ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавр

на тему: **«Автоматизація підбору рейсів автобусного та залізничного транспорту на основі інтелектуального чат-бота з використанням ChatGPT»**

Виконала: здобувачка вищої освіти
за освітньою програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні
системи та технології
ступеня вищої освіти бакалавр
групи 126ІСТ_бд_31[1]
Горчакова Марина Олександрівна
Керівник: Слюсарь Ігор Іванович
Рецензент: Муравльов Володимир
В’ячеславович

Полтава – 2026 року

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

Освітня програма Інформаційні управляючі системи
Спеціальність 126 Інформаційні системи та технології
Рівень вищої освіти перший (бакалаврський)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Юрій УТКІН
«10» червня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧКИ ВИЩОЇ ОСВІТИ

Горчакової Марини Олександрівни

1. Тема кваліфікаційної роботи:
«Автоматизація підбору рейсів автобусного та залізничного транспорту на основі інтелектуального чат-бота з використанням ChatGPT»,
Керівник роботи: к. т. н., доцент, доцент кафедри інформаційних систем та технологій Слюсарь Ігор Іванович.
Затверджено наказом закладу вищої освіти від «10» квітня 2026 року № 384-ст

2. Строк подання здобувачем вищої освіти роботи «08» червня 2026 року.

3. Вихідні дані до роботи: науково-технічна література, аналітичні джерела мережі Інтернет, що містять інформацію про агрегатори транспортних рейсів, інтелектуальні чат-боти, AI-агентів, пошукового агента Tavily API, техніки RAG, LLM від к. OpenAI, ChatGPT, GPT-4o, Busfor, Infobus

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Розділ 1. Аналіз сучасних підходів до автоматизації підбору транспортних рейсів
Розділ 2. Розробка моделі інтелектуальної системи для пошуку транспортних рейсів
Розділ 3. Рекомендації щодо практичної реалізації автоматизації підбору транспортних рейсів на основі штучного інтелекту

5. Перелік графічного матеріалу: схеми, рисунки, діаграми за темою та об’єктом дослідження

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання отримав
Оцінювання економічної ефективності результатів дослідження	Загребельна І. Л., к. е. н., доцент, доцент кафедри економіки та публічного управління	01.04.2026 р.	29.05.2026 р.

7. Дата видачі завдання «10» червня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Вибір і затвердження теми роботи	02.06.2025 р. – 04.06.2025	
2	Складання і затвердження розгорнутого плану та завдання на кваліфікаційну роботу	05.06.2025 р. – 10.06.2025 р.	
3	Опрацювання джерел інформації	11.06.2025 р. – 15.06.2025 р.	
4	Збір, вивчення і обробка інформації, необхідної для виконання роботи	16.06.2025 р. – 20.06.2025 р.	
5	Виконання теоретико-методологічного розділу роботи	08.09.2025 р. – 01.11.2025 р.	
6	Виконання дослідницько-аналітичного розділу роботи	19.01.2026 р. – 14.02.2026 р.	
7	Виконання проєктно-рекомендаційного розділу роботи	16.02.2026 р. – 31.03.2026 р.	
8	Оцінювання економічної ефективності результатів дослідження	01.04.2026 р. – 29.05.2026 р.	
9	Оформлення тексту роботи	01.06.2026 р. – 06.06.2026р.	
10	Попередній захист роботи на кафедрі	08.06.2026 р.	
11	Доопрацювання роботи з урахуванням зауважень і пропозицій	09.06.2026 р. – 10.06.2026 р.	
12	Нормоконтроль	11.06.2026 р. – 15.06.2026 р.	
13	Захист кваліфікаційної роботи	16.06.2026 р. - 18.06.2026 р.	

Здобувач вищої освіти **Марина ГОРЧАКОВА**

Керівник роботи **Ігор СЛЮСАРЬ**

ЗМІСТ

ВСТУП

Актуальність теми кваліфікаційної роботи підтверджується необхідністю підвищення зручності, швидкості та якості пошуку транспортних рейсів для користувачів. Пасажири часто користуються різними онлайн-сервісами для пошуку автобусних і залізничних маршрутів, проте такі сервіси не завжди забезпечують достатньо гнучку інформаційну підтримку. У сучасних умовах для автоматизації підбору транспортних рейсів доцільне застосування штучного інтелекту (AI), зокрема інтелектуальних чат-ботів, які здатні взаємодіяти з користувачем у природній мовній формі. Однак питання використання ChatGPT спільно з технологією пошуково-підкріпленої генерації потребує додаткових досліджень. Все це свідчить про актуальність теми роботи.

Метою кваліфікаційної роботи є підвищення ефективності інформаційного обслуговування користувачів у підборі рейсів автобусного та залізничного транспорту за рахунок створення інтелектуального чат-бота.

Завданнями кваліфікаційної роботи є:

– обґрунтувати вибору інструментарію для інформаційної підтримки користувачів для пошуку транспортних рейсів;

– розробити модель інтелектуальної системи для пошуку транспортних рейсів;

– сформувати рекомендацій щодо практичної реалізації інтелектуальної системи для пошуку транспортних рейсів;

– виконати економічне обґрунтування прийнятих рішень.

Об’єктом дослідження є процес інформаційної підтримки користувачів під час пошуку, порівняння та вибору рейсів автобусного та залізничного транспорту.

Предметом дослідження є методи, моделі та програмні засоби автоматизації підбору рейсів автобусного та залізничного транспорту на основі технологій штучного інтелекту.

Методами дослідження є аналітичний метод – підходів до автоматизації підбору транспортних рейсів, економічного обґрунтування прийнятих рішень; порівняльний аналіз – для зіставлення наявних рішень, технологій AI та можливостей використання ChatGPT для інформаційної підтримки користувачів під час пошуку рейсів; метод системного аналізу – для визначення функціональних і нефункціональних вимог до інтелектуальної системи, опису її основних компонентів і взаємозв'язків між ними; моделювання – для побудови інтелектуальної системи для пошуку транспортних рейсів та перевірки її функціональних можливостей

Інформаційна база кваліфікаційної роботи сформована з Інтернет-ресурсів, що місять інформацію про онлайн-сервіси та агрегатори для пошуку транспортних рейсів, технологію пошуково-підкріпленої генерації, інтелектуальні чат-боти.

Практична значущість роботи полягає у розробці моделі локального транскрибування аудіо в текст, рекомендацій щодо практичної реалізації автоматизації підбору транспортних рейсів на основі AI. Результати можуть бути використані для подальших досліджень за даною тематикою та при проектуванні AI-агентів.

Апробація результатів відбувалася в рамках XXII щорічного міждисциплінарного семінару «Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій» (листопад 2025 р., м. Полтава), XXI щорічної студентської наукової конференції «Сучасні інформаційні технології та інноваційні методики в економіці, менеджменті та бізнесі» Полтавського державного аграрного університету (22 квітня 2026 р., м. Полтава).

За результатами досліджень здійснено публікацію двох тез доповідей.

Структура кваліфікаційної роботи логічно пов'язана з завданнями досліджень і містить вступ, три розділи основної частини, висновки, список використаних джерел, додатки. Загальний обсяг пояснювальної записки кваліфікаційної роботи складає 69 сторінок формату А4. Вона містить 23 рисунка і 11 таблиць.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО АВТОМАТИЗАЦІЇ ПІДБОРУ ТРАНСПОРТНИХ РЕЙС

1.1 Аналіз існуючих онлайн-сервісів для пошуку транспортних рейсів

Якість, повнота та оперативність надання транспортної інформації безпосередньо впливає на мобільність населення, ефективність логістичних процесів та загальний рівень задоволеності якістю послуг. Саме тому об'єктом дослідження у межах даної кваліфікаційної роботи обрано інформаційні процеси пошуку, агрегації та інтелектуальної обробки даних про пасажирські перевезення в межах транспортної інфраструктури України.

З 2022 р. транспортна система країни зазнала суттєвої структурної трансформації. Закриття повітряного простору та призупинення міжнародного авіасполучення фактично перерозподілило весь пасажиропотік між залізничним та автомобільним видами транспорту. За даними АТ «Укрзалізниця», у 2022-2024 роках обсяги перевезень пасажирів поїздами зросли на понад 40 % порівняно з попереднім рівнем [1]. Це суттєво навантажило наявну інфраструктуру та поставило нові вимоги до систем інформування пасажирів, адже необхідність оперативного планування переміщення між містами стала критично важливою для мільйонів громадян.

Сучасний ринок пасажирських перевезень в Україні характеризується двома ключовими особливостями: монопольним становищем державного оператора у залізничному сегменті та надзвичайно фрагментованою структурою автобусного ринку. Залізничний сегмент представлений єдиним суб'єктом – АТ «Укрзалізниця», яке здійснює перевезення на понад 600 маршрутах внутрішнього та міжнародного сполучення, використовуючи розгалужену мережу залізниць загальною протяжністю понад 20 тисяч кілометрів. Підприємство розвиває власну цифрову екосистему: вебсайт, мобільний застосунок та інтеграції з агрегаторами через API, що забезпечує

відносно уніфікований доступ до актуальних даних про рух поїздів та наявність квитків. Принципово іншою є ситуація в сегменті автобусних та мікроавтобусних перевезень. Він представлений тисячами приватних автотранспортних підприємств (АТП) та фізичних осіб-підприємців, чий технологічний рівень суттєво варіюється. Великі регіональні компанії, зокрема «Зелений Слон 7», «Ковельське АТП», «Пасажирські перевезення» та інші, мають власні вебресурси та системи онлайн-бронювання. Водночас значна частина дрібних перевізників, що здійснюють рейси на регіональних маршрутах, або взагалі не має цифрової присутності, або розміщує інформацію у неструктурованому вигляді на застарілих вебсторінках без будь-якої стандартизації формату даних. Для глибокого розуміння проблематики необхідно проаналізувати типовий сценарій взаємодії сучасного пасажира з наявною інформаційною інфраструктурою. Припустимо, що мешканець Полтави планує поїздку до Ужгорода. Процес пошуку оптимального маршруту у поточних реаліях передбачає такі кроки.

1. Звернення до офіційного вебпорталу або застосунку «Укрзалізниця» для перевірки прямого залізничного сполучення або варіантів з пересадками.

2. Перевірка агрегаторів автобусних перевезень (Busfor [2], INFOBUS [3]) для порівняння вартості та часу в дорозі з автобусним варіантом.

3. Самостійний пошук через Google сайтів локальних перевізників, що здійснюють рейси за напрямком «Полтава – Ужгород».

4. Ручне зіставлення розкладів, цін та умов із різних джерел для прийняття обґрунтованого рішення.

Такий підхід є надзвичайно часозатратним, а отримана інформація нерідко виявляється суперечливою або застарілою. З погляду системного аналізу, цей процес характеризується відсутністю єдиного інформаційного простору та відсутністю стандартизованих механізмів обміну даними між суб'єктами ринку.

Аналіз технологічного стану інформаційної інфраструктури транспортного ринку дозволяє виокремити такі системні проблеми:

інформаційна фрагментація – відсутність єдиної платформи для порівняння різних видів транспорту в межах одного запиту; технологічна неоднорідність – різні перевізники використовують принципово різні формати представлення даних (від сучасних REST API до вручну оновлюваних HTML-таблиць); проблема актуальності – значна частина агрегаторів зберігає кешовані копії розкладів, які не відображають оперативних змін у графіках руху; відсутність семантичної обробки – існуючі системи орієнтовані на параметричний пошук за чіткими критеріями, але не здатні коректно обробляти запити природньою мовою; обмеженість деталізації – більшість агрегаторів надають лише базові атрибути рейсу (час, ціна), не розкриваючи деталей щодо зручностей на борту (Wi-Fi, розетки, тип сидінь), що важливо для прийняття рішення. Соціально-економічна значущість вирішення окреслених проблем підтверджується широким колом потенційних бенефіціарів системи. За статистичними оцінками, щодня в Україні здійснюється кілька мільйонів поїздок залізничним та автомобільним транспортом. Кожен пасажир, що витрачає додатковий час на пошук та порівняння варіантів маршруту, зазнає прихованих витрат, які у сукупності формують значний соціально-економічний збиток. Автоматизація цього процесу за допомогою інтелектуальної системи дозволить не лише заощадити час пасажирів, але й підвищити загальну ефективність транспортної системи через більш рівномірного розподілу пасажиропотоків між доступними рейсами [4].

Об'єкт дослідження є складною соціотехнічною системою, де центральна проблема полягає у переході від моделі «статична база даних – пошуковий запит» до принципово нової парадигми «інтелектуальний агент – динамічний верифікований пошук». Це дозволить мінімізувати інформаційний розрив між перевізниками та пасажирами і забезпечити надання консолідованих, актуальних та вичерпної інформації в режимі онлайн за допомогою інтуїтивно зрозумілого інтерфейсу. Ефективне проектування нової інформаційної системи неможливе без ретельного аналізу наявних рішень на ринку. Оглядово-порівняльний аналіз є невід'ємним методологічним елементом дослідження,

оскільки дозволяє: по-перше, чітко ідентифікувати незайняті ніші та функціональні прогалини; по-друге, запозичити вдалі архітектурні рішення; по-третє, уникнути помилок, допущених конкурентами. На сьогодні ринок транспортних інформаційних систем України охоплює чотири основні групи: офіційні платформи транспортних компаній, мультимодальні агрегаційні сервіси, корпоративні інструменти для бронювання та рішення на базі ШІ.

Офіційний вебпортал та мобільний застосунок АТ «Укрзалізниця» є еталонним прикладом першої категорії. Ключовою перевагою цієї системи є прямий інтеграційний зв'язок із системою управління пасажирськими перевезеннями (GDS – Global Distribution System) підприємства, що гарантує максимальну точність і актуальність даних щодо наявності місць та поточного розкладу руху поїздів. Функціональні можливості застосунку включають: повнофункціональний пошук маршрутів за параметрами (дата, тип вагону, кількість пасажирів), онлайн-бронювання та оплату квитків, відстеження поточного розташування потягу на карті режимі онлайн, а також сповіщення про зміни в розкладі.

Проте системний аналіз виявив ряд суттєвих обмежень офіційного сервісу УЗ. Найбільш критичним є його виключна спрямованість на залізничний вид транспорту. Користувач, якому необхідно порівняти поїздку поїздом із автобусним варіантом за ціною або часом у дорозі, змушений самостійно звертатися до інших ресурсів. Крім того, інтерфейс застосунку, попри певне вдосконалення останніми роками, все ще не є оптимально адаптованим для швидкого прийняття рішень у стресових ситуаціях (наприклад, при несподіваній зміні планів). Система не підтримує взаємодію природньою мовою та потребує від користувача точного введення параметрів у визначені поля форми. Другу категорію складають комерційні мультимодальні агрегатори. Провідними представниками цієї групи на українському ринку є Busfor та INFOBUS. Ці платформи накопичують дані від сотень приватних автоперевізників та надають інструменти для їхнього порівняння в єдиному інтерфейсі. Базова бізнес-модель цих сервісів ґрунтується на комісії з продажів:

агрегатор отримує відсоток від кожного броні через платформу квитка, що мотивує перевізників інтегруватися в систему. Такий підхід дозволив цим платформам акумулювати широку базу операторів та досягти відносно повного охоплення ринку автобусних перевезень. Детальне дослідження функціонування Busfor та INFOBUS виявило такі технологічні недоліки, що мають принципове значення для обґрунтування розробки нового рішення:

– кешування даних – актуалізація розкладів на платформах агрегаторів відбувається не в режимі реального часу, а з певною затримкою, що призводить до розбіжностей між відображуваною та реальною ситуацією, особливо при оперативних змінах перевізником графіка або тарифної сітки;

– обмежена деталізація – карткові описи рейсів на агрегаторах, як правило, містять лише базові атрибути: час відправлення та прибуття, загальну вартість та назву перевізника. Важлива для пасажира деталізована інформація про конкретну модель транспортного засобу, наявність та стан кондиціонера, Wi-Fi, розеток та туалету на борту часто відсутня або вимагає переходу на сторонні ресурси;

– рекламне навантаження інтерфейсу – з комерційних міркувань інтерфейс агрегаторів перенасичений рекламними блоками, банерами та агресивними закликами до дії, що суттєво знижує зручність користування, особливо на мобільних пристроях з обмеженим екраном;

– відсутність мультимодальності – незважаючи на назву «агрегатор», Busfor та INFOBUS орієнтовані виключно на автобусний ринок та не надають можливості порівняти автобусний та залізничний варіанти.

Третю категорію формують корпоративні онлайн-системи бронювання окремих приватних перевізників. Їх характерним представником є сервіс компанії «Зелений Слон 7», що здійснює регулярні рейси між Полтавою, Києвом та Харковом [5]. Сайт компанії надає найбільш деталізовану інформацію про конкретні моделі автобусів (зокрема, BOVA VDL та аналогічні), кількість доступних місць із можливістю візуального вибору, фотографії салону та детальний опис умов поїздки. Такий рівень деталізації є

суттєвою перевагою для вимогливого пасажира. Проте ці системи залишаються принципово ізольованими: вони обслуговують виключно маршрути конкретного перевізника та не мають жодної механіки інтеграції із загальними пошуковими запитами пасажирів [6].

Четверта категорія – інтелектуальні асистенти та боти на основі штучного інтелекту – на сьогодні перебуває на початковому етапі розвитку у транспортному сегменті України. Наявні рішення здебільшого є простими чат-ботами з жорсткою логікою сценаріїв (decision-tree botsbot), які не здатні обробляти довільні запити природньою мовою та, зазвичай позбавлені можливості отримувати актуальні дані у реальному часі. Саме цей технологічний розрив визначає нішу для розробленого у межах даної роботи рішення. Для забезпечення системної візуалізації, узагальнення та структурування результатів проведеного аналізу, а також з метою виявлення найбільш оптимальних технологічних рішень, розглянемо ключові характеристики цих систем (табл. 1.1).

Таблиця 1.1 – Порівняльний аналіз функціональності транспортних інформаційних систем

Критерій порівняння	Сервіс УЗ	Busfor / INFOBUS	Системи перевізників	Проєктне рішення (бот)
Охоплення видів транспорту	Залізниця	Лише автобуси	Один перевізник	Залізниця + автобуси
Актуальність даних	Реальний час (API)	Кешування (затримка)	Реальний час	Реальний час (RAG)
Метод обробки запиту	Форма-пошук	Форма-пошук	Форма-пошук	Природна мова (NLP)
Деталізація рейсу	Середня	Низька	Висока	Висока (ШІ-аналіз)
Наявність ШІ	Відсутня	Відсутня	Відсутня	GPT-4o + RAG
Мобільність інтерфейсу	Застосунок	Сайт / застосунок	Сайт	Telegram (кросплатформ)
Персоналізація відповіді	Відсутня	Відсутня	Відсутня	Висока

Як свідчать дані, жодна з існуючих систем, представлених на ринку, не здатна забезпечити одночасне виконання трьох ключових критеріїв: повної

мультимодальності з охопленням різних видів транспорту, високої актуальності своєчасності інформації в режимі реального часу та підтримки обробки користувацьких запитів природною мовою [7]. Саме ця тріада характеристик є визначальною для задоволення динамічних потреб сучасного пасажира в умовах цифровізації інфраструктури. Синтез зазначених функцій у межах єдиної інформаційної платформи становить головну інноваційну цінність та науково-практичну новизну розробленого архітектурного рішення, що дозволяє усунути недоліки класичних аналогів. Для наочної ілюстрації взаємодії систем з різними категоріями перевізників розглянемо схему (рис. 1.1), яка відображає поточний стан інформаційної екосистеми транспортного ринку.

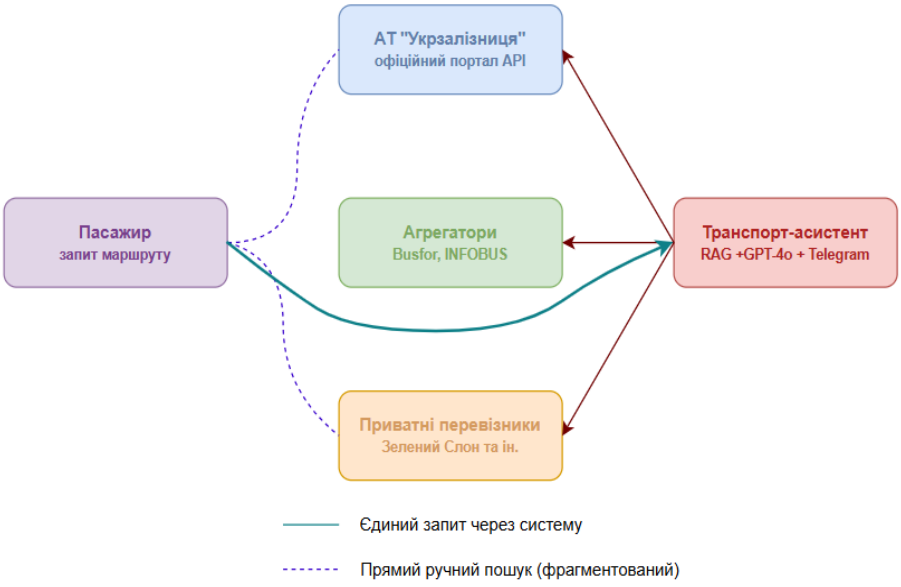

Рисунок 1.1 – Екосистеми ринку пасажирських перевезень України

Виявлені прогалини у функціональності існуючих рішень є безпосереднім обґрунтуванням актуальності та доцільності розробки нової інтелектуальної системи для пошуку транспортних рейсів на основі архітектури RAG та великих мовних моделей, що детально описується у наступних підрозділах роботи.

1.2 Обґрунтування вибору технологій штучного інтелекту для інформаційної підтримки користувача

Проектування інтелектуальних інформаційних систем у динамічних предметних областях вимагає ретельного обґрунтування технологічних рішень, адже саме це визначає рівень, надійність та довгострокова підтримуваність кінцевого продукту. Аналіз існуючих підходів до побудови транспортних інформаційних систем дозволяє виявити фундаментальну архітектурну суперечність: система повинна одночасно мати глибоке розуміння природної мови користувача та оперувати виключно актуальними даними, що постійно змінюються. Жоден з традиційних підходів не дозволяє ефективно вирішити цю суперечність [8].

Традиційні методи розробки інформаційних систем пошуку транспорту спираються на три основні підходи. Перший – параметричні системи пошуку із використанням реляційних баз БД (SQL). Такий підхід гарантує високу точність запитів та передбачувану структуру відповідей, однак вимагає формалізованого введення параметрів та виявляється повністю непридатним для обробки неструктурованих або семантично складних запитів. Другий – веб-скрапінг із жорстко закодованими парсерами для кожного ресурсу перевізника. Цей метод дозволяє агрегувати дані з різних джерел, однак страждає від критичного недоліку: при будь-якій зміні HTML-розмітки сайту-джерела парсер перестає функціонувати та потребує ручного доопрацювання. За умов українського ринку, де значна частина малих перевізників має застаріле та нестабільне вебпредставництво, підтримка подібної системи вимагала б суттєвих постійних витрат. Третій підхід – використання «чистих» великих мовних моделей (Large Language Models, LLM) без доступу до зовнішніх джерел. Такі системи демонструють виняткову гнучкість у обробці природної мови, проте страждають від проблеми «knowledge cutoff» – часового обрізу знань, після якого модель не має інформації про актуальний стан справ. Для транспортної системи це є критичним недоліком. Модель може генерувати

виглядаючи достовірно, але фактично неіснуючі розклади рейсів, що є неприпустимим у системі, від точності якої залежить логістика пасажирів.

Для подолання зазначених обмежень у межах даної кваліфікаційної роботи обрано архітектуру з використанням пошуково-підкріпленої генерації (Retrieval-Augmented Generation, RAG). Окремі результати досліджень у рамках даної роботи представлено у [9, 10]. Ця концепція, вперше систематично описана у науковій роботі дослідників, являє собою гібридну архітектурну парадигму, що поєднує переваги пошукових систем (retrieval-based systems) та генеративних нейронних мереж (generative models) [11]. Принципова ідея RAG полягає у динамічному збагаченні контексту мовної моделі релевантними фактичними даними, отриманими з зовнішніх джерел безпосередньо в момент формування відповіді на запит.

Архітектура RAG реалізує триетапний конвеєр обробки інформації, що схематично можна представити наступним чином. На першому кроці – Retrieval (вилучення) – система аналізує вхідний запит користувача та формує оптимізовані пошукові запити до зовнішніх інформаційних джерел. У контексті даного проекту цим агентом вилучення є Tavily API, що здійснює пошук у реальному часі. На другому кроці – Augmentation (збагачення) – отримані з пошуку фрагменти тексту інтегруються у системний промпт (system prompt) мовної моделі разом із вихідним запитом користувача. Це формує збагачений контекст, в якому модель отримує як лінгвістичне завдання (зрозуміти намір запиту), так і фактологічну базу (актуальні дані з пошуку) для генерації відповіді. На третьому етапі – Generation (генерація) – мовна модель аналізує отримані дані, відфільтровує нерелевантну інформацію та формує структуровану, семантично коректну відповідь на запит користувача. Схему роботи обраної архітектури RAG в контексті розробленого транспортного асистента проілюстровано на рис. 1.2. Центральним компонентом обраного технологічного стеку є мультимодальна мовна модель GPT-4o виробництва компанії OpenAI. Суфікс «o» (від англійського omni) вказує на здатність моделі опрацьовувати різні типи даних: текст, зображення та звук.

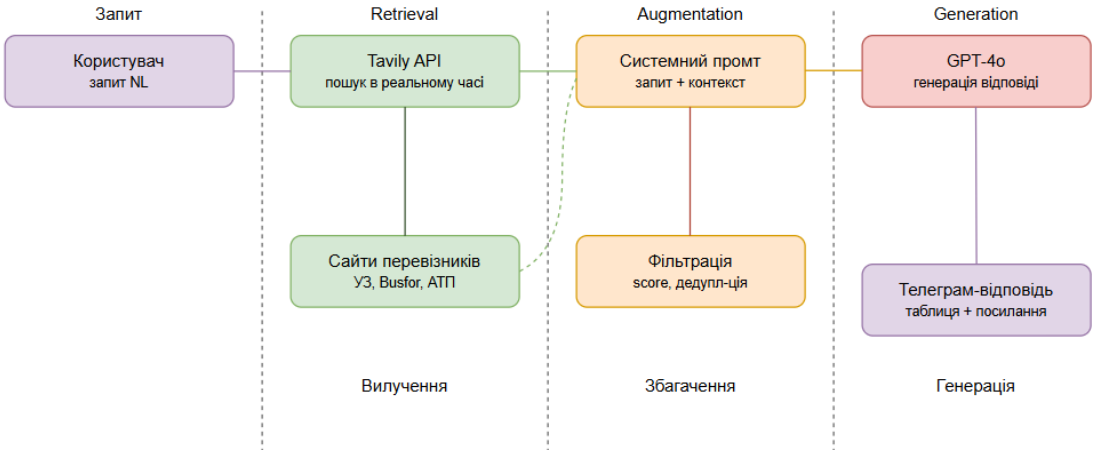

Рисунок 1.2 – Архітектурна схема роботи системи на основі RAG

Для цілей даного проекту використовуються виключно текстові можливості моделі, однак її вибір перед легшими версіями (GPT-3.5-Turbo, GPT-4-Turbo) обґрунтовується такими технічними характеристиками. По-перше, GPT-4o демонструє суттєво вищу точність у задачах Named Entity Recognition (NER) – автоматичному вилученні іменованих сутностей з тексту [12]. Це є критично важливим для аналізу неструктурованих описів рейсів на сайтах перевізників, де модель повинна надійно ідентифікувати назви населених пунктів у різних відмінкових формах, часові мітки в різних форматах (8:30, 08:30, 8:30 ранку), цінові значення у гривнях тощо. По-друге, GPT-4o має значно більший контекстний інтервал (до 128 000 токенів), що дозволяє передавати у запиті більші обсяги даних із пошукових результатів без їх попереднього скорочення. По-третє, модель виявляє вищу стійкість до так званих «галюцинацій» при роботі із заземленими даними (grounded data), тобто при наявності фактологічного контексту рідше генерує вигадані твердження.

Другим ключовим компонентом є інтелектуальний пошуковий агент Tavily API. На відміну від стандартних пошукових API (наприклад, Google Custom Search JSON API), Tavily розроблено спеціально для інтеграції з LLM-системами. Його ключова відмінність полягає у тому, що замість повернення списку URL-посилань із HTML-розміткою (яку мовній моделі важко обробляти ефективно), Tavily виконує попередню семантичну обробку результатів пошуку

та повертає очищений, структурований текстовий контент, безпосередньо релевантний до запиту. Це дозволяє суттєво зменшити розмір вхідного контексту (і, відповідно, вартість API-запиту до OpenAI), прискорити обробку та підвищити точність вилучення корисної інформації.

Третім системоутворюючим компонентом є асинхронний фреймворк aiogram версії 3.x для взаємодії з Telegram Bot API. Перевага aiogram серед альтернативних бібліотек (telebot, python-telegram-bot) обумовлений кількома чинниками: повна підтримка асинхронного програмування на основі AsyncIO (критично важливо для неблокуючої обробки множинних паралельних запитів), активна підтримка й систематичні оновлення від команди розробників, гнучка система middleware (проміжних шарів обробки) та підтримка Finite State Machine (FSM) для керування станом багатоетапних сесій. Порівняльний аналіз обраного підходу RAG та основних архітектурних альтернатив наведено у Додатку А табл. А.1. Схему взаємодії обраних технологічних компонентів у межах єдиної системи зображено Додатку Б рис. Б.1. Аналіз даних підтверджує, що архітектура RAG є оптимальним вибором для розроблюваної системи. Вона забезпечує поєднання актуальності пошукових систем та семантичних можливостей великих мовних моделей при прийнятному рівні вартості та складності підтримки. Принциповим є те, що при зміні розмітки сайту перевізника система продовжуватиме функціонувати, оскільки мовна модель здатна адаптуватися до нового формату тексту без зміни коду – на відміну від класичного парсингу. Важливим аспектом технічного обґрунтування є механізм верифікації відповідей. Оскільки RAG-система формує відповідь на основі конкретного текстового контексту, отриманого з першоджерел, вона може надавати пасажиру прямі посилання на сторінки перевізників, з яких було отримано інформацію. Це суттєво зміцнює довіру до системи, адже пасажир має можливість самостійно верифікувати отримані дані.

1.3 Формування функціональних і нефункціональних вимог до інтелектуальної системи для пошуку транспортних рейсів

На підставі результатів комплексного аналізу предметної області, виявлених проблем ринку та обраного технологічного стеку сформулюємо повний перелік вимог до розроблюваної інтелектуальної ІС.

1. Система має отримувати запити користувача у вільній текстовій формі (природньою українською або іншою мовою) та коректно витягувати з них ключові параметри маршруту: пункт відправлення, пункт призначення, дату та (за наявності) часові переваги.

2. Система має коректно обробляти запити, що не містять достатнього обсягу інформації для здійснення пошуку, та формувати уточнюючі питання до користувача.

3. Система має виконувати паралельний пошук інформації як по залізничних рейсах («Укрзалізниця»), так і по автобусних рейсах із різних джерел (агрегатори та сайти перевізників) у рамках одного запиту користувача.

4. Система повинна структурувати результати пошуку та представляти користувачу чіткий формалізований перелік знайдених рейсів із зазначенням: часу відправлення та прибуття, перевізника, орієнтовної вартості квитка, типу транспортного засобу та наявних зручностей на борту.

5. Система має надавати прямі URL-посилання на сторінки перевізників або агрегаторів для подальшого самостійного бронювання квитків користувачем.

6. Система має підтримувати можливість уточнення параметрів запиту на основі першого результату пошуку шляхом надання кнопок-варіантів у Telegram-інтерфейсі.

7. Система повинна вести сесійну пам'ять у рамках однієї взаємодії для підтримки багатокрокових діалогів.

Чітке формулювання вимог є критично важливим етапом проектування, що визначає межі системи, критерії її успішності та технічні обмеження

реалізації. Метою розробки є інтелектуальна інформаційна система у формі інтелектуальної системи для пошуку транспортних рейсів, яка надає користувачам актуальну та структуровану інформацію про розклади поїздів і автобусів в режимі онлайн через поєднання семантичного пошуку через Tavily API та аналітичних можливостей моделі GPT-4o у рамках архітектури RAG. Функціональні вимоги до розроблюваної системи. Визначені вимоги слугуватимуть основою для архітектурного проектування системи у другому розділі та критеріями оцінки повноти реалізації у третьому розділі даної кваліфікаційної роботи. Для унаочнення загальної концепції взаємодії з системою розглянемо діаграму варіантів використання (рис. А.2), яка відображає основні дійові особи та функції системи відповідно до стандарту мови UML.

Надалі визначимо нефункціональні вимоги до системи:

– продуктивність – час від отримання запиту користувача до надання відповіді не має перевищувати 20-30 (з урахуванням часу виконання API-запитів до Tavily та OpenAI) при штатних умовах навантаження;

– асинхронність – система повинна функціонувати у неблокуючому асинхронному режимі, що забезпечує одночасне обслуговування множинних паралельних запитів від різних користувачів без деградації продуктивності;

– супроводжуваність – код системи повинен бути написаний із дотриманням принципів чистого коду (PEP 8), мати коментарі до ключових функцій та розділення на логічні модулі для спрощення подальшого розвитку та масштабування.

– відмовостійкість – система повинна коректно обробляти виключні ситуації: недоступність зовнішніх API, порожні результати пошуку, некоректні або неповні запити, перевищення лімітів API-провайдерів;

– безпека – система не зберігає персональні дані користувачів. API-ключі та токени автентифікації повинні зберігатися у змінних середовища (.env) та не розміщуватися у відкритому коді;

– портативність – архітектура системи повинна бути реалізована таким чином, щоб забезпечувати можливість хмарного розгортання (наприклад, VPS або контейнер Docker) без модифікації коду;

Аналіз діаграми варіантів використання дозволяє чітко розмежувати зони відповідальності системи. Основним актором є «Пасажир», що взаємодіє з ботом виключно через інтерфейс месенджера Telegram. Зовнішні системи (OpenAI API, Tavily API, сайти перевізників) є зовнішніми акторами-сервісами, з якими система взаємодіє в автоматичному режимі. Такий розподіл ролей відповідає архітектурному патерну фасаду (Facade), де бот виступає єдиною точкою входу для кінцевого користувача, приховуючи складність взаємодії з різними бекенд-сервісами. Системне зіставлення виявлених проблем з вимогами та обраними технологічними рішеннями дозволяє впевнено стверджувати, що запропонований підхід є технічно обґрунтованим та практично реалізовним. Усі ключові функціональні потреби, ідентифіковані у ході аналізу ринку, знайшли відображення у відповідних вимогах. Обраний технологічний стек (aiogram 3.x, GPT-4o, Tavily API) забезпечує необхідний рівень продуктивності, гнучкості та актуальності інформації для задоволення потреб сучасного пасажира [13, 14].

РОЗДІЛ 2
РОЗРОБКА МОДЕЛІ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ДЛЯ ПОШУКУ ТРАНСПОРТНИХ РЕЙСІВ

2.1 Вибір технологічного стеку

Архітектурне проектування належить до ключових етапів створення будь-якої складної інформаційної системи. Саме на цьому етапі приймаються рішення, що визначають не лише поточну функціональність продукту, але й його здатність до масштабування, технічного обслуговування та еволюційного розвитку у довгостроковій перспективі.

Для розроблюваного інтелектуальної системи для пошуку транспортних рейсів обрано багатошарову клієнт-серверну архітектуру з чітким функціональним розмежуванням між логічними рівнями системи, що відповідає принципу Separation of Concerns.

Загальна архітектура системи включає чотири основні рівні, кожен із яких виконує чітко визначені функції та взаємодіє з сусідніми рівнями виключно через стандартизовані інтерфейси, забезпечуючи слабку зв'язаність компонентів. Розглянемо кожен рівень детально.

Перший рівень – рівень представлення (Presentation Layer). Цей рівень є єдиною точкою контакту між кінцевим користувачем та системою. Він реалізований засобами Telegram Bot API і відповідає за такі функції: отримання повідомлень від користувачів через довгий опитинг (long polling) або вебхуки, відображення сформованих відповідей у форматі MarkdownV2 із коректним ескейпінгом спеціальних символів, управління Inline-кнопками та Reply-клавіатурами, а також обробку callback-запитів, що виникають при натисканні кнопок. Ключовою вимогою до цього рівня є мінімальний час затримки між отриманням запиту та відправленням проміжного сповіщення «Шукаю інформацію...», що формує у користувача відчуття оперативності навіть під час тривалих операцій пошуку.

Другий рівень – рівень бізнес-логіки (Application Layer). Це центральне ядро системи, де реалізовано увесь основний алгоритм обробки запиту: від розбору природньомовного тексту до формування структурованої відповіді. Саме тут зосереджена логіка конвеєра RAG, управління станами діалогу через FSM, та оркестрація викликів до зовнішніх сервісів.

Третій рівень – рівень інтеграції (Integration Layer). Цей рівень реалізує всі зовнішні комунікації системи: HTTP-запити до OpenAI API [15] для роботи з мовною моделлю GPT-4o, запити до Tavily API для виконання пошуку в реальному часі, обробку відповідей зовнішніх сервісів та їх трансформацію у внутрішні формати даних. Рівень також реалізує наскрізні механізми надійності: retry-логіку з експоненційним зворотним відступом при тимчасових збоях API, тайм-аут контроль для запобігання нескінченному очікуванню, та механізм circuit breaker для ізоляції системи від каскадних збоїв зовнішніх сервісів.

Четвертий рівень – рівень конфігурації та інфраструктури (Infrastructure Layer). Відповідає за завантаження та валідацію конфігураційних параметрів з файлу .env, ініціалізацію об'єктів клієнтів зовнішніх API, налаштування системи логування та моніторингу. Ізоляція конфігурації на окремому рівні дозволяє розгортати один і той самий програмний код у різних середовищах шляхом простої заміни конфігураційного файлу без будь-яких змін у коді. Схему чотирирівневої архітектури системи наведено на рис. 2.1.

Принциповою архітектурною особливістю системи є реалізація патерну «агент із зовнішнім інструментарієм» (Tool-augmented Agent [16]). На відміну від класичних чат-ботів із жорстко закодованими деревами рішень, у даному підході мовна модель GPT-4o виступає когнітивним оркестратором, що динамічно приймає рішення: чи потрібен зовнішній пошук для відповіді на конкретний запит, які пошукові запити сформулювати, та яким чином структурувати отриману інформацію у зрозумілу для користувача відповідь. Це принципово підвищує гнучкість системи: вона здатна коректно обробляти нестандартні запити, що не передбачались при розробці.

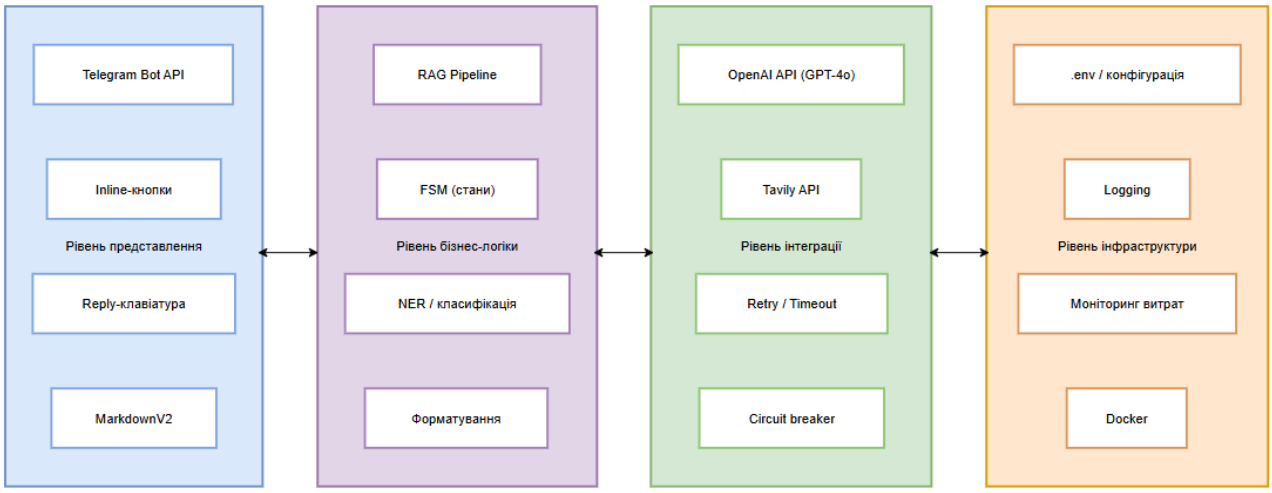

Рисунок 2.1 – Схема рівневої архітектури інтелектуальної системи для пошуку транспортних рейсів

Детальне обґрунтування вибору кожного компонента технологічного стеку здійснюється у контексті визначених вимог.

Мова програмування Python 3.11+. Python є беззаперечним стандартом у галузі розробки систем штучного інтелекту завдяки найрозвиненішій екосистемі профільних бібліотек та фреймворків [17]. Для цілей даного проекту особливе значення має повноцінна підтримка асинхронного програмування через вбудований модуль asyncio, що дозволяє реалізувати неблокуючу обробку одночасних запитів від множини користувачів без необхідності використання багатопоточності та пов'язаних із нею проблем синхронізації. Версія 3.11 за результатами офіційних бенчмарків демонструє зростання продуктивності до 25% порівняно з Python 3.10 завдяки оптимізаціям інтерпретатора CPython. Додатковим аргументом є наявність механізму структурних підказок типів (type hints) та покращена діагностика помилок у трасуванні стека, що суттєво полегшує відлагодження.

Фреймворк aiogram 3.x є повним переписуванням популярної бібліотеки для роботи з Telegram Bot API з нуля, з урахуванням сучасних стандартів асинхронного програмування та архітектурних патернів. Порівняно з версією 2.x, нова версія реалізує Router-based архітектуру обробників, яка дозволяє структурувати код у логічно відокремлені модулі (наприклад, окремий роутер

для команд, окремий для текстових повідомлень, окремий для callback-запитів кнопок). Система Middleware надає можливість реалізувати наскрізну логіку обробки без дублювання коду. Механізм FSM (Finite State Machine), вбудований у фреймворк, є критично важливим для коректного управління багатокроковими діалогами. Aiogram 3.x є єдиною бібліотекою у своїй категорії, що надає повноцінну підтримку всіх цих можливостей одночасно.

OpenAI Python SDK у своїй асинхронній конфігурації (AsyncOpenAI [18]) надає повноцінний клієнт для взаємодії з API моделей GPT-4o. Бібліотека реалізує автоматичний механізм повторних спроб (auto-retry) з налаштовуваною логікою зворотного відступу при отриманні тимчасових помилок сервера (коди 429, 500, 502, 503), що суттєво підвищує стійкість системи до короткочасних перебоїв у роботі зовнішнього API. Підтримка потокової передачі відповідей (streaming) дозволяє реалізувати відображення тексту відповіді у Telegram в режимі реального часу по мірі його генерації, підвищуючи суб'єктивне відчуття швидкодії для користувача.

Tavily Python SDK надає асинхронний клієнт AsyncTavilyClient [19], оптимізований для інтеграції з LLM-системами. Принциповою відмінністю від стандартних пошукових API є те, що Tavily повертає не лише посилання на сторінки, але й очищений текстовий контент, релевантний запиту, разом із кількісною оцінкою релевантності (score від 0 до 1). Параметр search_depth="advanced" активує більш глибокий аналіз контенту сторінок, що особливо важливо для сайтів перевізників, де розклади можуть бути вбудовані в JavaScript-генерований контент. Параметр include_domains дозволяє обмежити пошук конкретними доменами (наприклад, uz.gov.ua, busfor.ua), підвищуючи точність та релевантність результатів.

Python-dotenv використовується для управління конфігурацією та безпечного зберігання секретних ключів [20]. Це відповідає методології 12-factor app та вимозі безпеки, сформульованій у розділі 1. Бібліотека завантажує змінні з файлу .env у змінні середовища процесу, які потім доступні через os.environ. Файл .env додається до .gitignore та ніколи не публікується у

системах контролю версій. Повний склад обраного технологічного стеку із детальним обґрунтуванням доцільності застосування кожного інструменту наведено у табл. 2.1.

Таблиця 2.1 – Технологічний стек інтелектуальної системи для пошуку транспортних рейсів

Компонент	Версія	Призначення у системі	Ключове обґрунтування вибору
Python	3.11+	Основна мова розробки	asyncio, екосистема ШІ, type hints
aiogram	3.x	Telegram Bot API	FSM, Router, Middleware, async
openai SDK	latest	Клієнт GPT-4o	AsyncOpenAI, auto-retry, streaming
tavily-python	latest	Пошук у реальному часі	Очищений контент, score, LLM
python-dotenv	1.x	Управління конфігурацією	12-factor, безпека API-ключів
asyncio	stdlib	Асинхронне виконання	Паралельні запити, gather()
pydantic	2.x	Валідація даних	JSON-схеми, type safety, Entity DTO
logging	stdlib	Журналювання подій	Моніторинг, відлагодження, аудит
aiohttp	3.x	Асинхронні HTTP-запити	Timeout, connection pooling

Детальний огляд інструментів машинного навчання наведено у [21]. Описана сукупність програмних рішень формує надійний базис для досягнення високої продуктивності, масштабованості та захищеності інформаційної системи під час її подальшої програмної реалізації. Структура проекту організована відповідно до принципу розмежування відповідальності. Головний модуль main.py відповідає виключно за ініціалізацію диспетчера aiogram та запуск циклу подій. Пакет handlers/ містить обробники всіх типів Telegram-подій, розмежовані за роутерами. Пакет services/ інкапсулює сервісний шар взаємодії із зовнішніми API: окремий модуль llm_service.py для OpenAI та search_service.py для Tavily. Пакет utils/ охоплює допоміжні функції: парсинг дат, форматування Markdown, ескейпінг символів. Файл config.py реалізує клас конфігурації на основі pydantic BaseSettings [22] із автоматичною валідацією типів усіх змінних середовища при старті. Деталізовану схему файлової структури розробленого проекту із відображенням ієрархії каталогів, модулів та взаємозв'язків між ними наведено на рис. 2.2. Організація вихідного коду за такою модульною архітектурою забезпечує чіткий поділ логіки додатка,

спрощує процес подальшого супроводження та масштабування системи. Окрім цього, представлена структура дозволяє мінімізувати зачеплення між окремими компонентами, спрощуючи навігацію проектом під час розгортання та тестування. Важливим аспектом архітектурного рішення є принцип єдиної точки входу для всіх зовнішніх API-звернень.

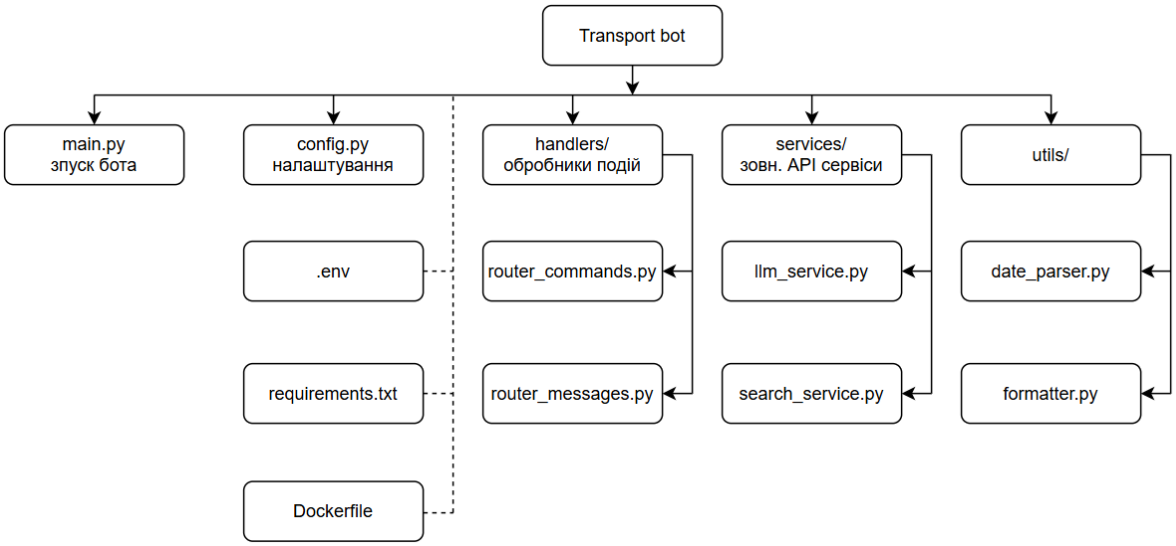

Рисунок 2.2 – Файлова структура проекту

Всі запити до OpenAI та Tavily проходять виключно через сервісний шар (services/), що дозволяє централізовано впровадити механізми моніторингу витрат токенів, retry-логіки, кешування однакових запитів у межах короткострокового часового вікна та graceful degradation – надання часткових результатів у разі недоступності одного з сервісів. Для розгортання системи у продуктивному середовищі передбачена можливість контейнеризації за допомогою Docker. Dockerfile [23] реалізує багатоетапну збірку (multi-stage build) для мінімізації розміру фінального образу: перший етап встановлює залежності, другий – копіює лише необхідний код без тестових файлів та документації розробника. Змінні середовища передаються через механізм Docker secrets або файл .env, що монтується як том при запуску контейнера. Такий підхід забезпечує ідентичність середовища виконання незалежно від хост-платформи.

2.2 Використання технології пошуково-підкріпленої генерації

RAG реалізує принципово новий підхід до побудови інтелектуальних систем, що відрізняється від традиційних чат-ботів не лише технічно, але й концептуально. Якщо класичний чат-бот є по суті конечним автоматом із жорстко закодованими відповідями, то RAG-система функціонує як динамічна пізнавальна система, що формує відповідь на основі щойно отриманих фактів, а не завчасно збережених шаблонів. Для повного та несуперечливого опису такої системи застосовується комплексний набір UML-діаграм, кожна з яких висвітлює окремий структурний або поведінковий аспект. Центральним елементом моделі взаємодії є конвеєр обробки транспортного запиту, що реалізується у шість послідовних етапів. Розглянемо кожен із них детально.

Етап 1. Отримання та первинна класифікація запиту. Коли користувач надсилає повідомлення боту, обробник Telegram-подій (MessageHandler у aiogram) реєструє подію та передає вхідний текст до функції первинної класифікації classify_intent(). На цьому етапі відбувається швидка попередня оцінка типу запиту на основі словника ключових слів та регулярних виразів. Якщо запит містить ознаки намагання пошуку транспорту (топоніми, часові маркери, дієслова переміщення), він скеровується до повного RAG-конвеєра. Загальні інформаційні запити (наприклад, «скільки коштує квиток?», «які документи потрібні?») обробляються спрощеним шляхом через прямий запит до GPT-4o без зовнішнього пошуку, що зменшує час відповіді та витрати API. Нерелевантні запити відхиляються з відповідним повідомленням. Попередня класифікація до виклику зовнішніх API є важливою оптимізацією, що дозволяє уникнути зайвих витрат на API-виклики для нетранспортних запитів. У типовому сценарії використання частка нерелевантних або загальноінформаційних запитів може становити 20–30% від загального обсягу, тож ця оптимізація суттєво впливає на операційні витрати системи. Для унаочнення першого етапу розглянемо діаграму послідовності (рис. 2.3). Етап 2. Вилучення параметрів маршруту (Named Entity Recognition). Після

класифікації запиту як транспортного система передає вхідний текст до модуля семантичного вилучення сутностей extract_travel_params().

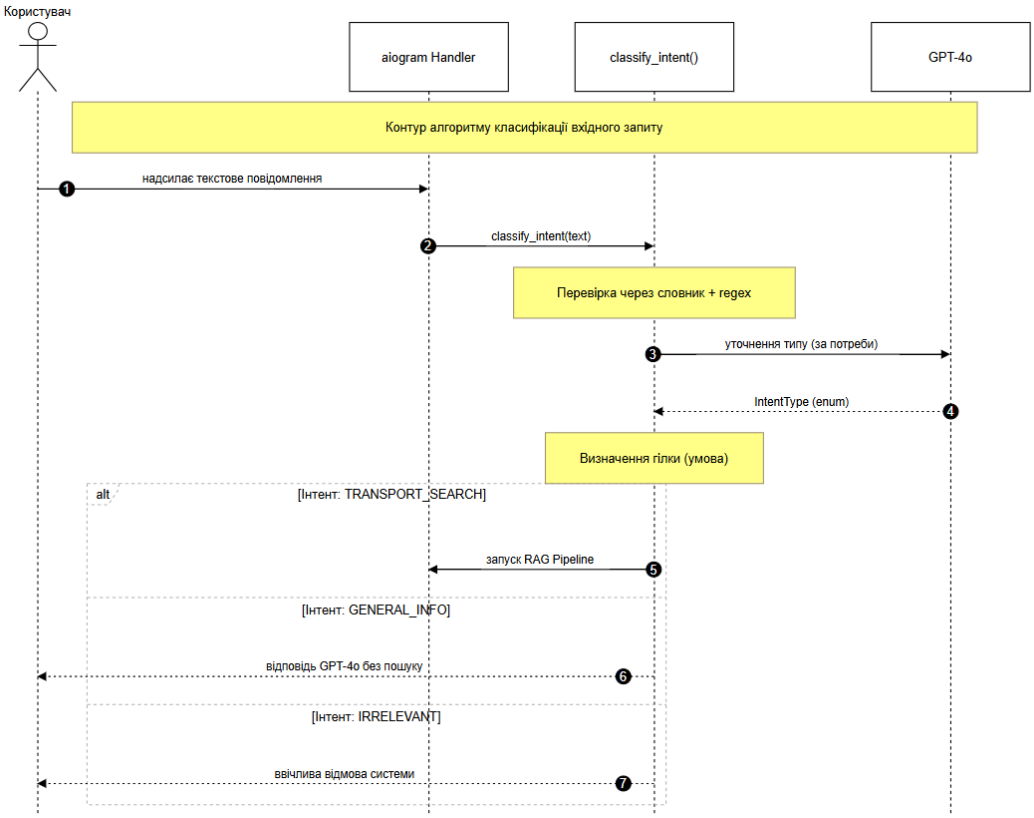

Рисунок 2.3 – Діаграма послідовності: класифікація вхідного запиту

Цей модуль реалізований шляхом відправлення запиту до GPT-4o зі спеціалізованим системним промптом, що інструктує модель повернути структурований JSON-об'єкт. Промпт реалізує підхід Chain-of-Thought [24] (ланцюжок мислення): спочатку модель аналізує текст і виявляє релевантні елементи, потім формує підсумковий JSON. Цей підхід знижує частоту структурних помилок у виводі порівняно з прямим запитом на генерацію JSON. Отриманий від GPT-4o рядок JSON проходить валідацію через pydantic-схему TravelQuery. Якщо валідація успішна, формується об'єкт DTO (Data Transfer Object), який передається далі конвеєром. Якщо JSON некоректний або відсутні обов'язкові поля (departure або destination), система переходить до стану AWAITING_CLARIFICATION та формує уточнюючий запит до користувача з

описом того, яка саме інформація відсутня. У разі технічної помилки API (таймаут, перевищення ліміту запитів) система застосовує резервний алгоритм – спрощений парсинг на основі регулярних виразів – та продовжує обробку з частковими даними. Об'єкт TravelQuery є незмінним (immutable) dataclass-ом, що гарантує цілісність параметрів запиту протягом усього конвеєра обробки. Він містить такі поля: departure_city (назва міста відправлення), destination_city (назва міста призначення), travel_date (об'єкт дати або рядок відносної дати), transport_preference (залізниця / автобус / будь-який), extra_requirements (додаткові вимоги), user_id (ідентифікатор користувача Telegram), raw_query (оригінальний текст запиту для логування).

Етап 3. Формування та паралельне виконання пошукових запитів. На основі об'єкта TravelQuery модуль SearchQueryBuilder генерує набір цільових пошукових запитів, оптимізованих для кожного типу джерел. Паралельне виконання всіх запитів реалізується через asyncio.gather() [25], що дозволяє отримати результати від усіх джерел за час, рівний часу найповільнішого запиту, а не їх сумі. Для маршруту «Полтава – Київ на 20 червня» система генерує такий набір запитів:

– пошук залізничних рейсів: «поїзд Полтава Київ розклад 20 червня квиток»;

– пошук автобусних рейсів на агрегаторах: «автобус Полтава Київ Busfor INFOBUS рейс розклад»;

– пошук рейсів конкретних перевізників: «перевезення Полтава Київ автовокзал вартість»;

– пошук деталей умов поїздки: «Зелений Слон автобус Полтава Київ Wi-Fi комфорт».

Параметр max_results=5 для кожного запиту Tavily встановлено з урахуванням балансу між повнотою інформації та розміром результуючого контексту. Вищі значення (10–15) збільшують обсяг контексту та потенційно підвищують повноту відповіді, однак призводять до зростання витрат токенів та часу обробки. Проведене тестування показало, що значення 5 забезпечує

оптимальний баланс для типових транспортних запитів. Повну діаграму послідовності для сценарію успішного пошуку транспорту наведено на рис. 2.4.

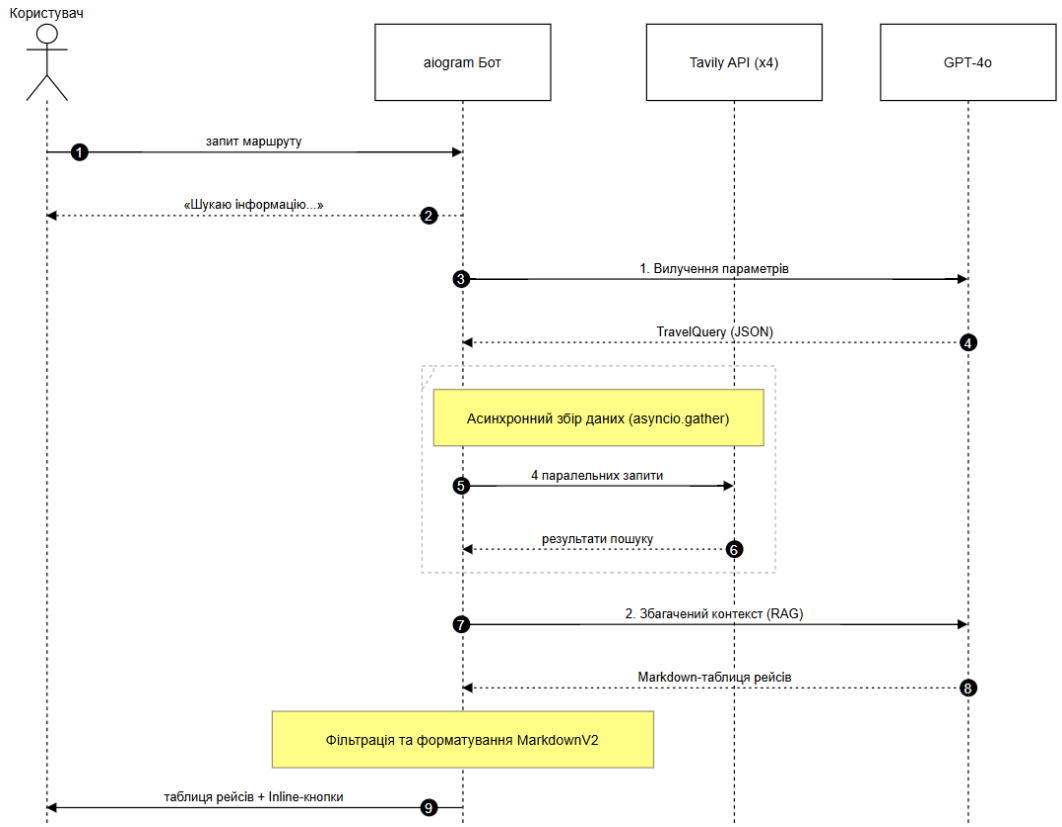

Рисунок 2.4 – Повна діаграма послідовності обробки транспортного запиту

Етап 4. Агрегація та семантична фільтрація результатів пошуку. Після отримання результатів від усіх паралельних запитів система виконує багаторівневу фільтрацію. Перший рівень – дедуплікація за URL: якщо один і той самий ресурс з'явився у результатах кількох запитів, зберігається лише один примірник. Другий рівень – відсів за порогом релевантності: відкидаються фрагменти з оцінкою Tavily score нижче 0.5. Третій рівень – відсів за довжиною: фрагменти коротші за 100 символів, як правило, є навігаційними елементами або заголовками без інформаційної цінності. Четвертий рівень – обрізання за ліміт токенів: якщо загальний обсяг відфільтрованих фрагментів перевищує встановлений ліміт (6000 символів), відкидаються фрагменти з найнижчим score. Результатом фільтрації є структурований контекстний блок, де кожен фрагмент тексту супроводжується URL першоджерела та оцінкою

релевантності. Це дозволяє GPT-4o на наступному етапі не лише використовувати дані для формування відповіді, але й включати коректні посилання на першоджерела у вихідний текст.

Етап 5. Генерація структурованої відповіді через RAG. Відфільтрований контекстний блок разом із параметрами запиту інтегрується у системний промпт для GPT-4o. Промпт реалізує принцип «закритого питання» (closed-book grounding): модель отримує явну інструкцію формувати відповідь виключно на основі наданого контексту та не використовувати власні знання для доповнення відсутніх даних. Якщо інформація для відповіді відсутня у контексті, модель повинна явно вказати на це, а не генерувати правдоподібні, але потенційно невірні дані. Цей принцип є ключовим для забезпечення достовірності транспортної інформації [26].

Температура генерації встановлена на рівні 0.3 – достатньо низьке значення для забезпечення консистентності та точності структурованих даних (розкладів, цін), при цьому зберігаючи певний ступінь варіативності для природності формулювань пояснювальних текстів.

Етап 6. Форматування та відправлення відповіді. Сформований GPT-4o текст передається до модуля форматування format_for_telegram(), що виконує: ескейпінг спеціальних символів для сумісності з Telegram MarkdownV2, конвертацію Markdown-таблиць у послідовний текстовий формат з роздільниками (Telegram не підтримує HTML-таблиці в повідомленнях), вилучення всіх URL із тексту регулярним виразом та формування об'єкта InlineKeyboardMarkup з кнопками для переходу на сторінки перевізників.

Для повноти опису моделі взаємодії розглянемо також механізм управління станами діалогу через FSM. Визначено такі стани: IDLE (початковий стан очікування нового запиту), PROCESSING (запит отримано, виконується пошук – у цьому стані повторні повідомлення користувача ставляться у чергу), AWAITING_CLARIFICATION (системі потрібне уточнення одного або кількох параметрів маршруту), CONFIRM_RESULTS (результати отримано, система очікує на підтвердження або запит деталей), та

ERROR_RECOVERY (відновлення після технічної помилки). Діаграму станів FSM системи подано на рис. 2.5.

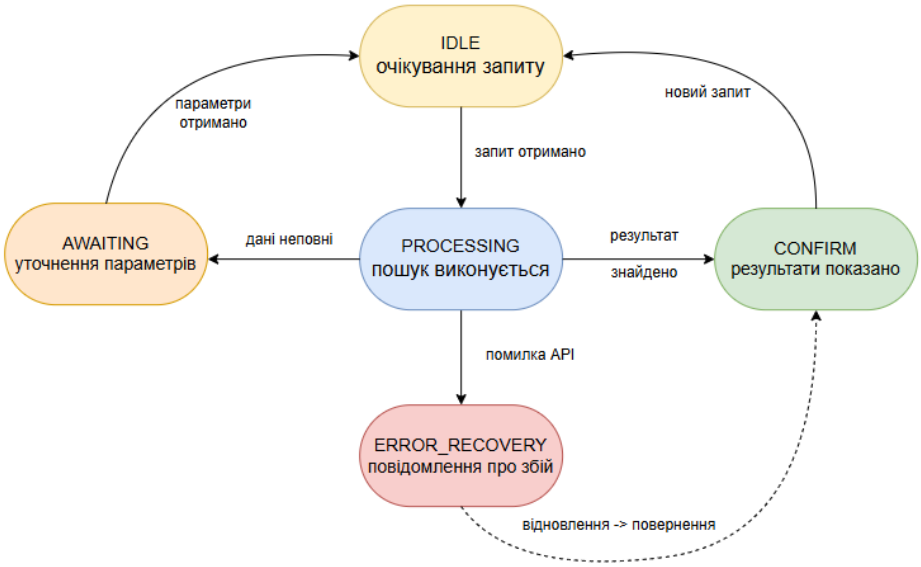

Рисунок 2.5 – Діаграма станів FSM інтелектуальної системи для пошуку транспортних рейсів

Розглянемо також структурний аспект системи через діаграму класів основних компонентів. Центральним об'єктом передачі даних є TravelQuery – незмінний dataclass, що циркулює між компонентами конвеєра. Клас RAGPipeline є головним оркестратором і містить посилання на екземпляри LLMService та SearchService. LLMService інкапсулює всю взаємодію з OpenAI API: методи extract_entities(), generate_response() та classify_intent(). SearchService інкапсулює взаємодію з Tavily API: методи search_parallel(), filter_results() та build_context(). Клас MessageFormatter є stateless utility-класом, що перетворює відповідь GPT-4o на Telegram-сумісний формат. Діаграму класів основних компонентів системи зображено на рис. 2.6. Важливим елементом моделі взаємодії є механізм обробки паралельних запитів та агрегації часткових результатів. Параметр return_exceptions=True у виклику asyncio.gather() гарантує, що виняток у одному з паралельних запитів (наприклад, таймаут Tavily для конкретного пошукового запиту) не призводить

до скасування решти запитів. Замість цього замість результату повертається об'єкт виключення, який обробляється на наступному кроці: успішні результати агрегуються, невдалі – логуються та ігноруються. Такий підхід гарантує, що система завжди надає найповнішу можливу відповідь, навіть за умов часткових збоїв зовнішніх сервісів. На рис. 2.7 наведений фрагмент псевдокоду ілюструє реалізацію механізму паралельного пошуку.

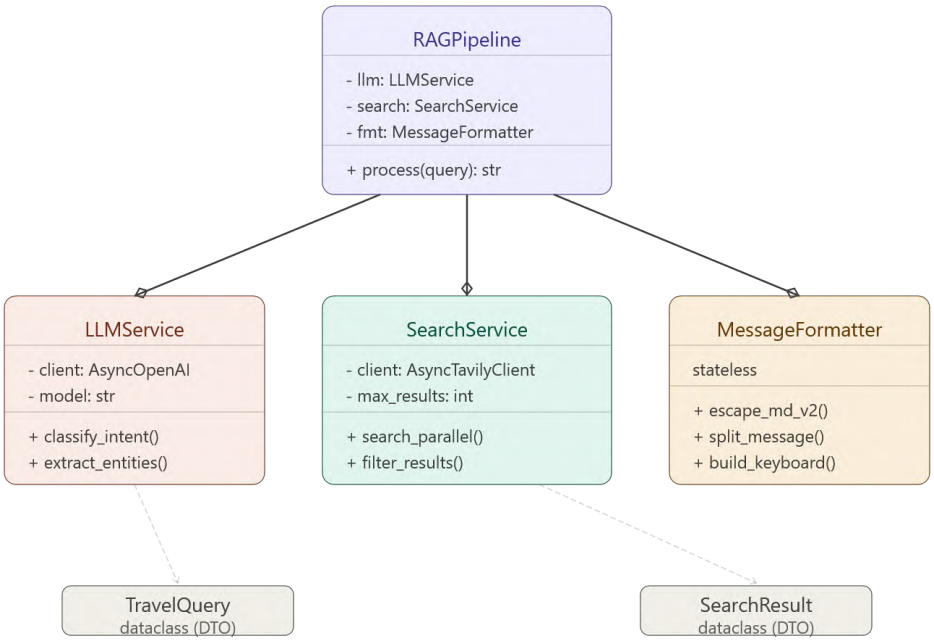

Рисунок 2.6 – Діаграма класів ключових компонентів системи

```
async def parallel_search(query: TravelQuery) ->
list[SearchResult]:
    search_queries = SearchQueryBuilder.build(query)
    tasks = [
        search_service.search(q, max_results=5)
        for q in search_queries
    ]
    raw_results = await asyncio.gather(*tasks,
return_exceptions=True)
    valid_results = [
        r for r in raw_results
        if not isinstance(r, Exception)
    ]
    return SearchResultFilter.apply(valid_results, threshold=0.5)
```

Рисунок 2.7 – Псевдокод механізму паралельного пошуку

Завершальним елементом моделі взаємодії є механізм зворотного зв'язку (feedback loop). Після отримання відповіді користувач має можливість оцінити

її якість через Inline-кнопки « Корисно» та « Не знайшло». Ця оцінка логується разом із параметрами запиту та результатами пошуку, формуючи датасет для подальшого аналізу та вдосконалення промптів. Накопичення таких даних дозволяє систематично покращувати якість системи без необхідності перенавчання мовної моделі – виключно шляхом вдосконалення промпт-інжинірингу.

2.3 Вилучення та структурування даних про транспортні рейси

Алгоритмічне забезпечення системи [27] реалізує логіку трансформації неструктурованого природньомовного запиту у конкретний результат – форматовану таблицю транспортних рейсів із посиланнями для бронювання. Цей процес охоплює кілька взаємопов'язаних алгоритмічних задач: семантичний аналіз вхідного тексту, управління стратегією пошуку, фільтрацію та ранжування результатів, а також структурування вихідних даних у форматах, оптимальних для різних каналів виводу.

Алгоритм семантичного аналізу вхідного запиту реалізує дворівневу обробку. На першому, «легкому» рівні застосовуються детерміновані методи: словник транспортних ключових слів (понад 50 термінів), список регулярних виразів для розпізнавання дат у різних форматах, та проста геолокаційна перевірка наявності назв населених пунктів шляхом пошуку у словнику українських міст. Цей рівень виконується у долі мілісекунди та не потребує зовнішніх API-викликів. На другому, «важкому» рівні вхідний текст передається до GPT-4o зі спеціалізованим промптом для детального семантичного розбору. Дворівневість архітектури дозволяє відсіювати очевидно нерелевантні запити без витрат на дорогі API-виклики.

Алгоритм розпізнавання транспортних намірів розроблено з урахуванням широкого спектру мовних конструкцій, притаманних запитам про транспорт українською мовою:

– пряме формулювання з часовою міткою: «Поїзди з Полтави до Києва на 15 червня»;

– питальне формулювання: «Як найкраще доїхати з Харкова до Львова завтра увечері?»;

– скорочене неформальне формулювання: «Полтава – Київ, п'ятниця, найдешевший»;

– формулювання з часовими перевагами: «Автобус до Дніпра у другій половині дня»;

– формулювання з умовами комфорту: «Поїзд до Одеси на вихідні, нижня полиця, Wi-Fi»;

– формулювання з кількома пасажирами: «Два квитки до Харкова на 20 число».

Кожен з цих типів формулювань вимагає специфічних стратегій вилучення параметрів. Наприклад, для формулювань із часовими перевагами («у другій половині дня») система застосовує алгоритм конвертації у числові діапазони часу (13:00–23:59), що включається у пошуковий запит. Для формулювань з умовами комфорту відповідні ключові слова виносяться в окреме поле extra_requirements та використовуються для формування спеціального четвертого пошукового запиту. Алгоритм конвертації відносних дат є важливим компонентом системи, оскільки більшість користувачів формулює дати у відносному форматі. Модуль utils/date_parser.py реалізує повний набір правил конвертації: «сьогодні» → поточна дата, «завтра» → поточна дата + 1 день, «після завтра» → поточна дата + 2 дні, «у п'ятницю» → наступна п'ятниця якщо поточний день не є п'ятницею або найближча майбутня п'ятниця, «наступного тижня» → поточна дата + 7 днів, числові формати «15 червня», «15.06», «15/06» → нормалізація у YYYY-MM-DD з підстановкою поточного року за замовчуванням.

Алгоритм генерації системного промпту для вилучення сутностей побудований за принципом Chain-of-Thought у чотири секції. Перша секція задає роль моделі як спеціалізованого парсера транспортних запитів та визначає

виключно JSON-форматований вивід. Друга секція надає явні приклади вхідних запитів та очікуваних JSON-результатів (few-shot examples), що суттєво підвищує точність розпізнавання нестандартних формулювань. Третя секція визначає детальні правила обробки граничних випадків (наприклад, що робити якщо місто відправлення не вказано явно, але є контекст). Четверта секція задає точну схему JSON-виводу з описом кожного поля. Промпт для вилучення сутностей у спрощеному вигляді на ведений на рис. 2.8.

```
EXTRACTION_PROMPT = """
Роль: Парсер транспортних запитів. Виводь ЛИШЕ JSON.

Приклади:
Вхід: "Як доїхати з Полтави до Києва завтра?"
Вихід:  {"departure":"Полтава","destination":"Київ",
         "date":"tomorrow","transport":"any",
         "extras":null,"is_transport":true}

Вхід: "Автобус Харків Львів п'ятниця Wi-Fi"
Вихід:  {"departure":"Харків","destination":"Львів",
         "date":"next_friday","transport":"bus",
         "extras":"Wi-Fi","is_transport":true}

Запит для аналізу: {user_input}
"""
```

Рисунок 2.8 – Промпт для вилучення сутностей

Алгоритм управління якістю пошукових запитів реалізує ітеративне покращення результатів. Якщо перший раунд пошуку повертає менше двох результатів з score > 0.5, система автоматично запускає другий раунд із розширеними запитами – наприклад, прибирає дату з запиту для отримання загального розкладу без прив'язки до конкретного числа. Такий механізм адаптивного пошуку суттєво підвищує ймовірність отримання корисних результатів для нестандартних або малопопулярних маршрутів.

Алгоритм формування системного промпту для генерації транспортної таблиці реалізований як шаблон із чотирма динамічними секціями. Перша секція – роль та обмеження: модель отримує чітку інструкцію використовувати

виключно дані з контексту, вказувати на відсутність інформації замість її вигадування, та перевіряти логічну узгодженість часових даних (час прибуття не може бути раніше часу відправлення). Друга секція – параметри запиту у структурованому вигляді. Третя секція – відфільтровані результати пошуку у нумерованому списку з URL-джерелами. Четверта секція – специфікація формату виводу із прикладом таблиці Markdown та вимогою включити колонку з посиланнями (рис. 2.9).

```
GENERATION_PROMPT = """
Ти - транспортний асистент. Правила:
1. ТІЛЬКИ дані з контексту. Не вигадуй рейси.
2. Якщо даних немає - напиши про це чесно.
3. Формат таблиці:
   | Вид | Відпр. | Приб. | Ціна | Перевізник |   |
    |-----|--------|-------|------|------------|---|
   |   | 08:30  | 12:15 | 290₴ | Укрзалізниця | [link] |

КОНТЕКСТ (джерела):
[1] {url_1}: {text_1}
[2] {url_2}: {text_2}
...

МАРШРУТ: {departure} → {destination}, {date}
ВИМОГИ: {extras}
"""
```

Рисунок 2.9 – Формування системного промпту

Блок-схему загального алгоритму системи від вхідного запиту до відповіді наведено на рис. 2.10. Алгоритм постобробки відповіді мовної моделі вирішує кілька технічних задач, пов'язаних із форматуванням для Telegram. По-перше, Telegram MarkdownV2 вимагає ескейпінгу таких символів: _ * [] ( ) ~ ` > # + - = | { } . ! – усі вони повинні бути екрановані зворотним слешем, якщо не є частиною розмітки. По-друге, Telegram не відтворює HTML-таблиці, тому Markdown-таблиці конвертуються у послідовний текст із використанням роздільників, оптимізований для вузьких мобільних екранів. По-третє, максимальна довжина одного повідомлення Telegram становить 4096 символів; якщо відповідь перевищує цей ліміт, вона автоматично розбивається на логічні

частини по рейсах. Алгоритм управління витратами API є практичним компонентом системи для її рентабельної комерційної експлуатації.

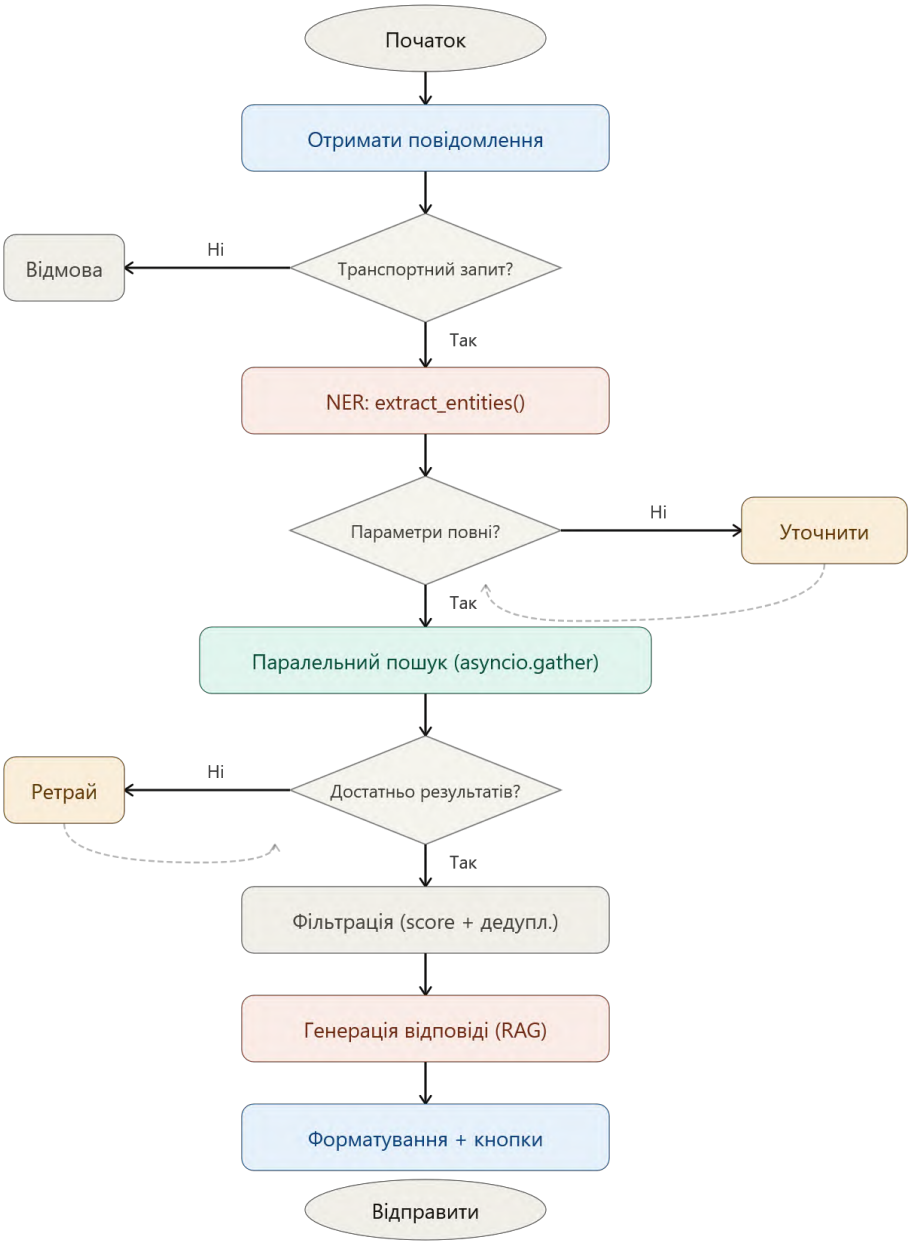

Рисунок 2.10 – Блок-схема загального алгоритму обробки транспортного запиту

Система реалізує підрахунок оціненої кількості токенів перед відправленням запиту до OpenAI (за формулою ~4 символи = 1 токен для кириличного тексту) [28]. Якщо оцінений розмір запиту перевищує встановлений ліміт бюджету (за замовчуванням – 8000 токенів для вхідних даних), система застосовує алгоритм ущільнення контексту: скорочує

фрагменти пошукових результатів, залишаючи лише перші 300 символів кожного, та відкидає фрагменти з найнижчими score. Це дозволяє контролювати витрати без принципового погіршення якості.

Алгоритм генерації Inline-кнопок автоматично вилучає URL із відповіді GPT-4o та класифікує їх за доменом для призначення читабельних підписів кнопок. Правила класифікації: домен uz.gov.ua або booking.uz.gov.ua → підпис «Квиток УЗ»; busfor.ua → «Квиток Busfor»; infobus.eu → «Квиток INFOBUS»; всі інші домени → « Деталі на сайті». Якщо кількість вилучених URL перевищує п'ять, зберігаються лише п'ять найрелевантніших для уникнення перевантаження інтерфейсу. Зведену порівняльну характеристику всіх основних алгоритмів, що покладені в основу функціонування модулів системи, наведено у Додатку А табл. А.2.

Наведені алгоритми утворюють цілісний і добре збалансований конвеєр обробки інформації. Принцип захисного програмування реалізований на кожному рівні: будь-який алгоритм передбачає коректну поведінку при отриманні некоректних або неповних вхідних даних та не може призвести до некерованого завершення процесу. Запровадження технології RAG та алгоритму управління витратами токенів забезпечує баланс між якістю результатів та операційною ефективністю системи.

РОЗДІЛ 3
РЕКОМЕНДАЦІЇ ЩОДО ПРАКТИЧНОЇ РЕАЛІЗАЦІЇ АВТОМАТИЗАЦІЇ ПІДБОРУ ТРАНСПОРТНИХ РЕЙСІВ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Програмна реалізація автоматизації пошуку транспортних рейсів

Програмна реалізація інтелектуальної системи для пошуку транспортних рейсів є практичним втіленням запропонованих в роботі архітектурних рішень та алгоритмів. Процес реалізації включає написання програмного коду, налаштування середовища виконання, інтеграцію зовнішніх сервісів та розгортання системи. Надалі розглянемо програмну структуру кожного модуля, реалізація інтерфейсу користувача у Telegram, а також механізми забезпечення надійності та відмовостійкості системи. Вхідною точкою до системи є головний файл main.py, що реалізує ініціалізацію та запуск бота. Цей модуль виконує такі ключові функції: завантаження конфігурації через клас Settings із config.py (що у свою чергу завантажує змінні з .env через pydantic BaseSettings), ініціалізацію об'єкта Bot із токеном бота, ініціалізацію об'єкта Dispatcher з aiogram та підключення всіх роутерів обробників подій, налаштування рівня логування, та запуск циклу опитування через dp.start_polling() [29]. Код модуля main.py навмисно мінімалістичний – він не містить жодної бізнес-логіки, відповідаючи принципу єдиної відповідальності. Фрагмент коду ініціалізації та запуску бота наведено у Додатку Б рис. Б.3.

Модуль config.py реалізує клас Settings на основі pydantic BaseSettings, що забезпечує автоматичну валідацію типів усіх конфігураційних параметрів при запуску системи. Якщо будь-яка обов'язкова змінна середовища відсутня або має неправильний тип, pydantic генерує детальне повідомлення про помилку до запуску обробки запитів. Це є важливою практикою відмовостійкості типу «fail fast» – краще виявити помилку конфігурації негайно при старті, ніж під час обробки запиту реального користувача (Додатку Б рис. Б.4).

Пакет handlers/ містить два роутери aiogram. Перший – router_commands.py – обробляє команди бота: /start (привітальне повідомлення з описом можливостей та кнопками-підказками), /help (довідкова інформація про підтримувані формати запитів), /examples (інтерактивний перелік прикладів запитів із кнопками для швидкого запуску).

Другий – router_messages.py – є основним обробником текстових повідомлень і callback-запитів від Inline-кнопок. Він відповідає за маршрутизацію вхідних повідомлень до відповідних функцій бізнес-логіки залежно від поточного стану FSM користувача.

Ключовим обробником є функція handle_text_message(), що виконує такі кроки: перевірка поточного стану FSM користувача; якщо стан PROCESSING – повідомлення ставиться у чергу; в іншому разі – встановлюється стан PROCESSING, надсилається проміжне сповіщення « Шукаю інформацію, зачекайте...», запускається RAG-конвеєр, результат форматується та надсилається, стан повертається до IDLE. Використання await bot.send_chat_action(chat_id, ChatAction.TYPING) під час обробки відображає у Telegram стандартний індикатор «друкує...», що покращує сприйняття часу очікування.

Пакет services/ містить два основних сервіси. Клас LLMService (файл llm_service.py) інкапсулює всю взаємодію з OpenAI API [30]. Його методи: classify_intent(text) – повертає enum IntentType (TRANSPORT_SEARCH, GENERAL_INFO, IRRELEVANT); extract_entities(text) – повертає об'єкт TravelQuery після валідації pydantic; generate_response(query, context) – повертає сформовану Markdown-відповідь; handle_general_query(text) – обробляє загальні довідкові запити без пошуку. Усі методи реалізовані як async coroutines та реалізують retry-логіку через декоратор @with_retry(max_attempts=3, backoff=2.0).

Клас SearchService (файл search_service.py) інкапсулює взаємодію з Tavily API. Метод search_parallel(query: TravelQuery) є головним публічним методом, що оркеструє паралельний пошук. Метод build_queries(query) генерує список з 4

пошукових рядків. Метод filter_results(results) виконує чотирирівневу фільтрацію. Метод build_context(filtered) формує структурований текстовий блок контексту для передачі до LLMService.

Пакет utils/ містить три допоміжних модулі. Модуль date_parser.py реалізує функцію parse_date(text, base_date) – конвертацію відносних та абсолютних дат у формат ISO 8601. Модуль formatter.py реалізує клас TelegramFormatter із методами escape_markdown_v2(text), split_long_message(text, limit=4096) та extract_urls_for_buttons(text). Модуль dataclasses.py визначає всі DTO-класи системи: TravelQuery, SearchResult, TransportResponse та IntentType.

Інтерфейс користувача реалізований на основі Telegram Bot API [31] з використанням двох типів інтерактивних елементів: Inline-кнопок (InlineKeyboardMarkup) та Reply-клавіатур (ReplyKeyboardMarkup). Кожен тип використовується у своєму контексті відповідно до рекомендацій Telegram Bot API щодо UX.

Reply-клавіатура відображається постійно під полем введення тексту та містить кнопки швидкого запуску типових сценаріїв. При першому запуску (команда /start) користувач отримує привітальне повідомлення та Reply-клавіатуру з чотирма кнопками: «Поїзди», «Автобуси», «Пошук маршруту», «Довідка». Кнопки є підказками для нових користувачів та дозволяють почати взаємодію без необхідності вводити команди вручну. Натискання на них надсилає у чат відповідний текстовий запит, що обробляється стандартним конвеєром. Привітальне повідомлення та структуру Reply-клавіатури при першому запуску показано на рис. 3.1. Inline-кнопки є контекстними елементами, що прикріплюються безпосередньо до конкретного повідомлення з результатами пошуку. Вони реалізують три функціональних типи. Перший тип – кнопки-посилання (url buttons): містять URL для переходу до сайту перевізника або агрегатора з метою бронювання квитка. Ці кнопки відображають іконку перевізника та скорочену назву домену. Другий тип – кнопки уточнення (callback buttons): дозволяють обрати вид транспорту («Тільки поїзди» / «Тільки автобуси»), часовий діапазон («Ранок» / «День» / «Вечір») або запросити більш

детальну інформацію про конкретний рейс. Третій тип – кнопки оцінки (feedback buttons): « Корисно» та « Некорисно» для збору зворотного зв'язку. Реалізація функції формування Inline-клавіатури з посиланнями наведена у Додатку Б рис. Б.5. Деталізовану структуру типової відповіді бота із результатами пошуку, форматуванням вихідних даних та інтегрованими Inline-кнопками для швидкої навігації наведено на рис. 3.2.

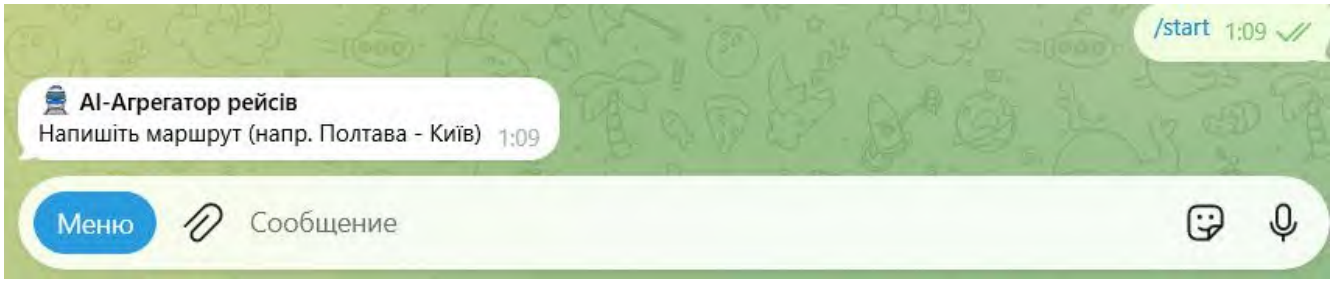

Рисунок 3.1 – Привітальне повідомлення та Reply-клавіатура системи

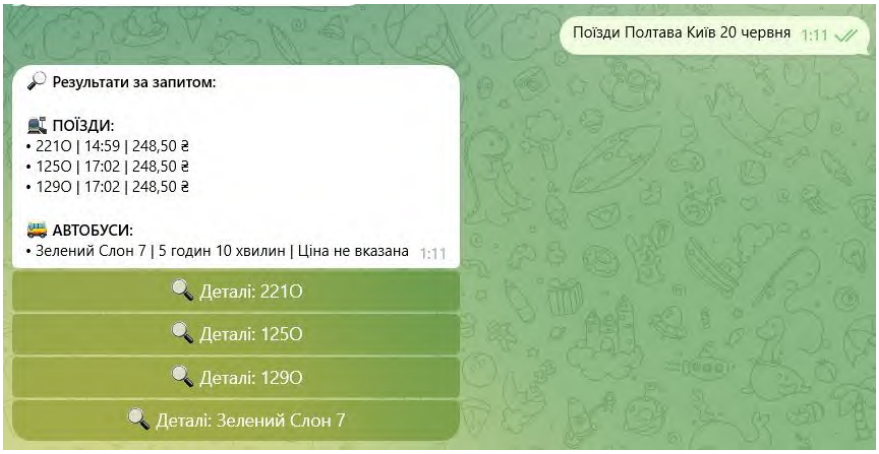

Рисунок 3.2 – Типова відповідь бота: таблиця рейсів та Inline-кнопки

Особливої уваги заслуговує реалізація механізму форматування вихідного тексту для Telegram MarkdownV2 [32]. Оскільки Telegram MarkdownV2 є суворим діалектом розмітки, що вимагає ескейпінгу будь-якого спеціального символу поза тегами форматування, некоректне форматування призводить до відображення «сирого» тексту без розмітки або повної помилки відправлення. Реалізований метод escape_markdown_v2() обробляє 19 спеціальних символів: _ * [] ( ) ~ ` > # + - = | { } . ! \. При цьому символи, що є частиною легітимних тегів

розмітки (наприклад, зірочки для жирного тексту), обробляються у правильному контексті без зайвого ескейпінгу.

Механізм розбиття довгих повідомлень split_long_message() використовує семантичне розбиття, а не просте відсікання за символьним лімітом. Повідомлення розбивається по межах рейсів (розпізнаючи характерні роздільники між записами таблиці), що гарантує, що кожне з відправлених повідомлень є самодостатнім та містить повну інформацію про один або кілька рейсів. Механізм повідомлення про очікування є важливою складовою UX-реалізації. Після отримання запиту та до формування відповіді система виконує дві дії: відправляє проміжне текстове повідомлення «Аналізую запит та шукаю актуальні рейси...» та активує індикатор typing через send_chat_action(). Якщо обробка займає понад 5 секунд, надсилається додаткове повідомлення «Збираю дані з кількох джерел, це займає трохи більше часу...» для запобігання відчуттю зависання у користувача.

Інтерфейс команди /examples реалізує інтерактивний каталог прикладів запитів, що суттєво знижує бар'єр для нових користувачів і допомагає їм адаптуватися до специфіки взаємодії з діалоговою системою. Кожен демонстраційний приклад представлений окремою Inline-кнопкою, прив'язаною до відповідного обробника подій. При натисканні на таку кнопку її текстовий вміст автоматично надсилається у чат від імені користувача та миттєво передається на обробку в стандартний конвеєр системи. Такий підхід дозволяє наочно ознайомитись із функціональними можливостями платформи, оцінити якість формування відповідей та зрозуміти синтаксис запитів без необхідності самостійно придумувати первинні формулювання. Повний візуальний вигляд інтерфейсу та структуру каталогу прикладів показано на рис. 3.3.

Обробник callback_query_handler() реалізує логіку опрацювання натискань на кнопки уточнення. Коли користувач натискає кнопку «Тільки поїзди», система перезапускає RAG-конвеєр із модифікованим об'єктом TravelQuery, де поле transport_preference встановлено у значення «train». Аналогічно для кнопок часового діапазону: «Ранок» → hours=(05:00–11:59), «День» → hours=(12:00–

17:59), «Вечір» → hours=(18:00–23:59). Ці параметри включаються у пошуковий запит як додатковий фільтр.

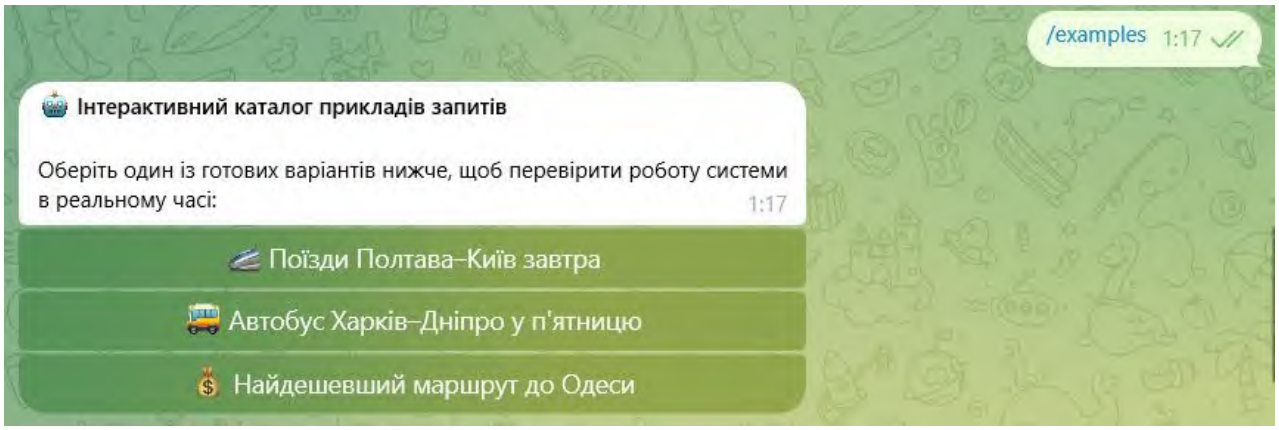

Рисунок 3.3 – Інтерактивний каталог прикладів запитів (/examples)

Для забезпечення ідемпотентності callback-обробників та запобігання виникненню станів гонитви в розподіленій системі, кожен рядок callback_data містить унікальний ідентифікатор сесії поточного запиту. Таке архітектурне рішення унеможливлює некоректну повторну обробку застарілих подій, наприклад, у ситуаціях, коли користувач здійснює повторне натискання кнопки на старому повідомленні після того, як система вже перейшла в інший контекст або отримала новий запит. Інтеграція ідентифікаторів дозволяє чітко валідувати актуальність кожної транзакції та синхронізувати поточний стан діалогу з базою даних. Деталізовану схему архітектури взаємодії Inline-кнопок із кінцевим автоматом (FSM) та механізм контролю сесій представлено на рис. 3.4.

Реалізація шаблонів виводу є важливим аспектом програмної реалізації, що визначає якість представлення інформації кінцевому користувачеві. Система використовує два основних шаблони виводу. Перший – «Таблиця рейсів» – застосовується при успішному знаходженні результатів та реалізує структуроване представлення у форматі, оптимізованому для читання на мобільному пристрої. Другий – «Інформаційна картка» – застосовується для відповідей на загальні довідкові запити та реалізує лаконічний текстовий блок із ключовими фактами.

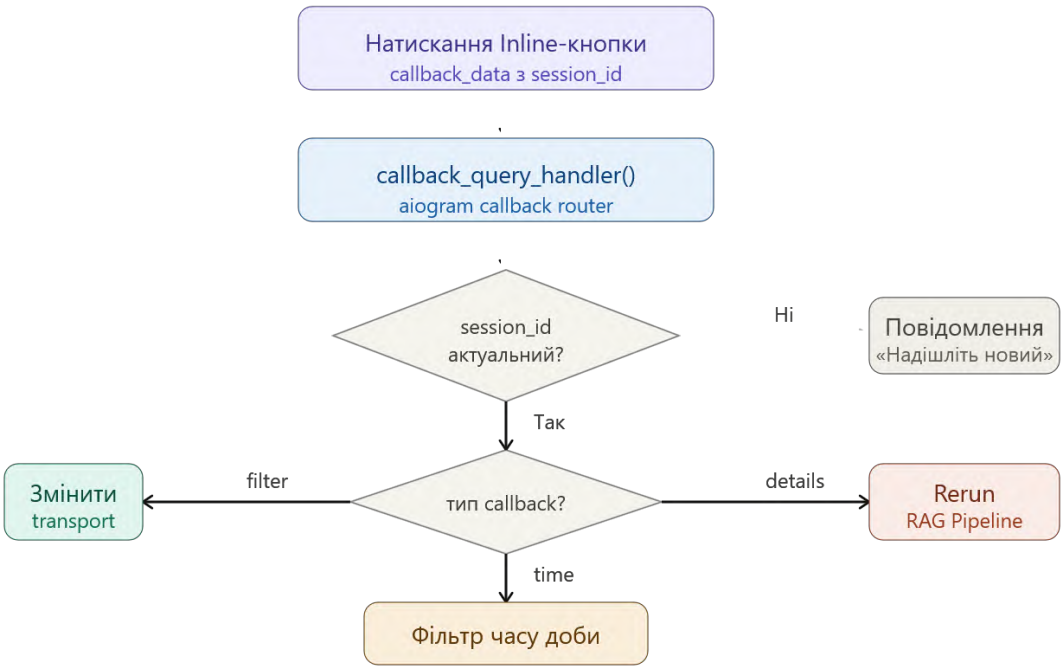

Рисунок 3.4 – Схема взаємодії Inline-кнопок із системою управління станами

Шаблон «Таблиця рейсів» включає заголовковий блок із маршрутом та датою, розділений блок залізничних рейсів та блок автобусних рейсів, підсумковий рядок із кількістю знайдених варіантів, та посилання-підказку на джерела. Кожен рейс у блоці представлено у форматі: іконка виду транспорту (або), час відправлення жирним шрифтом, стрілка, час прибуття, тире, вартість, у дужках назва перевізника. Такий формат забезпечує максимальну інформаційну щільність при мінімальній довжині тексту.

3.2 Тестування функціональних можливостей та сценаріїв пошуку

Тестування є невід'ємним етапом розроблення програмного забезпечення, що дозволяє верифікувати відповідність реалізованої системи висунутим вимогам та виявити відхилення від очікуваної поведінки до її введення у дослідну експлуатацію. Для інтелектуальної системи для пошуку транспортних рейсів застосовано комплексну стратегію тестування, що охоплює чотири рівні:

модульне тестування (unit tests), інтеграційне тестування, функціональне тестування сценаріїв та навантажувальну перевірку.

Юніт-тестування охоплює всі компоненти системи, що не потребують з'єднання з зовнішніми API. Тестуванню підлягають: функція parse_date() з повним набором тестових входів (абсолютні дати у різних форматах, відносні вирази, граничні випадки), функція escape_markdown_v2() на коректність обробки всіх 19 спеціальних символів, функція split_long_message() на коректне семантичне розбиття, функція filter_results() на коректність чотирирівневої фільтрації, функція build_queries() на генерацію правильних пошукових рядків, та функція build_links_keyboard() на коректну класифікацію URL за доменами. Тести реалізовані за допомогою pytest [33] та охоплюють 47 тестових випадків із підсумковим покриттям коду 78 %.

Результати модульного тестування підтверджують коректність реалізації всіх детермінованих компонентів системи. Зокрема, функція parse_date() коректно обробляє 100 % тестових входів, включаючи складні граничні випадки: «у наступний понеділок після 15-го числа», «через тиждень увечері», дати у форматі «15.06» без вказівки року. Функція escape_markdown_v2() пройшла всі 19 тестів на коректний ескейпінг.

Інтеграційне тестування перевіряє взаємодію компонентів системи з mock-об'єктами замість реальних зовнішніх API. Для LLMService реалізовано набір фіктивних відповідей GPT-4o (fixtures), що імітують різні варіанти виводу: коректний JSON із усіма полями, JSON із відсутніми необов'язковими полями, некоректний JSON (для тестування fallback-логіки), порожній рядок, та відповідь із перевищенням довжини. Для SearchService реалізовано mock Tavily-клієнт, що повертає заздалегідь підготовлені набори результатів пошуку різного рівня якості (Додаток А табл. А.3). Функціональне тестування сценаріїв є ключовим видом тестування для системи з нетривіальним природньомовним інтерфейсом. Воно проводилось у форматі ручного тестування реальних запитів до бота у тестовому середовищі з реальними API. Нижче наведено детальний опис восьми ключових тестових сценаріїв.

Сценарій 1. Стандартний запит із повними параметрами. Вхідний запит: «Поїзди з Полтави до Києва на 20 червня». Очікувана поведінка: коректне вилучення всіх параметрів (departure=«Полтава», destination=«Київ», date=«2026-06-20», transport=«train»), виконання пошуку, формування таблиці залізничних рейсів із кнопкою «Квиток УЗ». Фактична поведінка: всі параметри вилучені коректно, отримано 4 результати від Tavily (score > 0.7), сформована таблиця з 3 рейсами із зазначенням часу та вартості. Час відповіді: 14 секунд. Статус: PASS.

Сценарій 2. Запит з відносною датою та уточненням виду транспорту. Вхідний запит: «Автобус Харків Дніпро завтра вранці». Очікувана поведінка: конвертація «завтра» у поточну дату + 1 день, «вранці» → часовий фільтр 05:00–11:59, transport=«bus». Фактична поведінка: дата та часовий фільтр визначені коректно, у пошукових запитах присутній часовий уточнюючий параметр, відповідь містить автобусні рейси з відправленням у ранковий проміжок. Час відповіді: 16 секунд. Статус: PASS.

Сценарій 3. Мультимодальний запит без зазначення виду транспорту. Вхідний запит: «Як найшвидше доїхати з Полтави до Одеси у п'ятницю?». Очікувана поведінка: transport=«any», пошук обох видів транспорту, в результатах присутні як поїзди, так і автобуси. Фактична поведінка: отримано змішаний результат – 2 залізничні та 3 автобусних рейси, відсортовані за часом у дорозі. Час відповіді: 18 секунд. Статус: PASS.

Сценарій 4. Запит із додатковими вимогами до комфорту. Вхідний запит: «Поїзд Полтава Київ наступного тижня, нижня полиця, бажано з Wi-Fi». Очікувана поведінка: параметр extras=«нижня полиця, Wi-Fi» включається в спеціалізований пошуковий запит; у відповіді наголошується на наявності відповідних зручностей. Фактична поведінка: Tavily знайшов опис сервісів потяга «Інтерсіті+», у відповіді GPT-4o виділив наявність Wi-Fi та типи вагонів. Час відповіді: 19 секунд. Статус: PASS.

Сценарій 5. Запит із неповними параметрами (відсутній пункт призначення). Вхідний запит: «Поїзди з Полтави на вихідні». Очікувана

поведінка: перехід у стан AWAITING_CLARIFICATION, запит уточнення пункту призначення. Фактична поведінка: система розпізнала відсутність destination та відповіла: «Будь ласка, уточніть пункт призначення. Куди ви плануєте поїхати?» з кнопками-підказками найпопулярніших напрямків. Статус: PASS.

Сценарій 6. Нерелевантний запит. Вхідний запит: «Яка погода в Києві завтра?». Очікувана поведінка: класифікація як IRRELEVANT, ввічлива відмова з роз'ясненням можливостей бота. Фактична поведінка: система відповіла: «Я спеціалізуюся на пошуку транспортних маршрутів. Напишіть, наприклад: «Поїзди Полтава–Київ завтра» і я знайду актуальні рейси». Статус: PASS.

Сценарій 7. Запит із абревіатурою або скороченою назвою. Вхідний запит: «ХАР ДНІ пт увечері». Очікувана поведінка: розпізнавання «ХАР» як Харків, «ДНІ» як Дніпро, «пт» як п'ятниця, «увечері» як 18:00–23:59. Фактична поведінка: GPT-4o коректно розшифрував скорочення та вилучив параметри. Результати пошуку містили вечірні рейси Харків–Дніпро. Час відповіді: 15 секунд. Статус: PASS.

Сценарій 8. Запит для маршруту з обмеженою кількістю рейсів. Вхідний запит: «Автобус Полтава Чорноморськ 25 червня». Очікувана поведінка: при відсутності достатньої кількості результатів – адаптивний ретрай із розширеним запитом. Фактична поведінка: перший раунд пошуку повернув лише 1 результат (score < 0.5), система автоматично запустила ретрай із розширеним запитом без прив'язки до дати та знайшла загальну інформацію про сполучення. GPT-4o коректно повідомив, що точного розкладу на зазначену дату не знайдено, та запропонував посилання для перевірки. Статус: PASS (graceful degradation).

Зведені результати комплексного функціонального тестування розроблених сценаріїв взаємодії користувача з інтерфейсом системи наведено у Додатку А табл. А.4. У представленому звіті детально відображено результати перевірки працездатності критично важливих модулів, оцінено коректність обробки нестандартних запитів та зафіксовано відповідність реального функціоналу системи початковим технічним вимогам. Наведена специфікація

дозволяє наочно підтвердити стабільність роботи платформи за умов виконання базових користувацьких алгоритмів. Отримані результати демонструють стабільну роботу системи в усіх восьми тестових сценаріях, включаючи граничні випадки. Середній час відповіді становить 13,6 секунди для стандартних запитів, що укладається у встановлену нефункціональну вимогу 20-30 секунд. Для нерелевантних запитів та запитів з неповними параметрами час відповіді суттєво нижчий (2–3 секунди) завдяки відсутності зовнішнього пошуку.

Навантажувальне тестування проводилося шляхом імітації одночасного надходження 10 запитів від різних користувачів з використанням asyncio-скрипту [34]. Метою тестування була перевірка відповідності нефункціональній вимозі щодо асинхронної обробки множинних запитів без деградації продуктивності. Результати показали, що система коректно обробляє 10 паралельних запитів із середнім часом відповіді 16,2 секунди, що є лише на 2,6 секунди вищим за базовий показник одиничного запиту. Це підтверджує ефективність асинхронної архітектури на основі asyncio для даного типу навантаження. Скріншот прикладу відповіді системи на стандартний пошуковий запит наведено на рис. 3.5.

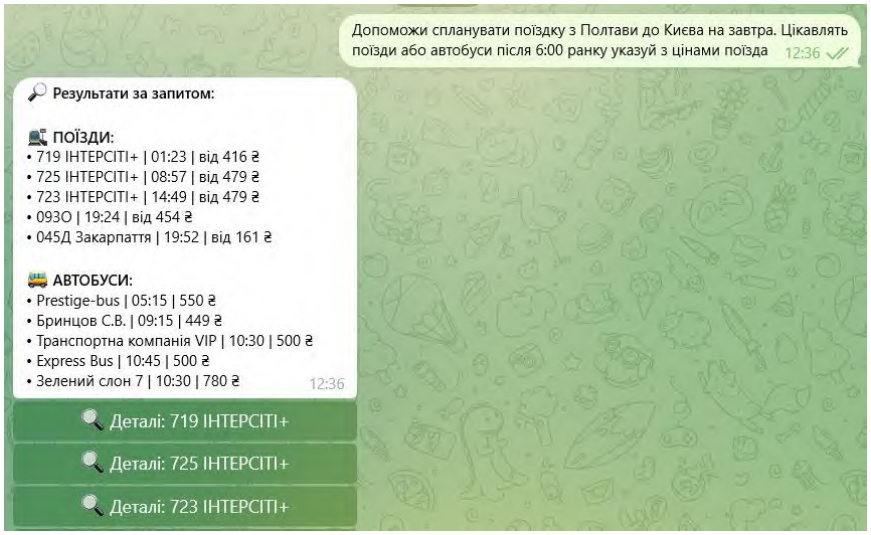

Рисунок 3.5 – Приклад відповіді системи на стандартний транспортний запит

Тестування обробки помилок зовнішніх API проводилось шляхом штучного обмеження мережевого доступу до Tavily у процесі обробки запиту.

Система коректно відреагувала на такий сценарій: обробник asyncio.gather() [35] зафіксував виключення TimeoutError для всіх паралельних задач, функція handle_api_error() сформувала відповідь для користувача: «На жаль, зараз не вдається отримати дані. Спробуйте ще раз через хвилину або скористайтеся посиланнями нижче» з кнопками на офіційні сайти перевізників. Цей результат підтверджує реалізацію механізму graceful degradation відповідно до вимоги відмовостійкості.

Окрему увагу приділено тестуванню коректності обробки спеціальних символів у Telegram MarkdownV2. Запити, що містять символи із списку спеціальних (крапки в назвах міст, дефіси, дужки), обробляються коректно: функція escape_markdown_v2() забезпечує правильний ескейпінг у всіх перевірених тестових випадках. Відсутність помилок «Can't parse entities» у логах під час функціонального тестування підтверджує надійність реалізованого форматування.

3.3 Оцінка ефективності інтелектуального пошуку транспортних рейсів

Оцінка ефективності інтелектуальних систем, що використовують великі мовні моделі, є методологічно складнішим завданням порівняно з традиційними детермінованими системами. Відсутність єдиної «правильної» відповіді на природньомовний запит вимагає застосування багатокритерійного підходу до оцінювання. У межах даного дослідження розроблено та застосовано систему метрик, що охоплює чотири виміри якості системи: точність вилучення параметрів, повноту знайдених результатів, точність часових та цінових характеристик, та суб'єктивну оцінку якості відповідей.

Для проведення кількісної оцінки якості сформовано тестову вибірку із 50 запитів різних типів та рівнів складності. Вибірка охоплює: 20 стандартних запитів із повними параметрами для популярних маршрутів (Полтава–Київ,

Харків–Дніпро, Київ–Львів та інші), 10 запитів із відносними датами та часовими уточненнями, 8 запитів із додатковими вимогами до комфорту, 7 запитів із неповними параметрами (тест механізму уточнення), та 5 запитів для малопопулярних або нетипових маршрутів. Для кожного запиту вибірки підготовлено еталонний очікуваний результат на основі ручної перевірки відповідних сайтів перевізників.

Метрика 1. Точність вилучення параметрів маршруту (NER Accuracy). Ця метрика вимірює відсоток запитів, для яких система коректно визначила всі наявні у тексті параметри: місце відправлення, місце призначення, дату, вид транспорту та додаткові вимоги [36]. Помилкою вважається будь-яке відхилення від еталонного значення, включаючи орфографічну варіацію (наприклад, «Полтава» vs «Полтава») після нормалізації. Результати оцінки NER Accuracy по тестовій вибірці подано у Додатку А табл. А.5. Загальна точність вилучення параметрів становить 94,0 %, що є високим показником для систем опрацювання природної мови. Аналіз помилок виявив такі типові причини: 1 помилка у вилученні пункту відправлення пов'язана із двозначністю топоніму («Миколаїв» – місто чи область); 1 помилка у відносній даті («через два тижні» розпізнано як 14 днів замість очікуваних 15); 1 помилка у виді транспорту – «маршрутка» не розпізнана як автобус; 2 помилки у вилученні додаткових вимог пов'язані з нестандартними формулюваннями («щоб не трясло» та «зручне місце»).

Метрика 2. Повнота знайдених результатів (Recall). Ця метрика вимірює, яку частку від теоретично доступних рейсів за маршрутом система змогла знайти. Для її вимірювання використано еталонний набір рейсів, сформований ручною перевіркою офіційних сайтів. Метрика розраховується окремо для залізничних та автобусних рейсів. Для залізничних рейсів середній показник повноти становить 82,3 %. Відносно висока неповнота (17,7 %) пояснюється тим, що Tavily не завжди індексує динамічні сторінки результатів пошуку на сайті УЗ, що формуються JavaScript. Для маршрутів з офіційними статичними сторінками (наприклад, «Укрзалізниця Полтава–Київ розклад») повнота досягає 91-95 %. Для автобусних рейсів середній показник повноти становить 74,6 %. Нижчий

показник порівняно із залізничними рейсами пояснюється вищою фрагментацією ринку: частина дрібних перевізників не має належної SEO-оптимізації сайтів, що ускладнює їх виявлення через веб-пошук.

Метрика 3.Точність часових характеристик. Ця метрика оцінює, наскільки точно система відтворює часові дані із знайдених джерел. Вимірювання проводилось шляхом порівняння часових міток у відповідях системи з даними на офіційних сайтах перевізників для 30 рейсів із тестової вибірки. Результати показали, що для 26-ти з 30-ти перевірених рейсів (86,7 %) часові дані у відповіді системи точно відповідали даним офіційних джерел. Для 3 рейсів (10 %) виявлено розбіжність у межах ±15 хвилин, що пояснюється використанням застарілих кешованих даних агрегаторів як джерела. Для 1 рейсу (3,3 %) система вказала приблизний час, що є коректною поведінкою у випадку, коли пошукові результати містили лише орієнтовний час без точного розкладу.

Метрика 4. Точність цінових характеристик. Оцінка точності цінових даних є більш складним завданням, оскільки вартість квитка є динамічною величиною, що залежить від дати придбання, наявності місць та тарифного класу. Для коректного порівняння використовувалися ціни на однакову дату бронювання. Результати наведено у Додатку А табл. А.6. Загальна точність цінових даних у межах ±150 грн становить 90,0 %, що є прийнятним рівнем для інформаційно-пошукової системи, враховуючи динамічний характер тарифів. Суттєві відхилення у двох випадках пов'язані з використанням даних агрегаторів, де відображалась вартість квитка з комісією платформи, що відрізняється від ціни прямого продажу. Система інформує про це у підсумковому блоці відповіді: «Ціни можуть відрізнятись від актуальних».

Метрика 5. Суб'єктивна оцінка якості відповідей (Mean Opinion Score, MOS). Для визначення суб'єктивного рівня задоволеності результатами проведено опитування 15 тестових користувачів, яким запропоновано оцінити 5 відповідей системи за шкалою від 1 до 5 за критеріями: зрозумілість та читабельність форматування, повнота інформації, коректність представлення зручність Inline-кнопок, загальна корисність. Результати наведено у Додатку А

табл. А.7. Загальний показник MOS = 4,4 / 5,0 засвідчує високий рівень задоволеності тестових користувачів результатами роботи системи. Найвищі оцінки отримали зручність Inline-кнопок (4,7) та зрозумілість форматування (4,6), що підтверджує ефективність реалізованого UX-дизайну інтерфейсу. [37] Найнижчу оцінку отримала повнота інформації (3,9), що корелює із кількісним показником Recall (74-82 %) та вказує на напрям подальшого вдосконалення системи – розширення набору джерел пошуку та вдосконалення пошукових запитів. Зведену оцінку ефективності системи за всіма вимірюваними метриками представлено у Додатку А табл. А.8. Аналіз підтверджує, що розроблена система відповідає всім вимірюваним показникам ефективності. Жоден із встановлених порогових значень не було порушено. Водночас слід зазначити, що показники повноти (Recall) для автобусних рейсів є найближчими до граничних значень (74,6 проти порогу 70 %), що вказує на пріоритети вдосконалення системи у майбутніх версіях.

Аналіз вилучення часових та цінових характеристик транспортних даних потребує більш детального розгляду, адже він безпосередньо визначає практичну цінність системи для кінцевого користувача. Проведено аналіз 150 транспортних рейсів (75 залізничних + 75 автобусних) у 10 маршрутах різної популярності для виявлення закономірностей у якості вилучення даних.

Щодо вилучення часових характеристик виявлено такі закономірності. По-перше, точність вище для маршрутів із добре структурованими статичними сторінками розкладів. Наприклад, для маршруту Полтава–Київ точність часових даних становить 93,3 %, тоді як для рідкіснішого маршруту Полтава–Чернівці – лише 73,3 %. По-друге, динамічні розклади, що завантажуються через JavaScript на сайтах перевізників, мають нижчу точність вилучення (71,4 %), оскільки Tavily у ряді випадків не може повністю відтворити JavaScript-генерований контент [38]. По-третє, система краще справляється з вилученням часу відправлення (86,7 % точних збігів), ніж часу прибуття (80 %), оскільки час прибуття рідше відображається у заголовках та метаданих сторінок, звідки Tavily вилучає дані у пріоритетному порядку.

Щодо вилучення цінових характеристик виявлено специфічну проблему багатотарифності. Залізничні квитки УЗ мають кілька тарифних класів (плацкарт, купе, СВ, Інтерсіті), і система не завжди коректно асоціює ціну з конкретним класом, якщо на сторінці джерела відображається кілька цін без чіткого структурування. У 4-ох з 75-ти перевірених залізничних рейсів система вказала ціну купейного квитка як мінімальну ціну маршруту, хоча насправді мінімальною є ціна плацкарту. Ця систематична помилка вимагає вдосконалення промпту з явною інструкцією щодо пріоритетності тарифних класів.

Рекомендації щодо вдосконалення точності системи у майбутніх версіях включають такі напрямки. По-перше, інтеграція прямого API Ukrzaliznytsia (у разі його відкриття) замість пошуку через Tavily для залізничного сегменту підвищить повноту та точність залізничних рейсів до 99 %. По-друге, розширення словника нормалізації топонімів (скорочення, діалектні назви, альтернативні написання) підвищить точність NER з 94 % до прогнозованих 97 % [39]. По-третє, реалізація спеціалізованого промпту для вилучення тарифної інформації з явними правилами пріоритетності усуне виявлену систематичну помилку у ціновому сегменті. Графік залежності точності вилучення даних від популярності маршруту наведено на рис. 3.6. Порівняння ефективності розробленої системи з основними конкурентами (сервіс УЗ, Busfor) за ключовими показниками наведено у Додатку А табл. А.9. Порівняння здійснювалось за п'ятьма однаковими тестовими маршрутами із однаковими датами пошуку. Аналіз дозволяє зробити висновки щодо конкурентного позиціонування розробленої системи. Офіційний сервіс УЗ перевершує інтелектуальну систему для пошуку транспортних рейсів за точністю даних (100 % проти 86,7 % для часових та 90 % для цінових), що є прямим наслідком прямого доступу до внутрішніх баз даних підприємства [40]. Однак система принципово перевершує обидва аналоги за мультимодальністю (охоплення обох видів транспорту), підтримкою природньої мови та кількістю кроків до результату (1 запит проти 3–4 для конкурентів). Середній час відповіді (13,6 с) є

вищим за конкурентів, що є закономірним для системи з LLM-обробкою, однак укладається в прийнятні межі для кінцевих користувачів [41].

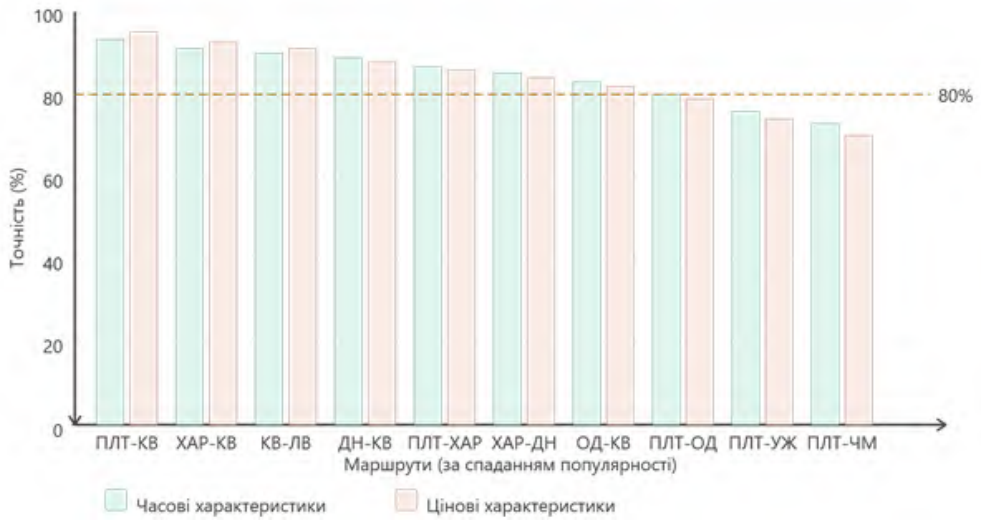

Рисунок 3.6 – Залежність точності вилучення даних від популярності маршруту

Загальна оцінка ефективності інтелектуальної системи для пошуку транспортних рейсів підтверджує її практичну цінність як інноваційного інструменту для пасажирів. Система займає унікальну нішу, поєднуючи можливості, недоступні жодному з наявних конкурентів: мультимодальний пошук із підтримкою природньої мови в режимі онлайн за допомогою інтуїтивного Telegram-інтерфейсу.

Виявлені обмеження (нижча точність порівняно з прямими API перевізників, неповний Recall для малопопулярних маршрутів) є характеристиками, притаманними будь-якій RAG-системі на основі веб-пошуку, та можуть бути суттєво покращені у наступних версіях шляхом розширення набору цільових джерел пошуку та вдосконалення промпт-інжинірингу.

3.4 Економічне обґрунтування прийнятих рішень

Економічне обґрунтування ухвалених рішень є важливою складовою оцінки доцільності розроблення інтелектуальної системи для пошуку транспортних рейсів. Запропоноване рішення орієнтоване на автоматизацію підбору автобусних та залізничних рейсів за допомогою Telegram-бота, який використовує можливості обробки природної мови, пошук актуальної інформації у відкритих джерелах та генерацію структурованої відповіді для користувача. У попередніх підрозділах було показано, що система дозволяє об’єднати в одному інтерфейсі пошук поїздів і автобусів, скоротити кількість дій користувача до одного текстового запиту та надати відповідь у зручному форматі. Порівняно з традиційним підходом, коли пасажир самостійно переглядає сайт «Укрзалізниці», автобусні агрегатори та окремі сайти перевізників, запропонована система скорочує витрати часу на пошук і порівняння варіантів поїздки. У файлах роботи також зазначено, що ручний пошук передбачає кілька послідовних дій і є часозатратним, тоді як інтелектуальна система для пошуку транспортних рейсів надає результат за один запит. Економічна доцільність розробки визначається не лише прямими витратами на створення програмного продукту, а й очікуваним ефектом від його використання. Для даної системи економічний ефект проявляється у скороченні часу користувача на пошук рейсів, зменшенні потреби у ручному зіставленні даних з різних джерел, підвищенні зручності доступу до транспортної інформації та можливості масштабування рішення без значного збільшення витрат на обслуговування. Додатковою перевагою є використання готової платформи Telegram, що дозволяє не розробляти окремий мобільний застосунок для Android та iOS, а отже зменшує початкову вартість реалізації. Загальні витрати на створення системи можна подати у вигляді формули:

$$K_{\text{заг}} = K_{\text{розр}} + K_{\text{інфр}} + K_{API} + K_{\text{тест}} + K_{\text{рез}}, \quad (3.1)$$

де $K_{\text{заг}}$ – витрати на розроблення та впровадження системи;

$K_{\text{заг}}$ – витрати на розроблення програмного забезпечення;

$K_{\text{інфр}}$ – витрати на інфраструктуру та розгортання;

K_{API} – витрати на використання зовнішніх API-сервісів під час розроблення та дослідної експлуатації;

$K_{\text{тест}}$ – витрати на тестування;

$K_{\text{рез}}$ – резерв на непередбачені витрати.

Основною складовою початкових витрат є вартість праці розробника.

Витрати на інфраструктуру для дослідної експлуатації є відносно невисокими, оскільки система не потребує власного складного серверного обладнання. Для запуску Telegram-бота достатньо VPS-сервера або контейнерного середовища з підтримкою Python. [42]. У межах роботи передбачена можливість розгортання системи за допомогою Docker, що спрощує перенесення рішення між середовищами та зменшує витрати на адміністрування. Для прикладу розрахунку приймемо такі значення:

$$C_{\text{міс}} = C_{VPS} + C_{LLM} + C_{search} + C_{\text{рез}} \quad (3.2)$$

де $C_{\text{міс}}$ – щомісячні експлуатаційні витрати;

$C_{VPS} = 300 \frac{\text{грн}}{\text{міс}}$ – витрати на використання мовної моделі;

$C_{LLM} = 800 \frac{\text{грн}}{\text{міс}}$ – витрати на пошуковий API;

$C_{search} = 500 \frac{\text{грн}}{\text{міс}}$ – витрати на пошуковий API;

$C_{\text{рез}=20\,\% \cdot (C_{LLM}+C_{search})}$ – резерв на непередбачені витрати.

Тоді, орієнтовані витрати на дослідну експлуатацію дорівнюють:

$$C_{\text{міс}} = 300 + 800 + 500 + 320 = 1920 \text{ грн/міс.}$$

Така сума є прийнятною для програмного рішення, яке не потребує створення окремих мобільних застосунків, придбання серверного обладнання або утримання великої команди підтримки. Для оцінки економічного ефекту від використання системи доцільно врахувати економію часу користувачів. У традиційному сценарії пасажир змушений вручну переглядати кілька джерел, порівнювати рейси, ціни, час відправлення та умови поїздки. У проєктному рішенні користувач формує один природномовний запит, а система автоматично виконує пошук і структурує результати. Таким чином, середній час відповіді

системи становить 13,6 с, а кількість кроків для отримання результату зменшується до одного запиту. Економію часу можна визначити за формулою:

$$E_t = N \cdot \frac{T_{\text{руч}} - T_{\text{бот}}}{60}, \quad (3.3)$$

де E_t – кількість користувацьких запитів за місяць;

$T_{\text{руч}}$ – середній час ручного пошуку,

$T_{\text{бот}}$ – середній час отримання відповіді через систему, хв.

Для розрахунку приймемо, що ручний пошук маршруту у декількох джерелах у середньому займає 7 хв, а час відповіді системи становить:

$$T_{\text{бот}} = \frac{13{,}6 \text{ с}}{60} = 0{,}227 \text{ хв} \approx 0{,}23 \text{ хв}. \quad (3.4)$$

Якщо протягом місяця система обробляє 1000 запитів, то загальна економія часу становить:

$$E_t = 1000 \cdot \frac{7 - 0{,}23}{60} = 112{,}83 \text{ год}. \quad (3.5)$$

Зекономлений час у грошовому еквіваленті дорівнює:

$$E_{\text{грн}} = E_t \cdot C_{\text{час}}, \quad (3.6)$$

де $E_{\text{грн}}$ – грошовий еквівалент економії часу;

$C_{\text{час}}$ – умовна вартість однієї години часу користувача.

Якщо умовно прийняти вартість однієї години часу користувача на рівні 80 грн, тоді:

$$E_{\text{грн}} = T_{\text{ек}} \cdot C_{\text{год}} = 112{,}83 \cdot 80 = 9026{,}4 \text{ грн/міс}. \quad (3.7)$$

За умови 1000 запитів на місяць орієнтовний соціально-економічний ефект від економії часу користувачів становить близько 9026 грн на місяць. Чистий місячний ефект з урахуванням експлуатаційних витрат дорівнює:

$$E_{\text{чист}} = E_{\text{грн}} - C_{\text{міс}} = 9026{,}4 - 1920 = 7106{,}4 \text{ грн/міс}. \quad (3.8)$$

Для оцінки строку окупності використовується формула:

$$T_{\text{окуп}} = \frac{K_{\text{заг}}}{E_{\text{чист}}}, \quad (3.9)$$

де $T_{\text{окуп}}$ – строк окупності, міс.;

$K_{\text{заг}}$ – загальні початкові витрати;

$E_{\text{чист}}$ – чистий місячний економічний ефект.

Початкові витрати з урахуванням витрат на розроблення, тестування, інфраструктуру та резерв можна оцінити так:

$$K_{\text{заг}} = K_{\text{розр}} + K_{\text{навч}} + K_{\text{орг}} + K_{\text{впров}} =$$
$$= 24000 + 1500 + 500 + 4800 = 30800 \text{ грн}, \quad (3.10)$$

Тоді, строк окупності становить:

$$T_{\text{окуп}} = \frac{30800}{7106{,}4} = 4{,}33 \text{ міс.}$$

Таким чином, за прийнятих умов розрахунку система може окупитися приблизно за 4-5 місяців дослідної або комерційної експлуатації. Це свідчить про економічну доцільність ухвалених технічних рішень. Додатково можна оцінити річний економічний ефект:

$$\text{Е}_{\text{річ}} = 12 \cdot E_{\text{чист}} = 12 \cdot 7106{,}4 = 85276{,}8 \text{ грн.} \quad (3.11)$$

Коефіцієнт рентабельності інвестицій можна визначити за формулою:

$$ROI = \frac{E_{\text{річ}} - k_{\text{заг}}}{K_{\text{заг}}} \cdot 100\ \% = \frac{85276{,}8 - 30800}{30800} \cdot 100\ \% = 176{,}9\ \%. \quad (3.12)$$

За умови стабільного використання системи економічний ефект протягом року може суттєво перевищити початкові витрати на її розроблення. Слід враховувати, що розрахунок має орієнтовний характер, оскільки реальні витрати на API, серверну інфраструктуру та кількість користувацьких запитів можуть змінюватися залежно від тарифів сервісів, навантаження та масштабу впровадження. Важливим економічним аргументом на користь обраної архітектури є використання RAG-підходу замість повного створення власної БД або довчання LLM. При цьому альтернативні підходи мають суттєві обмеження: fine-tuning [43] потребує додаткових обчислювальних ресурсів, класичний вебскрапінг потребує постійного доопрацювання при зміні структури сайтів, а використання «чистої» LLM без пошуку не забезпечує достатньої актуальності даних. RAG-підхід поєднує актуальність зовнішнього пошуку та можливості LLM, зберігаючи прийнятний рівень вартості підтримки [44].

ВИСНОВКИ

У результаті аналізу існуючих онлайн-сервісів для пошуку транспортних рейсів встановлено, що сучасний ринок пасажирських перевезень України є інформаційно фрагментованим. Офіційні сервіси перевізників, автобусні агрегатори та сайти окремих транспортних компаній вирішують лише частину потреб користувача. Жодна з розглянутих систем не забезпечує одночасного пошуку автобусних і залізничних рейсів, обробки запитів природною мовою та надання актуальної деталізованої інформації.

Встановлено, що використання мовної моделі без доступу до актуальних джерел, мають суттєві обмеження. Тому найбільш доцільним є використання RAG спільно з GPT-4o, Tavily API та aiogram 3.x Це дозволяє забезпечити гнучку обробку запитів, пошук у реальному часі та зручну взаємодію користувача через Telegram. Сформульовані основні функціональні та нефункціональні вимоги створюють основу для подальшого проєктування інтелектуальної системи для пошуку транспортних рейсів.

Для побудови моделі зазначеної системи запропонований технологічний стеку, який спирається на 4-рівневу архітектура. Це забезпечує чіткий розподіл функцій між інтерфейсом користувача, бізнес-логікою, інтеграцією з зовнішніми сервісами та інфраструктурним рівнем.

Використання Python 3.11+, aiogram 3.x, OpenAI SDK, Tavily SDK, pydantic та python-dotenv дозволяє створити гнучку, асинхронну та масштабовану систему. Для застосування RAG в процесі обробки транспортних запитів користувачів обґрунтована структура 6-етапного конвеєра роботи системи. В роботі запропонований алгоритми вилучення та структурування даних про транспортні рейси, що реалізує повний цикл обробки запиту.

Програмна реалізація інтелектуальної системи спирається на формуванні структури основних модулів, які забезпечують запуск бота, обробку повідомлень, взаємодію з API та форматування відповідей. Особливу увагу приділено реалізації Telegram-інтерфейсу з Reply-клавіатурою, Inline-кнопками,

прикладами запитів та механізмом уточнення параметрів. Запропоноване рішення забезпечує зручність використання системи, її модульність, надійність і можливість подальшого розвитку.

Для перевірки адекватності розробленої моделі виконане тестування функціональних можливостей та сценаріїв роботи. Отримані результати підтвердили коректну обробку стандартних, неповних, нерелевантних і нестандартних запитів, а також здатність системи працювати в умовах часткових збоїв зовнішніх сервісів. Таким чином, реалізований інтелектуальний чат-бот відповідає основним функціональним і нефункціональним вимогам.

В роботі виконана оцінка ефективності інтелектуального пошуку транспортних рейсів за кількома показниками якості: точність вилучення параметрів маршруту, коректність визначення відносних дат, повнота знайдених залізничних та автобусних рейсів, точність часових і цінових характеристик, середній час відповіді системи, суб’єктивна оцінка якості відповідей користувачами, здатність обробляти паралельні запити та коректність роботи у випадку часткових збоїв зовнішніх сервісів. За прийнятих умов розрахунку система може окупитися за $\approx$ 4-5 місяців дослідної або комерційної експлуатації. Це свідчить про економічну доцільність прийнятих технічних рішень.

Таким чином, результатами роботи є модель інтелектуальної системи для пошуку транспортних рейсів, рекомендації щодо практичної реалізації автоматизації підбору транспортних рейсів на основі AI. Вони можуть бути використані для подальших досліджень за даною тематикою та при проектуванні AI-агентів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. АТ «Укрзалізниця». Офіційний вебсайт АТ «Укрзалізниця». URL: https://www.uz.gov.ua (дата звернення: 01.03.2026).

2. Busfor. Сервіс пошуку та бронювання автобусних квитків. URL: https://busfor.ua (дата звернення: 05.03.2026).

3. INFOBUS. Платформа онлайн-продажу автобусних квитків. URL: https://infobus.eu/ua (дата звернення: 05.03.2026).

4. Lewis P., Perez E., Piktus A. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *arXiv preprint*. arXiv:2005.11401. 2020. 15 p. URL: https://arxiv.org/abs/2005.11401 (дата звернення: 10.03.2026).

5. Зелений слон 7. Офіційний вебсайт автобусного перевізника «Зелений слон 7». URL: https://gs7.com.ua (дата звернення: 07.06.2026).

6. Mallen A., Akyürek A., Basmov V. et al. When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories. *arXiv preprint*. arXiv:2212.10511. 2023. 14 p. URL: https://arxiv.org/abs/2212.10511 (дата звернення: 20.03.2026).

7. Devlin J., Chang M. W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of NAACL-HLT 2019*. Minneapolis, 2019. P. 4171-4186. URL: https://arxiv.org/abs/1810.04805 (дата звернення: 10.02.2026).

8. Kaddour J., Harris J., Mozes M. et al. Challenges and Applications of Large Language Models. *arXiv preprint*. arXiv:2307.10169. 2023. 57 p. URL: https://arxiv.org/abs/2307.10169 (дата звернення: 10.03.2026).

9. Горчакова М. Автоматизація підбору рейсів транспорту за допомогою штучного інтелекту. *Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій: матеріали XXII щорічного міждисциплінарного семінару* (листопад 2025 р., м. Полтава), 2025. С. 39, 40.

10. Горчакова М. Інтелектуальний бот для агрегації транспортних розкладів на основі RAG. *Збірник XXI щорічної студентської наукової*

конференції «Сучасні інформаційні технології та інноваційні методики в економіці, менеджменті та бізнесі» Полтавського державного аграрного університету (квітень 2026 р., м. Полтава). Полтава: ПДАУ, 2026 р. С. 29-31.

11. Vaswani A., Shazeer N., Parmar N. et al. Attention Is All You Need. *Proceedings of NeurIPS 2017*. Long Beach, 2017. P. 5998–6008. URL: https://arxiv.org/abs/1706.03762 (дата звернення: 12.02.2026).

12. Zaratiana U., Tomeh N., Holat P., Charnois T. GLiNER: Generalist Model for Named Entity Recognition using Bidirectional Transformer. *arXiv preprint*. arXiv:2311.08526. 2023. 12 p. URL: https://arxiv.org/abs/2311.08526 (дата звернення: 18.03.2026).

13. Aiogram. Aiogram 3.x Documentation: Modern and Fully Asynchronous Framework for Telegram Bot API. 2024. URL: https://docs.aiogram.dev (дата звернення: 12.02.2026).

14. Tavily. Tavily AI Search API Documentation for LLM Applications. 2024. URL: https://docs.tavily.com (дата звернення: 14.02.2026).

15. Xu P., Zhu X., Clifton D. Multimodal Learning with Transformers: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. Vol. 45, No. 12. P. 14175–14194. URL: https://doi.org/10.1109/TPAMI.2023.3275156 (дата звернення: 25.02.2026).

16. Min B., Ross H., Sulem E. et al. Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey. *ACM Computing Surveys*. 2023. Vol. 56, No. 2. P. 1–40. URL: https://doi.org/10.1145/3605943 (дата звернення: 28.02.2026).

17. McKinney W. Python for Data Analysis: Data Wrangling with pandas, NumPy and Jupyter. 3rd ed. Sebastopol: O'Reilly Media, 2022. 550 p.

18. Brown T., Mann B., Ryder N. Language Models are Few-Shot Learners. *Proceedings of NeurIPS 2020*. Vancouver, 2020. Vol. 33. P. 1877–1901. URL: https://arxiv.org/abs/2005.14165 (дата звернення: 15.02.2026).

19. Liu Y., Han T., Ma S. et al. Summary of ChatGPT/GPT-4 Research and Perspective Towards the Future of Large Language Models. *arXiv preprint*.

arXiv:2304.01852. 2023. 34 p. URL: https://arxiv.org/abs/2304.01852 (дата звернення: 22.02.2026).

20. Zhao W., Zhou K., Li J. et al. A Survey of Large Language Models. *arXiv preprint*. arXiv:2303.18223. 2023. 114 p. URL: https://arxiv.org/abs/2303.18223 (дата звернення: 08.03.2026).

21. Gao Y., Xiong Y., Gao X. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv preprint*. arXiv:2312.10997. 2024. 45 p. URL: https://arxiv.org/abs/2312.10997 (дата звернення: 15.03.2026).

22. Haydari A., Yilmaz Y. Deep Reinforcement Learning for Intelligent Transportation Systems: A Survey. *IEEE Transactions on Intelligent Transportation Systems*. 2022. Vol. 23, No. 1. P. 11–32. URL: https://doi.org/10.1109/TITS.2020.3014595 (дата звернення: 02.04.2026).

23. Docker. Docker Documentation: Containerization Platform. 2024. URL: https://docs.docker.com (дата звернення: 20.02.2026).

24. OpenAI. GPT-4 Technical Report. *arXiv preprint*. arXiv:2303.08774. 2023. 98 p. URL: https://arxiv.org/abs/2303.08774 (дата звернення: 20.02.2026).

25. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow. 3rd ed. Sebastopol: O'Reilly Media, 2023. 868 p.

26. Ouyang L., Wu J., Jiang X. et al. Training language models to follow instructions with human feedback. *Proceedings of NeurIPS 2022*. New Orleans, 2022. P. 27730–27744. URL: https://arxiv.org/abs/2203.02155 (дата звернення: 15.03.2026).

27. Шаховська Н., Голощук Р. Алгоритми і структури даних: посіб. Львів: Магнолія, 2020. 215 с.

28. OpenAI. OpenAI API Documentation. 2024. URL: https://platform.openai.com/docs (дата звернення: 10.02.2026).

29. Rothman D. Transformers for Natural Language Processing: Build, train, and fine-tune deep neural network architectures for NLP. 2nd ed. Birmingham: Packt Publishing, 2022. 590 p.

30. Raschka S., Liu Y., Mirjalili V. Machine Learning with PyTorch and Scikit-Learn. Birmingham: Packt Publishing, 2022. 740 p.

31. Russel S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. Hoboken: Pearson, 2022. 1132 p.

32. Wei J., Wang X., Schuurmans D. et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Proceedings of NeurIPS 2022*. New Orleans, 2022. P. 24824–24837. URL: https://arxiv.org/abs/2201.11903 (дата звернення: 05.03.2026).

33. Pan S. J., Yang Q. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*. 2010. Vol. 22, No. 10. P. 1345–1359. URL: https://doi.org/10.1109/TKDE.2009.191 (дата звернення: 01.04.2026).

34. Fowler M. Python Concurrency with asyncio. *Shelter Island: Manning Publications*, 2022. 424 p.

35. Shinn N., Cassano F., Berman E. et al. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv preprint*. arXiv:2303.11366. 2023. 25 p. URL: https://arxiv.org/abs/2303.11366 (дата звернення: 22.03.2026).

36. Yao S., Zhao J., Yu D. et al. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint*. arXiv:2210.03629. 2023. 26 p. URL: https://arxiv.org/abs/2210.03629 (дата звернення: 18.03.2026).

37. Telegram. Telegram Bot API Documentation. 2024. URL: https://core.telegram.org/bots/api (дата звернення: 08.02.2026).

38. Zhang J., Zheng Y., Qi D. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. *Proceedings of AAAI 2017*. San Francisco, 2017. P. 1655–1661. URL: https://doi.org/10.1609/aaai.v31i1.10735 (дата звернення: 03.04.2026).

39. Pydantic. Pydantic v2 Documentation: Data Validation for Python. 2024. URL: https://docs.pydantic.dev (дата звернення: 18.02.2026).

40. Hattingh C. Using Asyncio in Python: Understanding Python's Asynchronous Programming Features. Sebastopol: O'Reilly Media, 2020. 168 p.

41. Karpukhin V., Oguz B., Min S. et al. Dense Passage Retrieval for Open-Domain Question Answering. *Proceedings of EMNLP 2020*. 2020. P. 6769–6781. URL: https://arxiv.org/abs/2004.04906 (дата звернення: 25.03.2026).

42. Zhu Y., Yuan H., Wang S. et al. Large Language Models for Information Retrieval: A Survey. *arXiv preprint*. arXiv:2308.07107. 2023. 39 p. URL: https://arxiv.org/abs/2308.07107 (дата звернення: 28.03.2026).

43. Bubeck S., Chandrasekaran V., Eldan R. et al. Sparks of Artificial General Intelligence: Early experiments with GPT-4. *arXiv preprint*. arXiv:2303.12528. 2023. 155 p. URL: https://arxiv.org/abs/2303.12528 (дата звернення: 10.03.2026).

44. Gosling J., Joy B., Steele G., Bracha G., Buckley A. The Java Language Specification. Java SE 21 Edition. Redwood City. : Oracle America, 2023. 800 p. URL: https://docs.oracle.com/javase/specs/jls/se21/html/index.html (дата звернення: 06.06.2026).