями (OpenAI GPT-4o), так і з локальними моделями (через LM Studio), завдяки дотриманню OpenAI-сумісних контрактів API;
- бібліотека Pydantic: використовується для валідації даних та опису DTO (Data Transfer Objects), що гарантує сувору типізацію запитів між мікросервісами.

Така гетерогенна архітектура дозволяє використовувати найкращі інструменти для кожного конкретного завдання: Python для AI-логіки та Java для стабільної інтеграції з корпоративними системами університету.

## 4.2 Програмна реалізація проксі-сервісів: Router, Generic, Thinking

Модель маршрутизації (Router Model) є фундаментальним компонентом архітектури асистента Noodle AI, призначеним для інтелектуального спрямування вхідних запитів користувачів до найбільш відповідного модуля внутрішньої обробки. Її основна функція полягає в аналізі наміру (intent) та складності запиту користувача для прийняття рішення: чи повинен він оброблятися спеціалізованою Thinking & Tool Use Model Proxy (для складних завдань, що вимагають зовнішніх інтеграцій), чи більш прямою Generic Model Proxy (для простих розмовних відповідей, що не потребують міркування).

Критично важливою особливістю реалізації є стандартизація патерну взаємодії. Router Model діє як проксі, використовуючи специфікацію API, сумісну з OpenAI, для диспетчеризації запитів до цільової моделі. Це значно спрощує інтеграцію та масштабування системи, дозволяючи клієнтському інтерфейсу (UI) взаємодіяти з єдиною точкою входу.

Цей механізм інтелектуальної маршрутизації є вирішальним для оптимізації використання ресурсів, покращення часу відгуку (latency) та забезпечення того, щоб складні запити оброблялися з необхідним рівнем міркування та оркестрації інструментів, тоді як прості запити отримували швидкі та прямі відповіді через Generic Model.

По суті, Router Model діє як високотехнологічний диспетчер трафіку для взаємодій з користувачем. Програмна реалізація цього процесу включає наступні етапи:

- прийом запиту (query reception). Модель отримує вхідні дані від користувача разом із контекстом. У нашій реалізації на FastAPI це відбувається через єдиний endpoint /chat/completions;

## Лістинг 4.1. Код для Query Reception

```

54 @router.post("/chat/completions")
55 async def create_chat_completion(
56     request: ChatCompletionRequest,
57     router_service: RouterService = Depends(get_router_service)
58 ):
59     logger.debug(f"Router chat completion request received. Model: {request.model}, Messages: {[msg.model_dump_json() for msg in request.messages]}")
60
61     if request.stream:
62         logger.debug(f"Streaming router chat completion request received. Model: {request.model}")
63         return StreamingResponse(
64             stream_generator(router_service, request),
65             media_type="text/event-stream"
66         )

```

- аналіз наміру (Intent Analysis). Модель виконує аналіз запиту для розуміння прихованого наміру користувача. Це включає розуміння природної мови (NLU) та порівняння поточного запиту з відомими семантичними патернами. У програмному коді ми використовуємо спеціалізовану LLM з нульовою температурою для детермінованої класифікації;

## Лістинг 4.2. Код для Intent Analysis

```

35 async def _determine_route(self, query: str) -> str:
36     """
37     Uses the router LLM to determine if the query is 'GENERIC' or 'THINKING_TOOL_USE',
38     considering only the current query.
39     """
40     logger.debug(f"Determining route for query: '{query}'.")
41     try:
42         response = await self.router_chain.ainvoke({"query": query})
43         decision = response.content.strip().upper()
44         if decision not in ["GENERIC", "THINKING_TOOL_USE"]:
45             logger.warning(
46                 f"Router model returned an unexpected decision: '{response.content}', defaulting to GENERIC.")
47             return "GENERIC"
48         logger.debug(f"Route determined: {decision}")
49         return decision
50     except Exception as e:
51         logger.error(f"Error determining route for query '{query}': {e}")
52         raise
53

```

- прийняття рішення про маршрут (Route Decision). На основі аналізу Router Model приймає імовірнісне рішення щодо того, який модуль найкраще підходить для обробки: До Thinking & Tool Use Model Proxy: якщо запит передбачає потребу в отриманні зовнішньої інформації (наприклад, "Який у мене розклад на завтра?", "Хто такий професор Коваленко?"), маніпуляції даними (наприклад, "Створи курс у Moodle") або складному багатоетапному

міркуванні. Усі запити RAG (Retrieval-Augmented Generation) також спрямовуються сюди. До Generic Model Proxy: якщо запит є розмовним, базується на загальних знаннях або може бути вирішений без виклику зовнішніх інструментів;

#### Лістинг 4.3. Код для Route Decision

```

67     if route_decision == "GENERIC":
68         target_proxy = GENERIC_MODEL_PROXY_URL
69     elif route_decision == "THINKING_TOOL_USE":
70         target_proxy = THINKING_TOOL_MODEL_PROXY_URL
71     else:
72         logger.warning(f"Unexpected route decision '{route_decision}', defaulting to Generic Proxy.")
73         target_proxy = GENERIC_MODEL_PROXY_URL

```

- пересилання запиту (query forwarding). Після вибору маршруту роутер пересилає оригінальний запит до API-endpoint'a обраного модуля. Завдяки стандартизації під OpenAI API, це виглядає як прозоре проксіювання.

#### Лістинг 4.4. Код для query forwarding

```

95     target_url = f"{target_proxy}/chat/completions"
96     logger.debug(f"Proxying request to {target_url}. Decision: {route_decision}")
97     return await self._proxy_request(target_url, proxy_request)

```

Обробка відповіді (Response Handling). Відповідь від обраного модуля повертається роутеру (або безпосередньо UI через стрімінг) і відображається користувачеві.

Реалізація Router Model базується на трьох принципах:

- розв'язка (decoupling): Router відокремлює UI від конкретних реалізацій LLM та інструментів. Це дозволяє розробляти компоненти незалежно один від одного;
- розширюваність (extensibility): Нові інструменти або LLM можуть бути додані в систему шляхом оновлення конфігурації роутера, не впливаючи на існуючі компоненти;

- ефективність (efficiency): Спрямування простих запитів до легковажних моделей (Generic Proxy) покращує загальну латентність системи та економічну ефективність, залишаючи ресурсомісткі моделі (Thinking Proxy) тільки для завдань, де вони дійсно необхідні.

Generic Model Proxy (GMP) спроектовано як легкий, високопродуктивний мікросервіс, завданням якого є обслуговування масових інформаційних запитів, що не вимагають складного міркування. Сервіс реалізовано на базі фреймворку FastAPI, що забезпечує асинхронну обробку HTTP-запитів.

Ініціалізація та ін'єкція системної ролі. Основна логіка взаємодії з мовною моделлю інкапсульована у класі LLMService. Ключовою архітектурною особливістю є примусова інжекція системного промπτу. У файлі конфігурації config.py визначено змінну GENERIC\_SYSTEM\_PROMPT, яка містить інструкції для моделі (наприклад, "Ти - корисний помічник...").

При кожному виклику методу `get_chat_completion` або `get_chat_completion_stream`, системне повідомлення автоматично додається на початок списку повідомлень. Це гарантує, що модель завжди залишається в межах заданої ролі, навіть якщо користувач намагається змінити її поведінку через свій запит ("jailbreak").

#### Лістинг 4.5. Код для промπτу в Generic Model

```

async def get_chat_completion(self, messages: List[Message], temperature: float = 0.7) -> (str, dict):
    # Ensure the system prompt is always the first message
    langchain_messages: list[BaseMessage] = [SystemMessage(content=SYSTEM_PROMPT)]

    for msg in messages:
        if msg.role == "user":
            langchain_messages.append(HumanMessage(content=msg.content))
        elif msg.role == "assistant":
            langchain_messages.append(AIMessage(content=msg.content))
        # Ignore other roles for now, or raise an error

    self.llm.temperature = temperature

```

Реалізація асинхронного стрімінгу (SSE). Для забезпечення інтерактивності, GMP підтримує технологію Server-Sent Events (SSE). Це реалізовано через ендпоінт

/chat/completions, який аналізує прапорець stream у тілі запиту. Якщо stream=True, повертається об'єкт StreamingResponse.

Функція-генератор stream\_generator асинхронно ітерує по чанках (шматках тексту), які надходять від LLM, і "на льоту" формує стандартизовані JSON-пакети. Це дозволяє клієнтському інтерфейсу відображати відповідь посимвольно, мінімізуючи відчуття затримки (Time to First Token).

#### Лістинг 4.6. Код для Stream Generator

```

27  async def stream_generator(llm_service: LLMService, request: ChatCompletionRequest):
28      completion_id = f"chatcpl-{uuid.uuid4()}"
29
30      # Асинхронний цикл отримання чанків від LLM
31      async for chunk in llm_service.get_chat_completion_stream(
32          messages=request.messages,
33          temperature=request.temperature
34      ):
35          delta = DeltaMessage()
36          if chunk.content:
37              delta.content = chunk.content
38
39          # Формування DTO відповіді для поточного фрагменту
40          choice = ChatCompletionStreamResponseChoice(index=0, delta=delta, finish_reason=None)
41          stream_response = ChatCompletionStreamResponse(
42              id=completion_id,
43              model=request.model,
44              choices=[choice]
45          )
46          # Відправка даних у форматі 'text/event-stream'
47          yield f"data: {stream_response.model_dump_json()}\n\n"
48
49      # Сигнал завершення потоку
50      yield "data: [DONE]\n\n"

```

Обробка метаданих та usage-статистика. Для запитів без стрімінгу (stream=False) реалізовано механізм розрахунку використаних ресурсів. Хоча у поточній версії використовується спрощений підрахунок токенів (через split()), архітектура передбачає повернення об'єкта ChatCompletionResponseUsage. Це критично важливо для подальшого бізнес-аналізу та моніторингу витрат на використання платних API.

Валідація даних та обробка помилок. Використання бібліотеки Pydantic (chat.py) забезпечує сувору валідацію вхідних даних (ChatCompletionRequest). Будь-

яка невідповідність формату запиту автоматично перехоплюється FastAPI, що повертає статус 422 Unprocessable Entity. Додатково реалізовано блоки try...except для перехоплення помилок генерації (ValueError, Exception) та їх коректного логування через централізований logger. Це підвищує відмовостійкість сервісу та спрощує налагодження.

На відміну від Generic Proxy, який діє як простий ретранслятор, сервіс TTMP реалізовано як повноцінний інтелектуальний агент. Програмне ядро базується на бібліотеці LangChain, що дозволяє реалізувати патерн ReAct (Reasoning and Acting). Сервіс не просто генерує текст, а виконує цикл "Думка -> Дія -> Спостереження -> Відповідь".

Ініціалізація агента та реєстрація інструментів. У класі LLMService відбувається конфігурація агента. Ключовим моментом є використання функції create\_openai\_tools\_agent, яка зв'язує мовну модель (LLM) з набором інструментів (all\_tools) та спеціалізованим промптом.

Промпт агента (ChatPromptTemplate) містить критично важливі плейсхолдери:

- chat\_history: для збереження контексту діалогу.
- agent\_scratchpad: робоча область пам'яті, де агент зберігає проміжні кроки міркування та результати виконання інструментів.

#### Лістинг 4.7. Код класу LLMService для Thinking Model

```
class LLMService:
    """ MikronT """
    def __init__(self):
        # ... конфігурація API ...

        # Створення шаблону промту з підтримкою історії та "чернетки" агента
        self.prompt = ChatPromptTemplate.from_messages([
            SystemMessage(SYSTEM_PROMPT),
            MessagesPlaceholder("chat_history"),
            ("user", "{input}"),
            MessagesPlaceholder("agent_scratchpad"), # Місце для Tool Outputs
        ])

        # Ініціалізація виконавця агента (AgentExecutor)
        # all_tools містить функції: web_search, get_schedule, moodle_integration
        self.agent = create_openai_tools_agent(self.llm, all_tools, self.prompt)

        # Executor керує циклом "Think-Act-Observe"
        self.agent_executor = AgentExecutor(agent=self.agent, tools=all_tools, verbose=True)
```

Обробка подій та трансформація потоку (Event Stream Transformation). Найскладнішою частиною реалізації є метод `get_chat_completion_stream` та обробник подій `handle_stream_event`. Оскільки агент генерує не тільки текст, а й "події" (наприклад, "я починаю шукати в Google"), ці внутрішні події LangChain необхідно трансформувати у формат, зрозумілий клієнту. Метод `astream_events` дозволяє підписатися на внутрішні процеси агента. Обробник `handle_stream_event` фільтрує ці події та конвертує їх у JSON-структури з типом події (`chunk`, `tool_call`, `tool_start`).

#### Лістинг 4.8. Код для Event Stream Transformation

```

76  async def get_chat_completion_stream(self, messages: List[Message], temperature: float = 0.
77      # ... підготовка історії чату ...
78
79      # Підписка на потік подій виконання агента (version="v1")
80      async for event in self.agent_executor.astream_events(
81          input={"input": user_input, "chat_history": chat_history},
82          version="v1"
83      ):
84          yield self.handle_stream_event(event)
85
1 usage  MikronT *
86  def handle_stream_event(self, event: dict) -> str:
87      event_name = event["event"]
88      message = {"event": event_name}
89
90      # Обробка генерації тексту відповіді
91      if event_name == "on_chat_model_stream":
92          chunk = event["data"].get("chunk")
93          if chunk.content:
94              message["data"] = {"type": "chunk", "content": chunk.content}
95
96      # Обробка початку використання інструменту
97      elif event_name == "on_tool_start":
98          message["data"] = {
99              "type": "tool_start",
100             "tool_name": event["data"].get("name"),
101             "tool_input": event["data"].get("input")
102          }
103
104      # Обробка завершення роботи інструменту (отримання даних)
105      elif event_name == "on_tool_end":
106          tool_output = event["data"].get("output")
107          # Інкапсуляція виводу інструменту у спеціальні теги для UI
108          message["data"] = {
109              "type": "chunk",
110              "content": f"<tool_output>{tool_output}</tool_output>"
111          }
112
113      return json.dumps(message)
114

```

Формування історії діалогу для агента. Для коректної роботи агенту необхідно передавати історію повідомлень у специфічних класах LangChain (`HumanMessage`,

AIMessage). Реалізовано конвертер, який перетворює DTO Message у внутрішні об'єкти фреймворку перед запуском виконавця.

#### Лістинг 4.9. Код формування історії діалогу для агента

```

async def get_chat_completion(self, messages: List[Message], temperature: float = 0.7) -> (str, dict):
    # Convert app messages to LangChain chat history format
    chat_history = []
    user_input = ""
    for msg in messages:
        if msg.role == "system": # System prompt is handled separately in the agent prompt
            continue
        elif msg.role == "user":
            user_input = msg.content # The last user message is the current input
            chat_history.append(HumanMessage(msg.content))
        elif msg.role == "assistant":
            chat_history.append(AIMessage(msg.content))

```

Така реалізація забезпечує повну прозорість дій агента для користувача: клієнтський інтерфейс отримує не лише фінальну відповідь, але й інформацію про те, які інструменти викликаються в реальному часі, що підвищує довіру до системи (Explainable AI).

### 4.3 Розробка алгоритмів роботи системи

Алгоритмічна основа системи Noodle AI базується на патерні семантичної маршрутизації, який забезпечує динамічний розподіл навантаження між різними обчислювальними модулями. Розроблений алгоритм взаємодії описує повний життєвий цикл запиту користувача: від моменту введення тексту в інтерфейсі до отримання фінальної відповіді.

Загальний алгоритм роботи системи, зображений на діаграмі послідовності, можна розділити на чотири ключові етапи: ініціалізація, семантичний аналіз, виконання (execution) та агрегація відповіді.

Етап 1. Ініціалізація взаємодії. Процес розпочинається на стороні клієнта (Chatbot UI) через інтерфейс. Коли користувач вводить запит, інтерфейс формує HTTP-запит:

- система використовує стандартний формат ChatCompletionRequest;

- запит надсилається методом POST на єдину точку входу системи — Router Model Proxy (/chat/completions). Це забезпечує абстракцію клієнта від складної внутрішньої мікросервісної архітектури.

Етап 2. Семантичний аналіз та класифікація. Отримавши запит, Router Model Proxy не намагається сформулювати відповідь самостійно. Його завдання - визначити інтенст (намір) користувача:

- роутер звертається до внутрішньої легкої мовної моделі (Router LLM);
- критично важливим параметром тут є температура генерації  $Temperature=0$ . Це усуває елемент випадковості, роблячи класифікацію детермінованою: один і той самий запит завжди призведе до одного й того ж маршруту;
- Router LLM повертає мітку класу: GENERIC (для загальних запитів) або THINKING\_TOOL\_USE (для запитів, що вимагають інструментів).

Етап 3. Сценарій А: обробка простих запитів (гілка GENERIC). Якщо класифікатор визначив запит як загальний, активується "швидкий шлях" (рис. 4.1).

- проксіювання: Router пересилає запит без змін до Generic Model Proxy;
- генерація: Generic Model генерує пряму текстову відповідь;
- повернення: відповідь повертається роутеру.

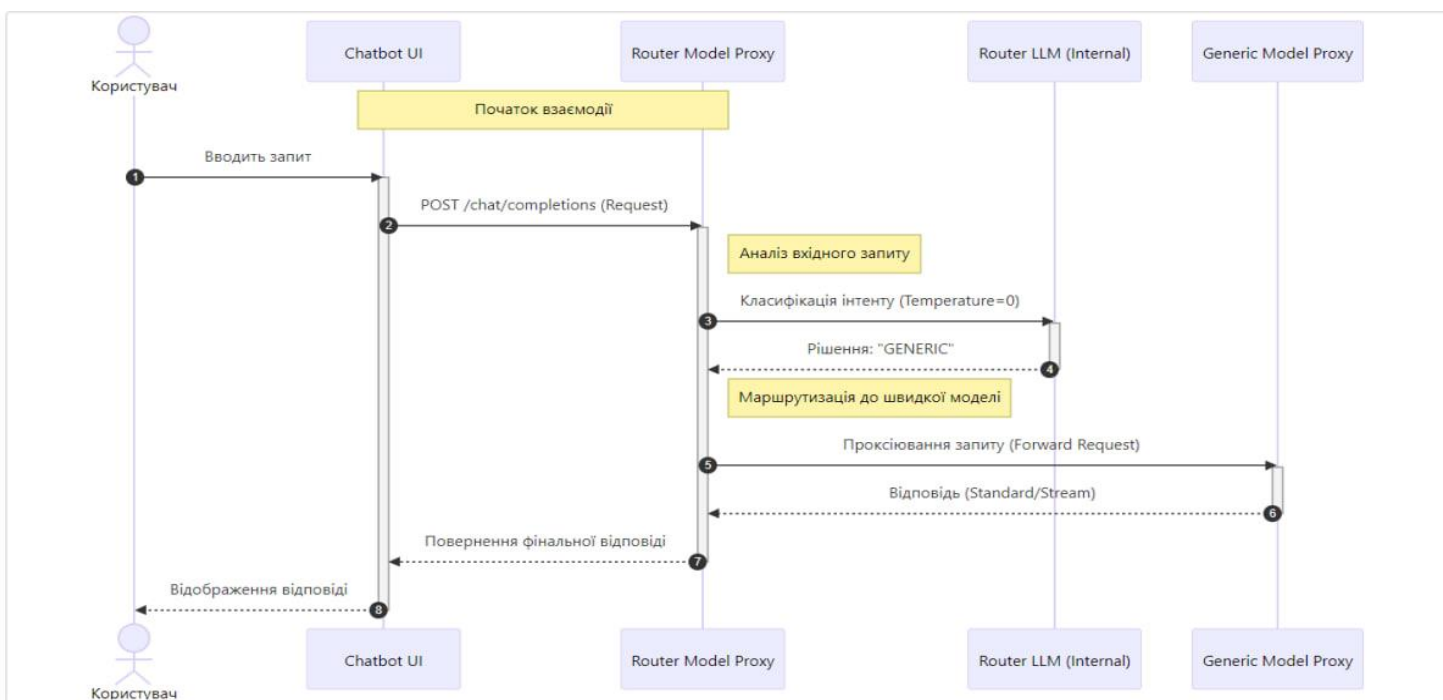


Рис. 4.1. Діаграма послідовності для Generic Model

Сценарій Б: обробка комплексних запитів (гілка THINKING\_TOOL\_USE). Якщо запит вимагає доступу до даних, активується "агентний шлях" (рис. 4.2):

- ініціалізація агента: Запит пересилається до Thinking & Tool Use Model Proxy.
- агентний цикл (Loop Reasoning & Tool Execution). Всередині цього проксі запускається ітеративний процес ReAct (Reasoning + Acting):
  - o планування (Thinking): модель аналізує запит і формує план дій (наприклад: "Мені потрібно знайти розклад, для цього я викличу інструмент Schedule");
  - o виклик інструменту (Action): агент формує запит до зовнішнього інструменту (External Tools), наприклад, до сервісу розкладу (Schedule) або системи управління навчанням (Moodle);
  - o спостереження (Observation): зовнішній інструмент виконує запит і повертає результат у форматі JSON (наприклад, список пар);
  - o аналіз (Analysis): агент отримує дані, аналізує їх і вирішує, чи достатньо інформації для відповіді. Якщо ні — цикл повторюється.

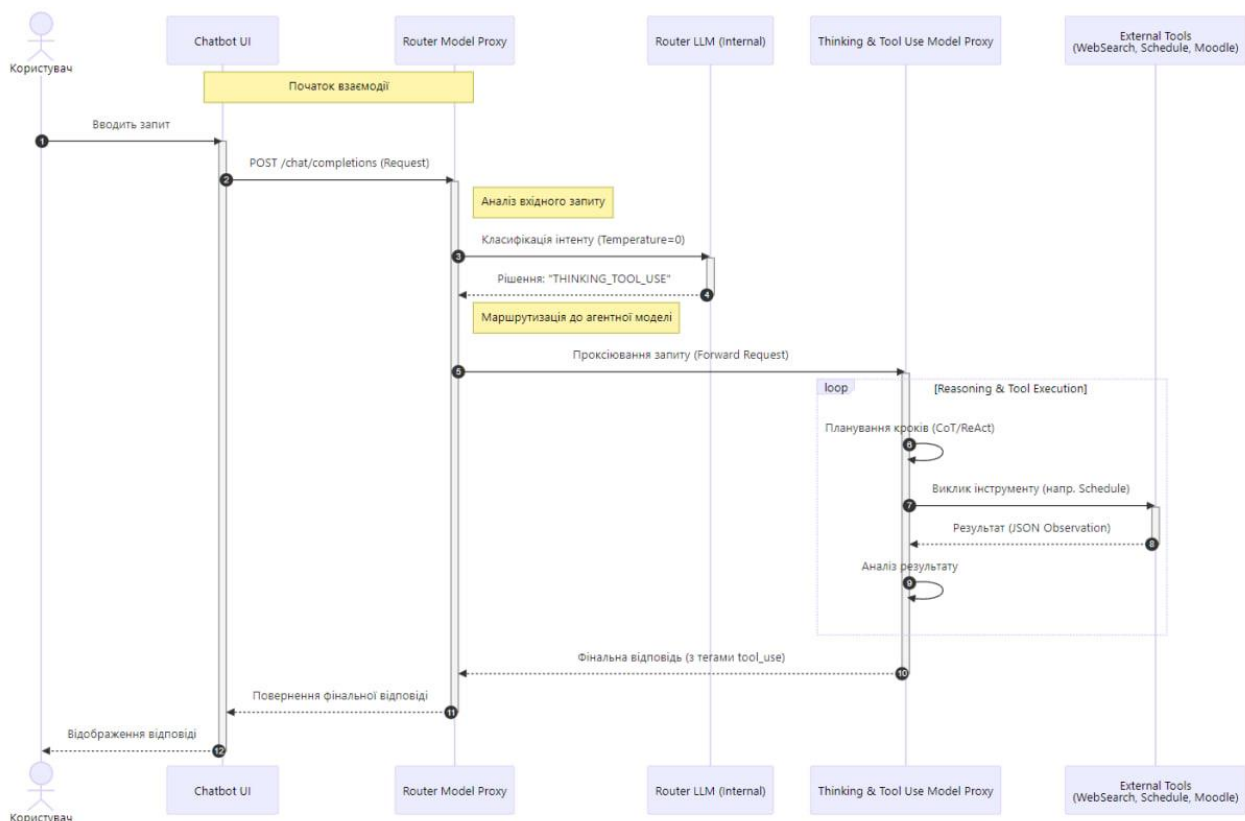


Рис. 4.2. Діаграма послідовності для Thinking Model

Етап 4. Агрегація та відображення (кроки фіналу). Незалежно від обраного шляху (Generic або Thinking), Router Model Proxy отримує фінальну відповідь і транслює її назад до Chatbot UI. Інтерфейс користувача візуалізує текст. У випадку використання Thinking Model, інтерфейс також може візуалізувати проміжні кроки (теги tool\_use), показуючи користувачеві, які саме інструменти були використані (наприклад, "Перевіряю ваш розклад..."). Приклади відображення можна знайти в додатку В.

#### 4.4 Програмна реалізація системи моніторингу використання ресурсів

При генерації ШІ асистентом відповідей на питання ми, очікувано, спостерігаємо значне використання ресурсів системи, адже запускаємо ми LLM локально з використанням графічного процесора, на який відповідно і припадає основне навантаження:

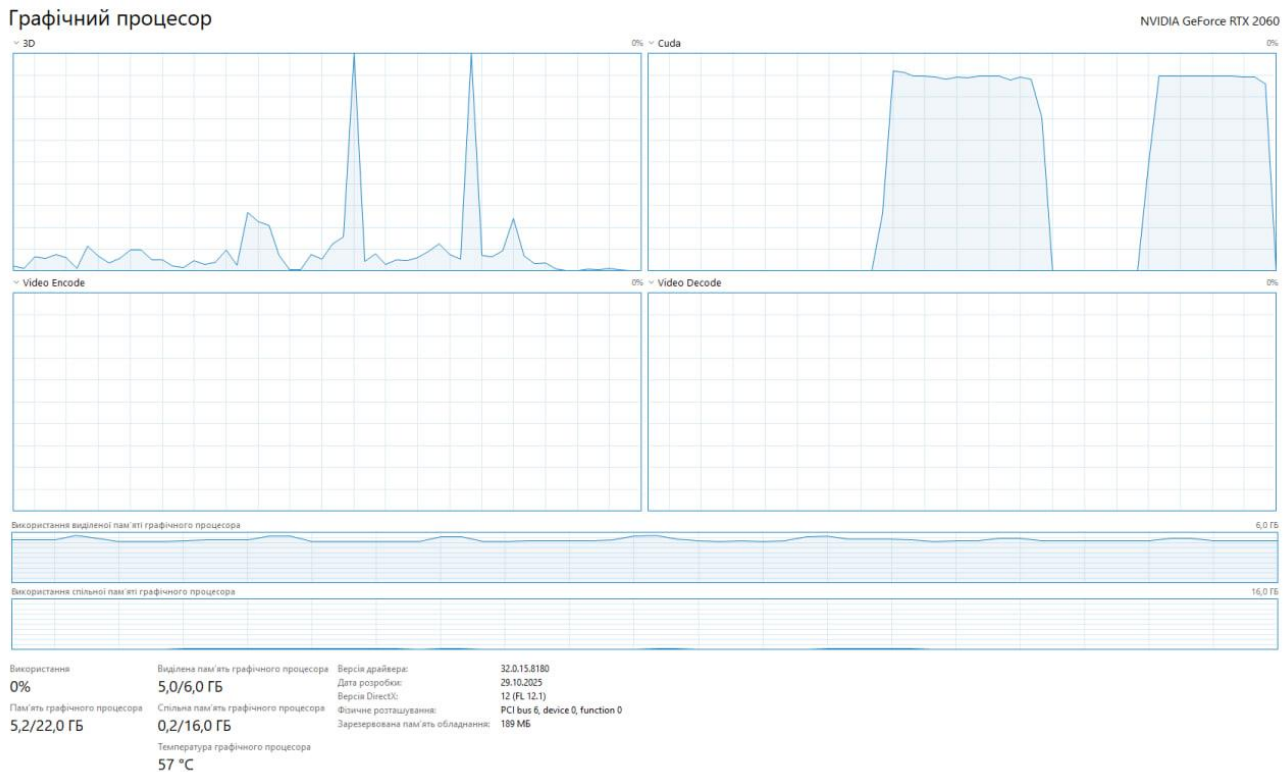


Рис. 4.3. Використання ресурсів комп'ютера при створення відповіді з використанням ШІ

Як бачимо – найбільше використовується графічний процесор та його відеопам'ять (рис. 4.3).

Для забезпечення надійності роботи розподіленої системи Noodle AI та контролю експлуатаційних витрат розроблено програмну підсистему моніторингу. Вона інтегрована безпосередньо в код кожного мікросервісу і вирішує завдання збору телеметрії, відстеження помилок та обліку використання токенів LLM.

Основою системи спостереження є модуль `app.core.logging_config`, який уніфікує формат логів для всіх сервісів (Router, Generic, Thinking). Використання стандартної бібліотеки `logging` дозволяє перехоплювати події на різних рівнях (DEBUG, INFO, ERROR) та виводити їх у стандартний потік виводу (`sys.stdout`), що є стандартом для контейнеризованих застосунків (Docker).

Лістинг 4.10. Налаштування логування

```

4  def setup_logging():
5      logger = logging.getLogger('thinking-tool-model-proxy')
6      logger.setLevel(logging.DEBUG)
7
8      if not logger.handlers:
9          handler = logging.StreamHandler(sys.stdout)
10         formatter = logging.Formatter(
11             '%(asctime)s - %(levelname)s - %(name)s - %(message)s'
12         )
13         handler.setFormatter(formatter)
14         logger.addHandler(handler)
15
16         # Set uvicorn loggers to INFO to avoid too much verbosity, but still get request logs
17         logging.getLogger("uvicorn.access").setLevel(logging.INFO)
18         logging.getLogger("uvicorn.error").setLevel(logging.ERROR)
19
20     return logger
21
22     logger = setup_logging()

```

Такий підхід дозволяє адміністратору системи в реальному часі відстежувати проходження запиту через каскад проксі-сервісів, аналізуючи часові мітки для виявлення затримок (latency bottlenecks).

Критично важливою функцією для бізнес-аналізу є облік витрат ресурсів LLM. Оскільки система використовує як локальні, так і платні моделі (OpenAI API), кожен проксі-сервіс зобов'язаний повертати метрики використання у кожній відповіді, щоб оптимізувати витрати.

У контролерах сервісів (`create_chat_completion`) реалізовано алгоритм підрахунку токенів. У поточній реалізації використовується евристичний метод підрахунку (на основі слів) або отримання даних від провайдера, які інкапсулюються в об'єкт `ChatCompletionResponseUsage`.

Фрагмент реалізації у `RouterService` та проксі-сервісах:

#### Лістинг 4.11. Логування статистики

```
# Log stream statistics after completion
duration = time.time() - start_time
logger.info(
    f"Streaming chat completion statistics. ID: {completion_id}, Model: {request.model}, "
    f"Prompt Tokens: {cb.prompt_tokens}, Completion Tokens: {cb.completion_tokens}, "
    f"Total Tokens: {cb.total_tokens}, Duration: {duration:.4f}s"
)
```

Для відлагодження помилок у розподіленій системі реалізовано механізм ідентифікації запитів. Кожен запит, що генерується системою, отримує унікальний ідентифікатор `completion_id` на основі стандарту UUID v4.

#### Лістинг 4.12. Stream generator

```
async def stream_generator(llm_service: LLMService, request: ChatCompletionRequest):
    completion_id = f"chatcpl-{uuid.uuid4()}"
```

Цей ідентифікатор передається у відповіді клієнту та фіксується у логах на всіх етапах обробки. Це дозволяє однозначно пов'язати помилку на клієнтській частині (UI) з конкретним записом у логах Router або Thinking Model, значно спрощуючи процес діагностики та підтримки системи, що відповідає вимогам, які були створені раніше.

## 4.5 Створення бенчмарку та аналіз результату

Для об'єктивної оцінки точності та стабільності роботи алгоритму семантичної маршрутизації (Router Model) розроблено автоматизований бенчмарк. Його метою є визначення оптимальних параметрів моделі (зокрема, температури генерації) та перевірка здатності класифікатора розрізняти інтент користувача в специфічному університетському контексті.

Бенчмарк реалізовано мовою Python з використанням бібліотек для асинхронних HTTP-запитів (aiohttp, openai), обробки даних (pandas, numpy) та візуалізації (matplotlib, seaborn).

Алгоритм тестування базується на наступних принципах:

- покриття параметричного простору: тестування проводиться для діапазону температур від 0.0 до 1.0 з кроком 0.1. Це дозволяє знайти баланс між детермінізмом (низька температура) та "креативністю" (висока температура), хоча для задач класифікації очікується, що нижчі температури покажуть кращий результат;
- навантажувальне тестування: використовується асинхронна архітектура (asyncio) з семафором (asyncio.Semaphore(50)), що дозволяє надсилати до 50 запитів одночасно, імітуючи реальне навантаження на систему;
- запобігання кешуванню: для гарантії чесності експерименту до кожного запиту додається унікальний випадковий рядок ("nonce"), що змушує LLM генерувати нову відповідь, а не повертати кешований результат.

Для тестування сформовано спеціалізований датасет із 60 запитів, розділених на два збалансовані класи (по 30 прикладів у кожному):

- клас GENERIC (Прості запити): включає питання про статичні факти ("Де знаходиться 1-й корпус?", "Як звати ректора?"), загальну інформацію ("Коли день студента?") та світську бесіду ("Привіт, ти бот?");
- клас THINKING\_TOOL\_USE (Комплексні запити): включає запити, що вимагають персоналізації ("Які у мене бали?"), доступу до зовнішніх систем

("Знайди в Moodle курс..."), логічного міркування ("Розрахуй, скільки балів треба...") або генерації контенту ("Напиши заяву...").

Такий розподіл дозволяє оцінити здатність моделі розрізняти семантично схожі, але функціонально різні запити.

Реалізація бенчмарку складається з трьох основних блоків: конфігурації, асинхронного виконання запитів та аналітичної візуалізації.

Блок 1. Асинхронний запит до моделі (get\_decision). Функція get\_decision виконує одиничний тестовий прогін. Вона формує запит до локального сервера (через OpenAI-сумісний клієнт), додає "сіть" для обходу кешу та нормалізує отриману відповідь.

#### Лістинг 4.13. Обхід кешу

```

async def get_decision(item, temp, iteration):
    # Додаємо "сіть" (nonce), щоб сервер не віддавав кешовану відповідь
    # Це критично для перевірки температури
    nonce = generate_nonce()
    query_with_salt = f"{item['input']}\n\n"

    try:
        response = await client.chat.completions.create(
            model=MODEL_NAME,
            messages=[
                {"role": "system", "content": SYSTEM_PROMPT},
                {"role": "user", "content": query_with_salt}
            ],
            temperature=temp,
            max_tokens=10
        )
        raw = response.choices[0].message.content.strip()
        actual = raw.replace(_old: ".", _new: "").upper()

        # Нормалізація
        if "THINKING" in actual:
            actual = "THINKING_TOOL_USE"
        elif "GENERIC" in actual:
            actual = "GENERIC"
        else:
            actual = "UNKNOWN"

        return {
            "query": item['input'],
            "expected": item['expected'],
            "actual": actual,
            "temp": temp,
            "match": item['expected'] == actual
        }

```

Блок 2. Оркестрація виконання (`run_benchmark`). Основна функція створює пул асинхронних завдань для всіх комбінацій "запит-температура-ітерація". Використання `tqdm` дозволяє візуалізувати прогрес виконання тисяч запитів у реальному часі. Отримані результати агрегуються у `pandas.DataFrame` для подальшого аналізу.

Блок 3. Тестові дані, що використовуватимуться для оцінки правильності перенаправлення запитів на відповідну модель.

#### Лістинг 4.14. Тестові дані

```

40 test_data = [
41     # --- GENERIC (30 шт) ---
42     {"input": "Де знаходиться 1-й корпус?", "expected": "GENERIC"},
43     {"input": "Як звати ректора?", "expected": "GENERIC"},
44     {"input": "Який номер телефону приймальної комісії?", "expected": "GENERIC"},
45     {"input": "Коли починається зимова сесія за графіком?", "expected": "GENERIC"},
46
47     # --- THINKING_TOOL_USE (30 шт) ---
48     {"input": "Які у мене бали з вищої математики?", "expected": "THINKING_TOOL_USE"},
49     {"input": "Знайди в Moodle курс 'Основи Python' і дай посилання.", "expected": "THINKING_TOOL_USE"},
50     {"input": "Чи перетинаються у мене пари в середу?", "expected": "THINKING_TOOL_USE"},
51 ]

```

Блок 3. Аналітика та візуалізація (`analyze_and_plot`). Цей модуль генерує комплексний звіт, що включає текстову статистику (середня точність, найкраща температура) та графічну панель із чотирьох діаграм:

- точність vs температура. Лінійний графік, що показує, як змінюється загальна точність класифікації при зміні температури. Це дозволяє візуально підтвердити гіпотезу про ефективність низьких температур;
- точність за категоріями. Дозволяє виявити дисбаланс у розпізнаванні класів (наприклад, чи модель схильна частіше помилково класифікувати складні запити як прості);
- аналіз стабільності. Стовпчикова діаграма, що показує кількість запитів, на які модель давала різні відповіді при одній і тій самій температурі (flip-flopping). Це критичний показник надійності, що характеризує придатність системи до оптимального використання ресурсів при обробці запитів;

- матриця помилок (confusion matrix). Теплова карта для найкращої температури, що показує точну кількість помилок типу False Positive та False Negative.

#### Лістинг 4.15. Матриця помилок

```
# D. Матриця плутанини (Confusion Matrix) для НАЙКРАЩОЇ температури
ax4 = fig.add_subplot(gs[1, 1])
best_temp_df = df[df['temp'] == best_temp]
cm = confusion_matrix(best_temp_df['expected'], best_temp_df['actual'], labels=["GENERIC", "THINKING_TOOL_USE"])

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax4,
            xticklabels=["GENERIC", "THINKING"],
            yticklabels=["GENERIC", "THINKING"])
ax4.set_title(f'Матриця помилок (при Temp = {best_temp})')
ax4.set_xlabel('Predicted')
ax4.set_ylabel('Actual')

plt.tight_layout()
plt.show()
```

Результати відпрацювання бенчмарку на основі даних з додатків, можна побачити на матриці помилок (рис. 4.4) та гістограмі точності по категоріях (рис. 4.5):

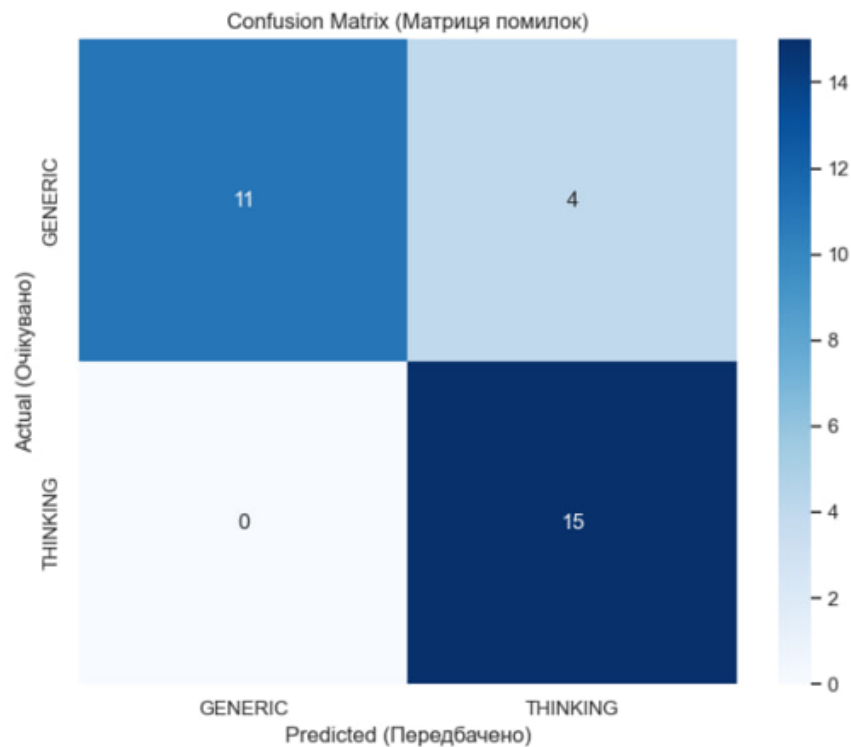


Рис. 4.4. Матриця помилок точності розподілення запитів по категоріях моделей

По осі X вказано категорії моделей, а по осі Y – кількість правильних та неправильних результатів.

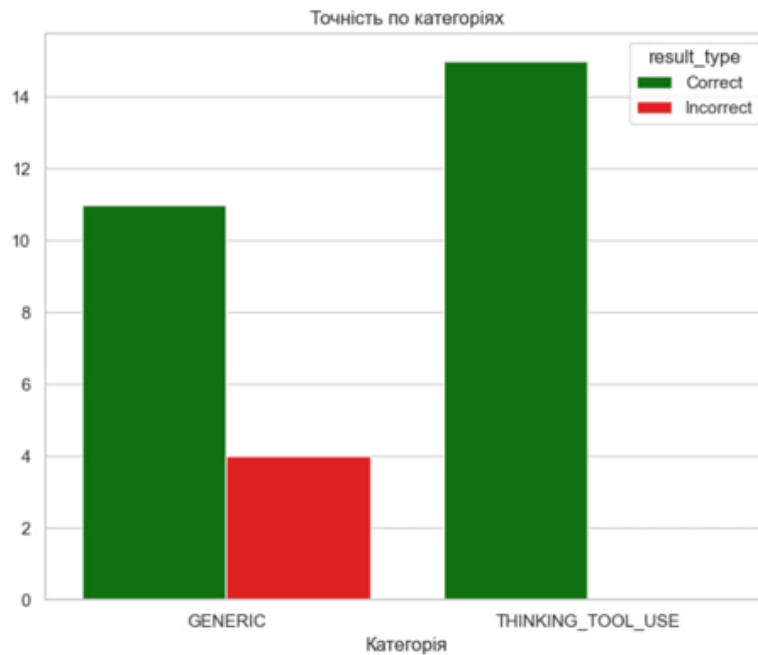


Рис. 4.5. Гістограма точності розподілення запитів по категоріях моделей

#### 4.6 Висновки до розділу

Здійснено практичну програмну реалізацію спроектованої архітектури системи Noodle AI. Головним результатом стала трансформація теоретичних моделей та алгоритмів у повнофункціональний програмний комплекс, побудований на принципах мікросервісної архітектури.

Реалізовано гетерогенну серверну інфраструктуру: для забезпечення гнучкості та масштабованості системи застосовано гібридний технологічний стек. Ядро інтелектуальної обробки (проксі-сервіси AI) реалізовано на Python 3.10 з використанням асинхронного фреймворку FastAPI. Для інтеграції з корпоративною системою розкладу використано Java 17 та фреймворк Spring Boot, що забезпечило надійну типізацію та стабільність роботи адаптера.

Програмно імплементовано алгоритм семантичної маршрутизації (Router Model). Реалізація базується на детермінованому виклику LLM (з параметром

temperature=0.0), що забезпечує стабільну класифікацію інтену користувача на категорії GENERIC та THINKING\_TOOL\_USE.

Створено спеціалізовані проксі-сервіси (Generic та Thinking Models):

- Generic Model Proxy реалізовано як сервіс для швидких відповідей;
- Thinking & Tool Use Model Proxy реалізовано як складний інтелектуальний агент на базі бібліотеки LangChain;
- розроблено та інтегровано адаптери зовнішніх даних (Tools) – Schedule, Moodle, WebSearch.

Впроваджено систему моніторингу та метрик: у всі мікросервіси інтегровано механізми централізованого логування та підрахунку токенів (ChatCompletionResponseUsage).

Проведено верифікацію та бенчмаркінг. Розроблено автоматизований скрипт для навантажувального тестування маршрутизатора. Результати експерименту підтвердили високу точність класифікації та стабільність роботи системи при конкурентних запитах. Емпірично доведено ефективність використання низьких температур для задач маршрутизації.

## ВИСНОВКИ

У магістерській роботі вивчено актуальне науково-прикладне завдання оптимізації організації навчального процесу у закладах вищої освіти шляхом розробки та імплементації інтелектуальної системи Noodle AI. Проведено комплексне дослідження, що охоплює теоретичний аналіз, архітектурне проектування, алгоритмічну розробку та програмну реалізацію.

Проведено аналіз сучасного стану освітніх технологій та тенденцій їх еволюції до парадигми Education 5.0. Встановлено, що якісно нові інструменти взаємодії зі студентами повинні забезпечувати не лише надання статичної довідкової інформації, а й глибоку гіпер-персоналізацію навчання та оперативний доступ до динамічних даних, таких як персональний розклад чи поточна успішність. Виявлено недоліки традиційних монолітних архітектур чат-ботів. Обґрунтовано доцільність переходу до мікросервісної агентної архітектури, що дозволяє мінімізувати ризики та забезпечити гнучкість системи.

Спроектовано та розроблено архітектуру системи, в основу якої покладено патерн "Модель-Проксі" (Proxy Model). Центральним елементом архітектури зроблено унікальний алгоритм семантичної маршрутизації, реалізований у сервісі Router Model Proxy. Досягнуто оптимального балансу ефективності та вартості. Створено набір інструментальних адаптерів. Зокрема, реалізовано Schedule Service на платформі Java з використанням фреймворку Spring Boot.

Практичну цінність роботи підтверджує здійснена програмна реалізація системи з використанням сучасного стеку технологій (Python, FastAPI, LangChain), яка завершилася проведенням експериментальним тестуванням.

Результати експерименту з використанням бенчмаркінгу підтвердили високу точність розробленого алгоритму маршрутизації та стабільність обробки конкурентних запитів під навантаженням. Системою моніторингу та логування забезпечено можливість детального аналізу ефективності використання токенів. Таким чином, розроблена система Noodle AI є завершеним програмним продуктом, впровадження якого підвищує ефективність навчального процесу.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "George A. S., Baskar T., Siranchuk N. The Evolution of Education 5.0 in the Innovation Era: A Review of the Progression from Teacher-Centered Learning to Student-Driven Models // Partners Universal International Innovation Journal (PUIIJ). 2025. Vol. 3, No. 1. P. 19–28.
2. UNESCO. Guidance for generative AI in education and research. Paris : UNESCO, 2023. 44 p.
3. Hwang G. J., Xie H., Wah B. W., Gašević D. Vision, challenges, roles and research issues of Artificial Intelligence in Education (AIED) // Computers and Education: Artificial Intelligence. 2020. Vol. 1. Art. 100001.
4. Chen L., Chen P., Lin Z. Artificial Intelligence in Education: A Review // IEEE Access. 2020. Vol. 8. P. 75264–75278.
5. Биков В. Ю., Спірін О. М., Пінчук О. П. Сучасні завдання цифрової трансформації освіти // Вісник Кафедри ЮНЕСКО «Неперервна професійна освіта ХХІ століття». 2020. № 1. С. 27–36.
6. Vaswani A. et al. Attention Is All You Need // Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008.
7. OpenAI. GPT-4 Technical Report. arXiv preprint arXiv:2303.08774. 2023.
8. Zhao W. X. et al. A Survey of Large Language Models.
9. Touvron H. et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv preprint arXiv:2307.09288. 2023.
10. Bommasani R. et al. On the Opportunities and Risks of Foundation Models.
11. Lewis P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks // Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 9459–9474.
12. Wei J. et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models // Advances in Neural Information Processing Systems. 2022. Vol. 35. P. 24824–24837.
13. Yao S. et al. ReAct: Synergizing Reasoning and Acting in Language Models.

14. Schick T. et al. Toolformer: Language Models Can Teach Themselves to Use Tools.
15. Chase H. LangChain: Building applications with LLMs. URL: <https://python.langchain.com> (дата звернення: 10.12.2025).
16. Liu Y. et al. LLM Agents: A Survey. arXiv preprint arXiv:2309.07864. 2023.
17. Gao Y. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv preprint arXiv:2312.10997. 2023.
18. Mialon G. et al. Augmented Language Models: a Survey. arXiv preprint arXiv:2302.07842. 2023.
19. Qin Y. et al. Tool Learning with Foundation Models. arXiv preprint arXiv:2304.08354. 2023.
20. Wang L. et al. A Survey on Large Language Model based Autonomous Agents. arXiv preprint arXiv:2308.11432. 2023.
21. Richardson C. Microservices Patterns: With examples in Java. Manning Publications, 2018. 520 p.
22. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. O'Reilly Media, 2021. 600 p.
23. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. PhD dissertation. University of California, Irvine, 2000.
24. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560 p.
25. Ramirez S. FastAPI: Modern Python Web Development. O'Reilly Media, 2024.
26. Percival H., Gregory B. Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices. O'Reilly Media, 2020.
27. Walls C. Spring in Action. 6th ed. Manning Publications, 2022. 520 p.
28. Dougiamas M., Taylor P. Moodle: Using Learning Communities to Create an Open Source Course Management System // Proceedings of the ED-MEDIA 2003 Conference. Honolulu, 2003. P. 171–178.
29. Moodle HQ. Moodle Web Service API Documentation. URL: [https://docs.moodle.org/dev/Web\\_services](https://docs.moodle.org/dev/Web_services) (дата звернення: 10.12.2025).

30. Anthropic. Model Context Protocol (MCP) Specification. URL: <https://modelcontextprotocol.io> (дата звернення: 10.12.2025).
31. Hickson I. Server-Sent Events. W3C Recommendation. URL: <https://html.spec.whatwg.org/multipage/server-sent-events.html> (дата звернення: 10.12.2025).
32. OpenAI. API Reference: Chat Completions. URL: <https://platform.openai.com/docs/api-reference/chat> (дата звернення: 10.12.2025).
33. Pandas Development Team. pandas: powerful Python data analysis toolkit. 2024. URL: <https://pandas.pydata.org>.
34. OWASP. OWASP Top 10 for Large Language Model Applications. Version 1.1. 2023. URL: <https://owasp.org/www-project-top-10-for-large-language-model-applications/>.
35. Liang P. et al. Holistic Evaluation of Language Models // Annals of the New York Academy of Sciences. 2023. Vol. 1525. P. 140–146.
36. Papineni K., Roukos S., Ward T., Zhu W. J. BLEU: a Method for Automatic Evaluation of Machine Translation // Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL). 2002. P. 311–318.
37. Глоба Л. С., Кот Т. М. Підхід до побудови інтелектуальних чат-ботів // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка. 2019. № 69. С. 58–63.
38. Висоцька В. А. Методи та засоби опрацювання інформаційних ресурсів в системах електронної комерції // Вісник Національного університету «Львівська політехніка». Інформаційні системи та мережі. 2018. № 901. С. 15–28.
39. Литвин В. В., Висоцька В. А., Пулейко І. В. Інтелектуальні системи підтримки прийняття рішень: навчальний посібник. Львів : Видавництво Львівської політехніки, 2019. 320 с.
40. Міністерство освіти і науки України. Стратегія розвитку вищої освіти в Україні на 2022-2032 роки. Київ, 2022.

## **ДОДАТКИ**

## Додаток А

### Алгоритм створення курсу на Moodle

Веб-сервіси / Зовнішні сервіси / Додати зовнішній сервіс

#### Moodle4Noodle

Основне Користувачі Курси Журнал оцінок Модулі Сторінки сайту **Сервер** Звіти Розробка

##### Зовнішній сервіс

Назва

Noodle Teacher Bot

Коротка назва

TeacherBot

Дозволено

Тільки авторизовані користувачі ?

Сховати додаткове...

Можна завантажувати файли ?

Може завантажувати файли ?

Потрібні права

? Нічого не вибрано

Знайти

Додати сервіс

Скасувати

! Обов'язково

Рис. А.1. Введення назви курсу

Веб-сервіси / Зовнішні сервіси / Функції / Додати функції

#### Moodle4Noodle

Основне Користувачі Курси Журнал оцінок Модулі Сторінки сайту **Сервер** Звіти Розробка

#### Noodle Teacher Bot

##### Додати функції

Назва

! x core\_course\_create\_courses:Create new courses

Знайти

Додати функції

Скасувати

! Обов'язково

Рис. А.2. Додавання функцій

Облікові записи / Додати нового користувача

## Moodle4Noodle

Знайти



Основне **Користувачі** Курси Журнал оцінок Модулі Більше ▾

Розгорнути всі

▾ Основне

Ім'я входу ?  !  
Дозволено тільки символи нижнього регістру

Оберіть спосіб ідентифікації: ?  ▾

Заблокований обліковий запис ?

Згенерувати пароль та повідомити користувача

Ваш пароль повинен мати принаймні 8 символів, принаймні 1 цифр(а), принаймні 1 букв(а) нижнього регістру, принаймні 1 букв(а) верхнього регістру, принаймні 1 спеціальних символів, таких як \*, - або #

Новий пароль ?  ! ?

Пароль має містити принаймні 1 спеціальних символів, таких як \*, - або #.

Примусити змінити пароль ?

Прізвище !

Ім'я !

Електронна пошта !

Вилученість ?  ▾

Рис. А.3. Створення облікового запису

Веб-сервіси / Авторизовані користувачі

## Moodle4Noodle

Основне **Користувачі** Курси Журнал оцінок Модулі Сторінки сайту **Сервер** Звіти Розробка

## Виберіть авторизованих користувачів

Авторизовані користувачі

Авторизовані користувачі (1)

Bot Teacher (teacher@moodle.paws.dedyn.io)

Додати

Видалити

Рис. А.4. Вибір користувачів

Параметри пошуку >

## Змінити налаштування для авторизованих користувачів

- Bot Teacher  
teacher@moodle.paws.dedyn.io  
Відсутні права: 2 ▼
  - ▷ Створювати курси  
moodle/course:create
  - ▷ Ховати/Показувати курси  
moodle/course:visibility

Рис. А.5. Налаштування користувачів

Права / Призначити системні ролі

## Moodle4Noodle

Основне Користувачі Курси Журнал оцінок Модулі Сторінки сайту Сервер Звіти Розробка

Назад

## Призначення ролі 'Менеджер' в Ядро системи

Попередження: будь-які ролі, які ви призначите на цій сторінці, застосовуватимуться до користувачів усієї системи, включаючи до

Існуючі користувачі

Існуючі користувачі (1)  
Bot Teacher (teacher@moodle.paws.dedyn.io)

Рис. А.6. Призначення ролі

Знайти

Очистити

Параметри пошуку >

## Змінити налаштування для авторизованих користувачів

- Bot Teacher  
teacher@moodle.paws.dedyn.io

Рис. А.7. Змінити налаштування для авторизованих користувачів

## Moodle4Noodle

Основне Користувачі Курси Журнал оцінок Модулі Сторінки сайту **Сервер** Звіти Розробка

### Створення маркера

▼ Маркер

Ім'я

Якщо ви не введете назву, буде використовуватися випадкова назва.

Користувач ! × Bot Teacher teacher@moodle.paws.dedyn.io

Сервіс ! Noodle.ai Teacher Service ▼

IP обмеження

Діє до  Включити

Рис. А.8. Створення маркера

## Moodle4Noodle

На головну Налаштування Учасники Звіти Банки питань Більше ▼

### Доступні курси

Автоматично створений курс Python

Опис створено через API...

Ефективний тайм-менеджмент: як встигати все



Цей курс навчить вас пріоритизувати завдання, боротися з прокрастинацією та використовувати сучасні інструменти планування

**Викладач:** Користувач Адміністратор

Рис. А.9. Доступні курси

## Додаток Б

### Тестові дані для розподілення між Generic та Thinking Model

#### Лістинг Б.1

```

39 # --- 60 ТЕСТОВИХ ЗАПИТІВ (УНІВЕРСИТЕТ) ---
40 test_data = [
41     # --- GENERIC (30 шт) ---
42     {"input": "Де знаходиться 1-й корпус?", "expected": "GENERIC"},
43     {"input": "Як звати ректора?", "expected": "GENERIC"},
44     {"input": "Який номер телефону приймальної комісії?", "expected": "GENERIC"},
45     {"input": "Коли починається зимова сесія за графіком?", "expected": "GENERIC"},
46     {"input": "Скільки коштує проживання в гуртожитку?", "expected": "GENERIC"},
47     {"input": "Яка адреса головного сайту університету?", "expected": "GENERIC"},
48     {"input": "Привіт, ти бот?", "expected": "GENERIC"},
49     {"input": "Чи працює бібліотека в неділю?", "expected": "GENERIC"},
50     {"input": "Як розшифровується ЕКТС?", "expected": "GENERIC"},
51     {"input": "Де можна поїсти в університеті?", "expected": "GENERIC"},
52     {"input": "Скільки триває велика перерва?", "expected": "GENERIC"},
53     {"input": "Який прохідний бал був на право минулого року?", "expected": "GENERIC"},
54     {"input": "Назви перелік факультетів.", "expected": "GENERIC"},
55     {"input": "Хто завідувач кафедри маркетингу?", "expected": "GENERIC"},
56     {"input": "Де знаходиться медпункт?", "expected": "GENERIC"},
57     {"input": "Які документи потрібні для вступу (загальний список)?", "expected": "GENERIC"},
58     {"input": "Коли день студента?", "expected": "GENERIC"},
59     {"input": "Чи є в університеті басейн?", "expected": "GENERIC"},
60     {"input": "Як доїхати до спорткомплексу?", "expected": "GENERIC"},
61     {"input": "Яка стипендія у відмінників (сума)?", "expected": "GENERIC"},
62     {"input": "Добрий день!", "expected": "GENERIC"},
63     {"input": "Що таке силабус?", "expected": "GENERIC"},
64     {"input": "Коли засновано університет?", "expected": "GENERIC"},
65     {"input": "Де знайти розклад дзвінків?", "expected": "GENERIC"},
66     {"input": "Чи є Wi-Fi в холі?", "expected": "GENERIC"},
67     {"input": "Дякую за допомогу.", "expected": "GENERIC"},
68     {"input": "Як звати голову студради?", "expected": "GENERIC"},
69     {"input": "Який код ЄДРПОУ університету?", "expected": "GENERIC"},
70     {"input": "Скільки поверхів у 5-му корпусі?", "expected": "GENERIC"},
71     {"input": "Де курилка?", "expected": "GENERIC"},
72 ]

```

#### Лістинг Б. 2

```

# --- THINKING_TOOL_USE (30 шт) ---
{"input": "Які у мене бали з вищої математики?", "expected": "THINKING_TOOL_USE"},
{"input": "Знайди в Moodle курс 'Основи Python' і дай посилання.", "expected": "THINKING_TOOL_USE"},
{"input": "Чи перетинаються у мене пари в середу?", "expected": "THINKING_TOOL_USE"},
{"input": "Напиши заяву на матеріальну допомогу.", "expected": "THINKING_TOOL_USE"},
{"input": "Розрахуй, скільки балів мені треба на іспиті для 'п'ятірки'.", "expected": "THINKING_TOOL_USE"},
{"input": "Перевір, чи вільна аудиторія 305 зараз.", "expected": "THINKING_TOOL_USE"},
{"input": "Знайди email викладача, який веде 'Філософію'.", "expected": "THINKING_TOOL_USE"},
{"input": "Що мені потрібно знати, щоб закрити хвост?", "expected": "THINKING_TOOL_USE"},
{"input": "Підсумуй останні новини з сайту про гранти.", "expected": "THINKING_TOOL_USE"},
{"input": "Склади графік підготовки до диплому на місяць.", "expected": "THINKING_TOOL_USE"},
{"input": "Чому не працює мій логін в кабінеті?", "expected": "THINKING_TOOL_USE"},
{"input": "Порівняй вимоги до курсової на 3 і 4 курсі.", "expected": "THINKING_TOOL_USE"},
{"input": "Знайди в бібліотеці підручники з Java 2023 року.", "expected": "THINKING_TOOL_USE"},
{"input": "Якщо я пропущу 30% пар, чи допустять мене до сесії?", "expected": "THINKING_TOOL_USE"},
{"input": "Напиши код сортування масиву для лаби.", "expected": "THINKING_TOOL_USE"},
{"input": "Переклади анотацію до статті на англійську.", "expected": "THINKING_TOOL_USE"},
{"input": "Як оформити академічну відпустку? Покроково.", "expected": "THINKING_TOOL_USE"},
{"input": "Знайди вільний час у розкладі декана.", "expected": "THINKING_TOOL_USE"},
{"input": "Проаналізуй мій середній бал динаміці за 2 роки.", "expected": "THINKING_TOOL_USE"},
{"input": "Створи слайди для презентації проекту.", "expected": "THINKING_TOOL_USE"},
{"input": "Виріши цю задачу з теорії ймовірностей.", "expected": "THINKING_TOOL_USE"},
{"input": "Перевір цей текст на граматичні помилки.", "expected": "THINKING_TOOL_USE"},
{"input": "Придумай тему для стартапу в сфері EdTech.", "expected": "THINKING_TOOL_USE"},
{"input": "Як налаштувати VPN університету на iPhone?", "expected": "THINKING_TOOL_USE"},
{"input": "Скільки кредитів мені залишилось до бакалавра?", "expected": "THINKING_TOOL_USE"},
{"input": "Напиши ввічливий лист викладачу про хворобу.", "expected": "THINKING_TOOL_USE"},
{"input": "Поясни складну тему з лекції простими словами.", "expected": "THINKING_TOOL_USE"},
{"input": "Побудуй маршрут між корпусами 1 та 4.", "expected": "THINKING_TOOL_USE"},
{"input": "Які документи з бази мені треба оновити?", "expected": "THINKING_TOOL_USE"},
{"input": "Створи SQL запит для вибірки студентів-боржників.", "expected": "THINKING_TOOL_USE"},

```

## Додаток В

### Приклади запитів до чат-боту

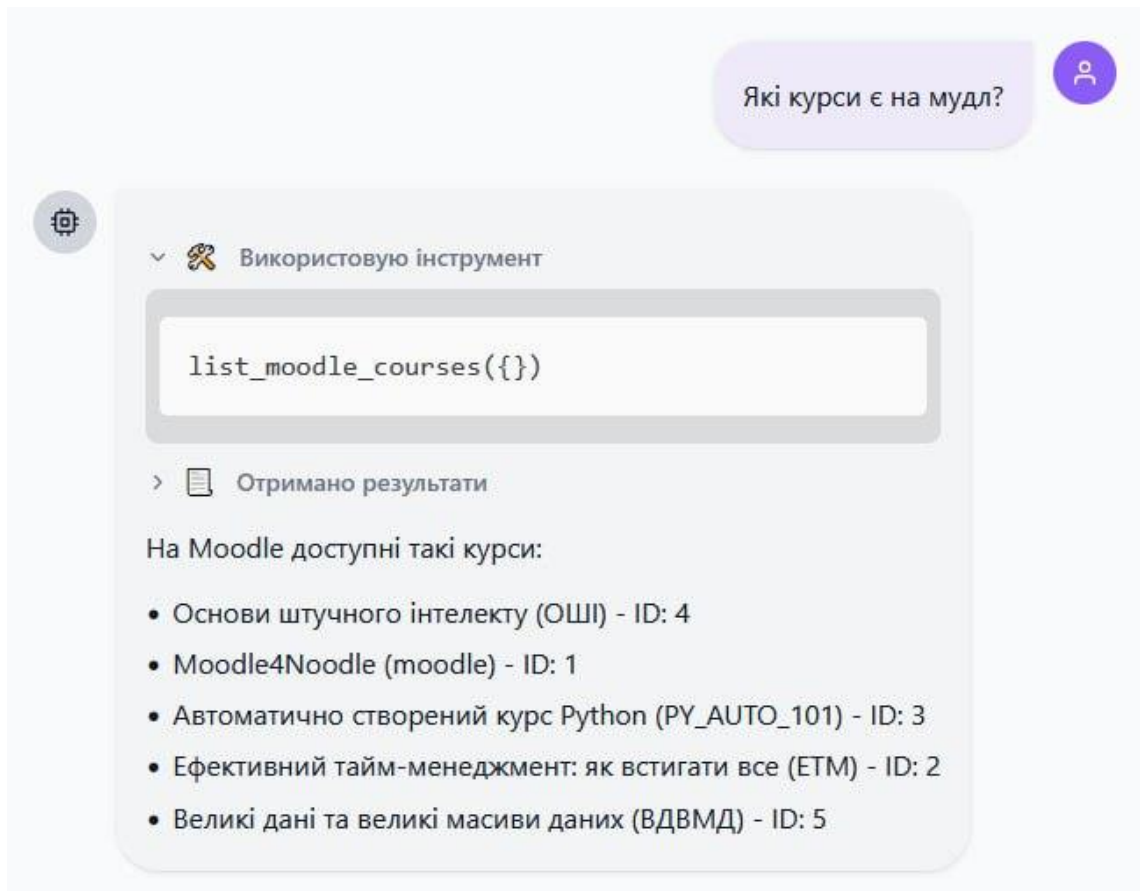


Рис. В.1. Запит щодо курсів а мудлі

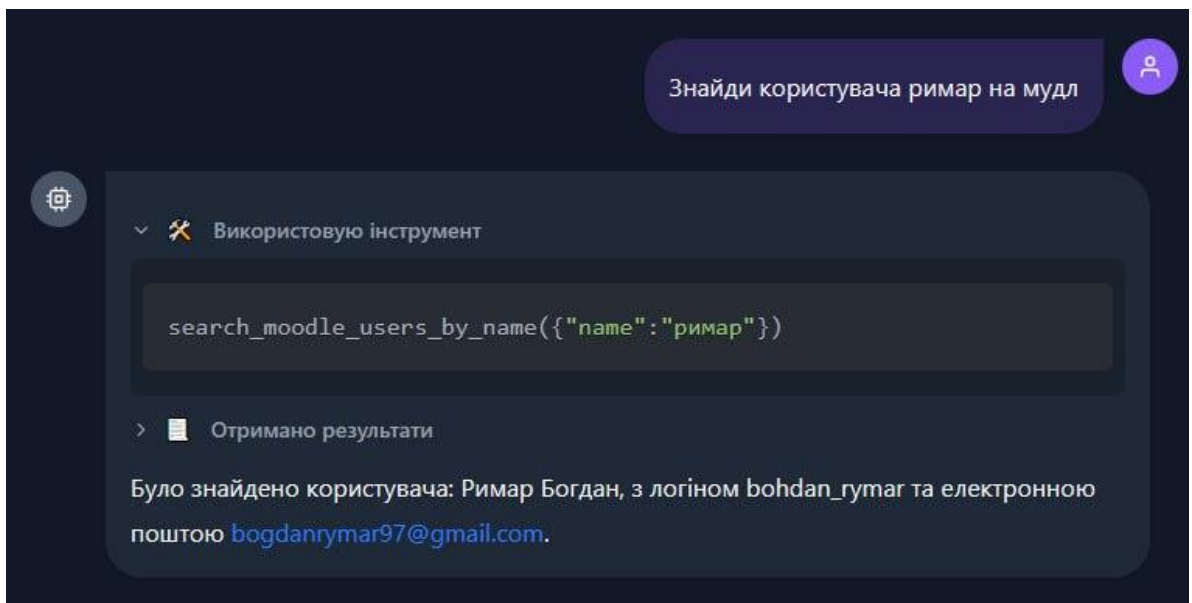


Рис. В.2. Запит щодо пошуку користувача на Moodle

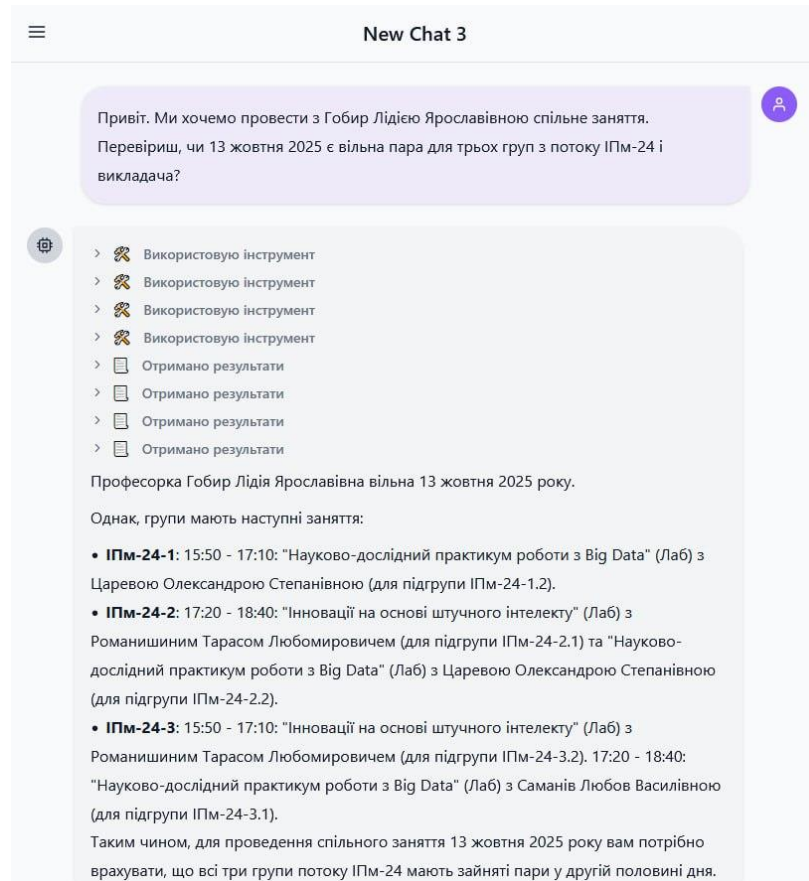


Рис. В.3. Запит щодо розкладу груп та викладачів

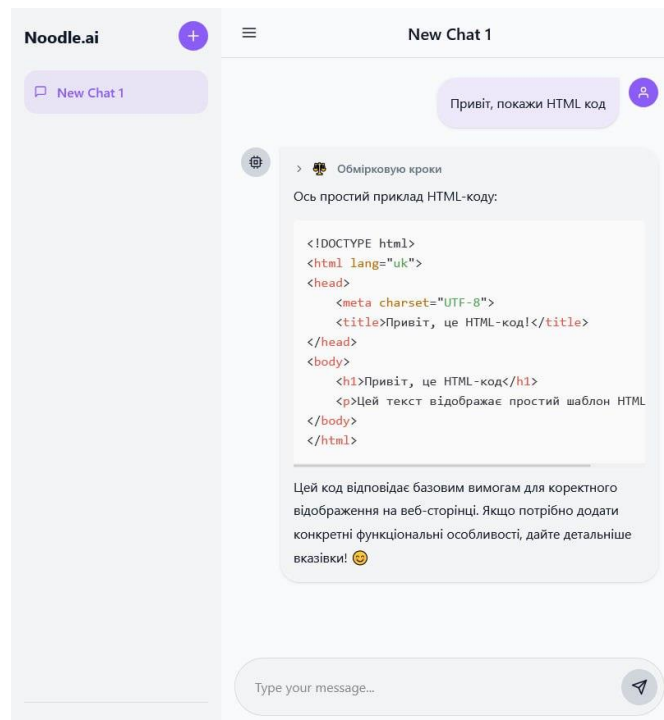


Рис. В.4. Прохання показати зразок коду