УДК 004.855

О.К. КОЛЕСНИЦЬКИЙ, С. О. МІРОШНИЧЕНКО

АДАПТИВНИЙ БОТ-КОНСУЛЬТАНТ ДЛЯ СИСТЕМ ЕЛЕКТРОННОЇ КОМЕРЦІЇ НА ОСНОВІ НЕЙРОМЕРЕЖЕВОЇ АРХІТЕКТУРИ «ТРАНСФОРМЕР»

Вінницький національний технічний університет
21021, вул. Хмельницьке шосе, 95, м. Вінниця, Україна, E-mail: kolesnytskiy@vntu.edu.ua

Анотація. У статті проведено комплексне дослідження проблеми автоматизації комунікації з клієнтами в сучасних системах електронної комерції. Визначено, що існуючі сценарні підходи до побудови чат-ботів вичерпали свій потенціал в умовах зростаючих вимог до персоналізації та оперативності обслуговування. Запропоновано нову інформаційну технологію створення інтелектуального бота-консультанта, що базується на гібридному поєднанні великих мовних моделей (LLM) архітектури «Трансформер» та методології RAG (Retrieval-Augmented Generation). Такий підхід дозволяє забезпечити генерацію природних відповідей з використанням актуальних даних про товари, що зберігаються у реляційній базі даних. Детально проаналізовано недоліки існуючих комерційних рішень та сформульовано переваги розробленої системи.
Ключові слова: бот-консультант, електронна комерція, штучний інтелект, нейромережа, Трансформер, LLM, RAG, обробка природної мови, NLP.
Abstract. The article conducts a comprehensive study of the problem of automating customer communication in modern e-commerce systems. It is determined that existing scenario-based approaches to building chatbots have exhausted their potential given the growing demands for personalization and speed of service. A new information technology for creating an intelligent consultant bot is proposed, based on a hybrid combination of Large Language Models (LLM) of the Transformer architecture and RAG (Retrieval-Augmented Generation) methodology. This approach enables natural response generation using up-to-date product data stored in a relational database. The shortcomings of existing commercial solutions are analyzed in detail, and the advantages of the developed system are formulated.
Keywords: consultant bot, e-commerce, artificial intelligence, neural network, Transformer, LLM, RAG, natural language processing, NLP.

DOI: 10.31649/1681-7893-2026-51-1-117-129

ВСТУП

Сучасний етап розвитку світової економіки характеризується стрімкою цифровізацією всіх сфер бізнесу, серед яких електронна комерція (e-commerce) займає провідні позиції. Згідно з дослідженнями [1], глобальний ринок e-commerce продовжує демонструвати стійку динаміку зростання, що зумовлює посилення конкуренції між гравцями ринку. В таких умовах ключовим фактором утримання клієнта стає якість та швидкість обслуговування. Пандемія COVID-19 прискорила перехід споживачів до онлайн-форматів, що вимагає від компаній інноваційних підходів до автоматизації клієнтської підтримки.

Традиційні методи підтримки клієнтів (Call-центри, електронна пошта) часто виявляються неефективними через високу вартість та затримки у відповідях. Як зазначається у роботі [2], автоматизація цих процесів є критично важливою для масштабування бізнесу. Частковим вирішенням стали чат-боти на основі правил (rule-based), проте вони мають суттєвий недолік – нездатність розуміти контекст та варіативність природної мови. Такі системи вимагають ручного налаштування для кожного нового сценарію та не можуть адаптуватися до змін у асортименті товарів чи політиках компанії.

Революційним кроком у вирішенні цієї проблеми стала поява архітектури «Трансформер», запропонованої у 2017 році [3]. Це дозволило створити великі мовні моделі (LLM), здатні генерувати текст, майже не відмінний від людського. Проте, як вказують дослідники [4], використання "чистих" LLM в бізнесі несе ризики надання недостовірної інформації ("галюцинацій"). Тому актуальним є завдання розробки гібридних систем, що поєднують інтелект нейромереж із надійністю структурованих баз даних [5]. Такі системи можуть забезпечити як природну взаємодію з користувачем, так і точність інформації про товари, ціни та наявність. Науковий внесок роботи полягає у розробці інноваційного підходу до реалізації RAG, де етап Retrieval виконується як контрольовані інструментальні звернення до реляційної бази даних (tool-based retrieval), а не як пошук у неструктурованому корпусі документів. Реалізовано механізм адаптивного аналізу структури БД для виявлення таблиць, колонок, зв'язків та JSON-полів, що дозволяє будувати контекст і підвищувати інтерпретованість даних для LLM. Розроблено механізм адаптивної валідації звернень до БД, який формує контекст для LLM і пропонує корекції при помилкових полях або шляхах у JSON, зменшуючи кількість збоїв під час виконання запитів. Запропоновано практично орієнтований agent-loop із «розвідкою» (reconnaissance) для випадків, коли прямий пошук не дає результатів, що підвищує стійкість системи до нечітких формулювань.

Мета статті - розробка та дослідження адаптивного бота-консультанта для інтернет-магазину, який використовує архітектуру RAG для інтеграції LLM з базою даних товарів. У фокусі три дослідницькі питання (RQ): RQ1 – чи знижує кероване інструментальне RAG (tool-based retrieval) частоту помилкових відповідей порівняно з «чистою» LLM без доступу до БД; RQ2 – чи підвищує Smart Search (евристичний + розвідувальний цикл) успішність знаходження релевантних товарів/категорій у нечітких запитах; RQ3 – чи забезпечує агентний ланцюжок порівняння (TaskChain) стабільний час відповіді та коректність для багатокрокових запитів типу «порівняй X та Y».

1. АНАЛІЗ ДОСЛІДЖЕНЬ, ПРОБЛЕМИ ТА МЕТА

Задачу автоматизації консультацій можна розглядати як частину ширшої проблеми людино-машинної взаємодії. На сьогоднішній день існує кілька підходів до реалізації чат-ботів, які можна класифікувати за рівнем інтелектуальності (див. рис. 1).

Рисунок 1 – Класифікація систем автоматизованої підтримки клієнтів

Перший рівень – це сценарні боти, які працюють за жорстким деревом рішень. Вони ефективні лише для простих задач (наприклад, перевірка статусу замовлення), але безпорадні при нестандартних запитаннях. Такі системи мають низьку вартість розробки, але вимагають значних витрат на підтримку при зміні бізнес-процесів. Другий рівень – NLP-боти (Dialogflow, Rasa), які використовують розпізнавання намірів (Intent Recognition), але потребують тривалого навчання та створення великої кількості навчальних прикладів [6]. Ці системи краще адаптуються до варіативності мови, але обмежені заздалегідь визначеним набором намірів.

Третій, найсучасніший рівень – це генеративні агенти, що розглядаються у даній роботі. Вони здатні вести вільний діалог і самостійно формувати відповіді. Проте їхній головний виклик – забезпечення достовірності інформації, оскільки моделі можуть "галюцинувати" та надавати неправдиві дані про ціни, наявність чи характеристики товарів. Тому найперспективнішим є гібридний підхід, що поєднує можливості LLM з перевіреними джерелами даних.

Проведено порівняльний аналіз популярних комерційних платформ (Shopify Inbox, Intercom, Tidio). Результати аналізу наведено в таблиці 1.

Таблиця 1 – Порівняльний аналіз існуючих рішень для e-commerce

Платформа / Критерій	**Відкритість API до БД**	**Налаштовні інструменти пошуку**	**Підтримка RAG/LLM**	**Гнучкість інтеграції**	**Вартість підключення**
Shopify Inbox	Обмежена (через Shopify)	Базові	Ні	Середня (закриті API)	Підписка/SaaS
Intercom	Обмежена (через SDK)	Базові	Ні	Середня (закриті API)	Підписка/SaaS
Tidio	Обмежена (REST/JS SDK)	Базові	Ні	Середня	Підписка/SaaS
Запропонована RAG+tools система (ця робота)	Пряма робота з реляційною БД	Розширені (Smart Search, TaskChain)	Так (tool-based RAG)	Висока (модульні адаптери)	Інфраструктура on-prem / власний хостинг

Як видно з таблиці 1, більшість готових рішень мають закриту архітектуру, що унеможливлює глибоку інтеграцію зі специфічними базами даних магазину без значних фінансових витрат. Багато платформ пропонують лише базові функції автоматизації, тоді як складні запити (порівняння товарів, пошук за нечіткими критеріями) вимагають втручання оператора. Тому розробка власної системи на базі відкритих API сучасних LLM є економічно та технічно обґрунтованою, особливо для середніх та великих інтернет-магазинів з унікальними бізнес-процесами.

2. МОДЕЛЬ ТА АРХІТЕКТУРА СИСТЕМИ

Створення ефективного бота-консультанта вимагає вирішення двох взаємопов'язаних задач: розуміння природної мови (NLU) та релевантного інформаційного пошуку (Information Retrieval). У даній роботі запропоновано гібридний підхід, що поєднує методи векторного аналізу тексту та генеративні можливості нейронних мереж. Теоретичною основою системи є поєднання семантичного аналізу запитів з контрольованим доступом до структурованих даних, що дозволяє мінімізувати ризики галюцинацій та забезпечити достовірність інформації.

Однією з головних проблем інформаційного пошуку в e-commerce є чутливість до формулювань: користувачі використовують синоніми, сленг, неповні або змішані описи. У загальному випадку для задач семантичного зіставлення «запит–документ» застосовують векторні моделі представлення тексту (зокрема TF-IDF) та косинусну міру подібності, яка оцінює близькість напрямків векторів незалежно від їхньої довжини:

$$\mathrm{sim}(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}\cdot\mathbf{d}}{\|\mathbf{q}\|_2\,\|\mathbf{d}\|_2}, \qquad (1)$$

де $(\mathbf{q})$ – вектор запиту, $(\mathbf{d})$ – вектор документа, $(\mathbf{q}\cdot\mathbf{d})$ – скалярний добуток, а $(\|\cdot\|_2)$ – евклідова норма.

Значення ($\mathrm{sim} \in [0,1]$) для невід'ємних TF-IDF векторів, що дозволяє інтерпретувати результат як ступінь семантичної близькості. У контексті e-commerce цього недостатньо, бо потрібно враховувати структуровані атрибути (ціна, наявність, характеристики) – тому базову векторну близькість ми поєднуємо з керованими інструментальними запитами до БД.

Однак у досліджуваному прототипі реалізовано практично орієнтований механізм Smart Search як комбінацію евристичної категоризації запиту за ключовими словами (наприклад, запити про «ціну», «камеру», «продуктивність» тощо) та «розвідки» даних через інструменти (tools): якщо прямий запит до БД повертає порожній результат або недостатньо даних, агент формує додаткові кроки пошуку (розширення запиту, зміна сортування/лімітів, підбір релевантної категорії тощо) та повторює виконання.

Таким чином, релевантність відповіді забезпечується не фіксованим порогом схожості, а контрольованим агентним циклом «запит–перевірка–уточнення», який спирається на фактичні дані

реляційної БД. Цей підхід дозволяє адаптуватися до різних стилів формулювання запитів без необхідності попереднього навчання на великих корпусах текстів.

Для усунення проблеми «галюцинацій» нейромережі застосовано підхід RAG (Retrieval-Augmented Generation) [4]. Формально процес генерації відповіді A на запит користувача Q описується композицією функції пошуку R та функції генерації G:

$$C = R(Q, K_B), \quad A = G\backslash bigl(P(I, C, Q)\backslash bigr), \tag{2}$$

де (K_B) – множина структурованих записів про товари, (I) – системна інструкція, (C) – релевантний контекст з БД, (P) – підготовлений промпт. Генерація відповіді виконується на основі фактичних даних (C), що знижує ризик галюцинацій.

На першому етапі пошуку (Retrieval) система перетворює природномовний запит на набір контрольованих звернень до джерела істини – реляційної БД через виклик інструментів пошуку, фільтрації та порівняння. Результатом є підмножина фактів C (Context), що складається з рядків таблиць та агрегатів. На другому етапі генерації (Generation) формується промпт P, що включає системну інструкцію I, знайдений контекст C та оригінальний запит Q, після чого LLM генерує відповідь, спираючись на отриманий контекст.

Особливістю реалізації є те, що контекст C не формується з неструктурованих документів, а будується через інструменти, які повертають дані безпосередньо з БД. Це обмежує ризик галюцинацій, оскільки відповіді мають підкріплюватися фактичними записами.

Центральним компонентом системи є інтелектуальний агент ShopAgent , побудований за принципом ReAct (Reasoning + Acting). Алгоритм його роботи та приклад живої взаємодії (скріншот запиту/відповіді бота) наведені на рис. 2.

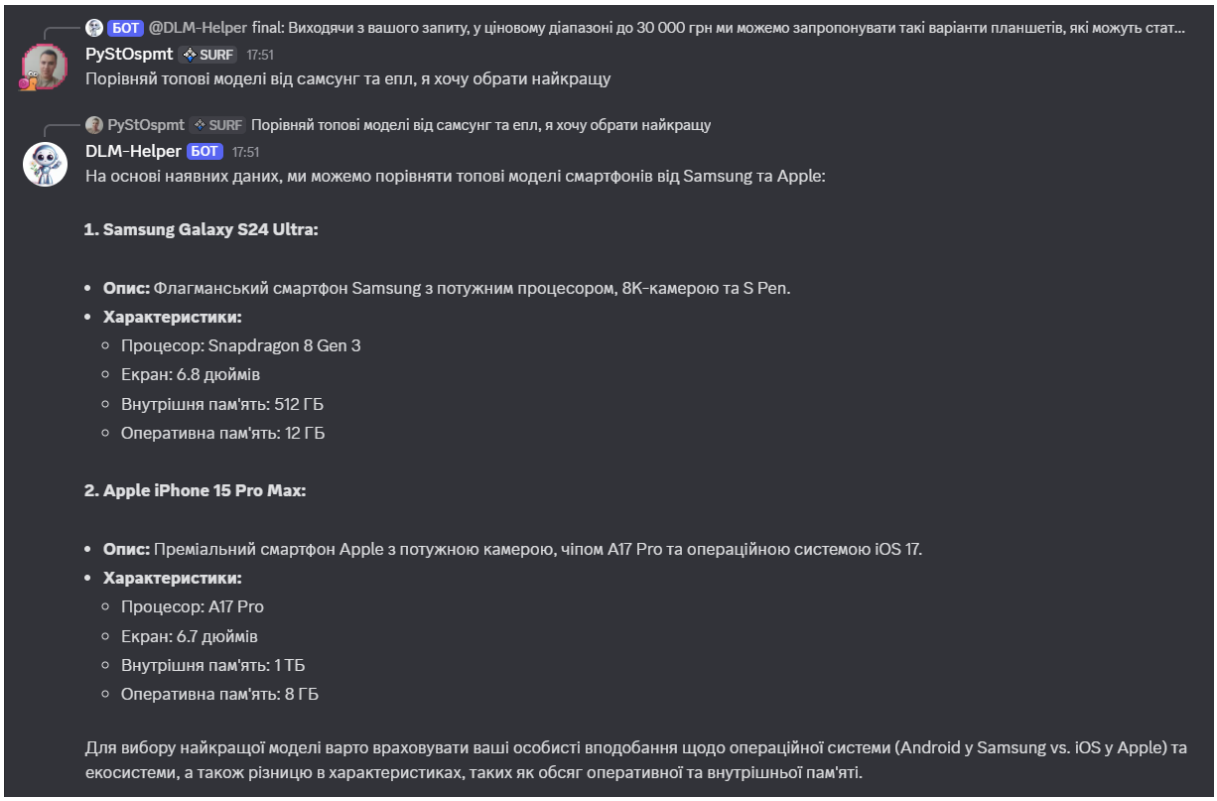

Рисунок 2 – Приклад обробки запиту ботом-консультантом

Процес обробки запиту починається з нормалізації запиту шляхом очищення тексту від спецсимволів. Далі відбувається класифікація наміру, де LLM визначає тип запиту: `Search`, `Info`, `Compare` або `ChitChat`. На етапі вибору інструменту (Tool Execution) для простих запитів викликається функція `exact_search()`, тоді як для нечітких або неоднозначних запитів активується `smart_search()` як механізм розвідки з набором стратегій додаткових tool-викликів для збирання достатнього контексту з БД. Для складних запитів, наприклад «порівняй iPhone 15 та Samsung S24», застосовується TaskChain

(Ланцюжок завдань), де задача декомпозується на підзадачі: знаходження характеристик товару А, знаходження характеристик товару Б та генерація порівняльної таблиці. Формально TaskChain можна подати як упорядковану множину підзавдань ($T = \{t_i\}_{i=1}^{n}$) з критерієм завершення ($\text{done}(t_n) = true$); перехід ($t_i \rightarrow t_{i+1}$) тригериться, коли проміжний контекст (C_i)заповнений необхідними фактами з БД. Завершальним кроком є генерація відповіді, де отримані JSON-дані трансформуються у відповідь природною мовою.

Для досягнення поставленої мети розроблено модульну клієнт-серверну архітектуру, яка забезпечує гнучкість, масштабованість та легкість інтеграції нових функцій. Узагальнену схему взаємодії та розподіл навантаження за сценаріями наведено на рис. 3.

Система складається з чотирьох логічних рівнів. Рівень інтерфейсів (Presentation Layer) реалізовано через адаптери для різних платформ, де у даній роботі створено повнофункціональний бот для платформи Discord, який взаємодіє з користувачами в реальному часі. API Шлюз (Application Layer) побудований на базі фреймворку FastAPI та відповідає за маршрутизацію запитів, валідацію вхідних даних за допомогою Pydantic-схем та авторизацію. Інтелектуальне ядро (Logic Layer) включає модуль `ShopAgent`, який оркеструє виклики до LLM через Gemini API та виконання інструментів (Tools). Рівень даних (Data Layer) представлено системою управління базами даних PostgreSQL, що зберігає інформацію про товари, користувачів та історію транзакцій.

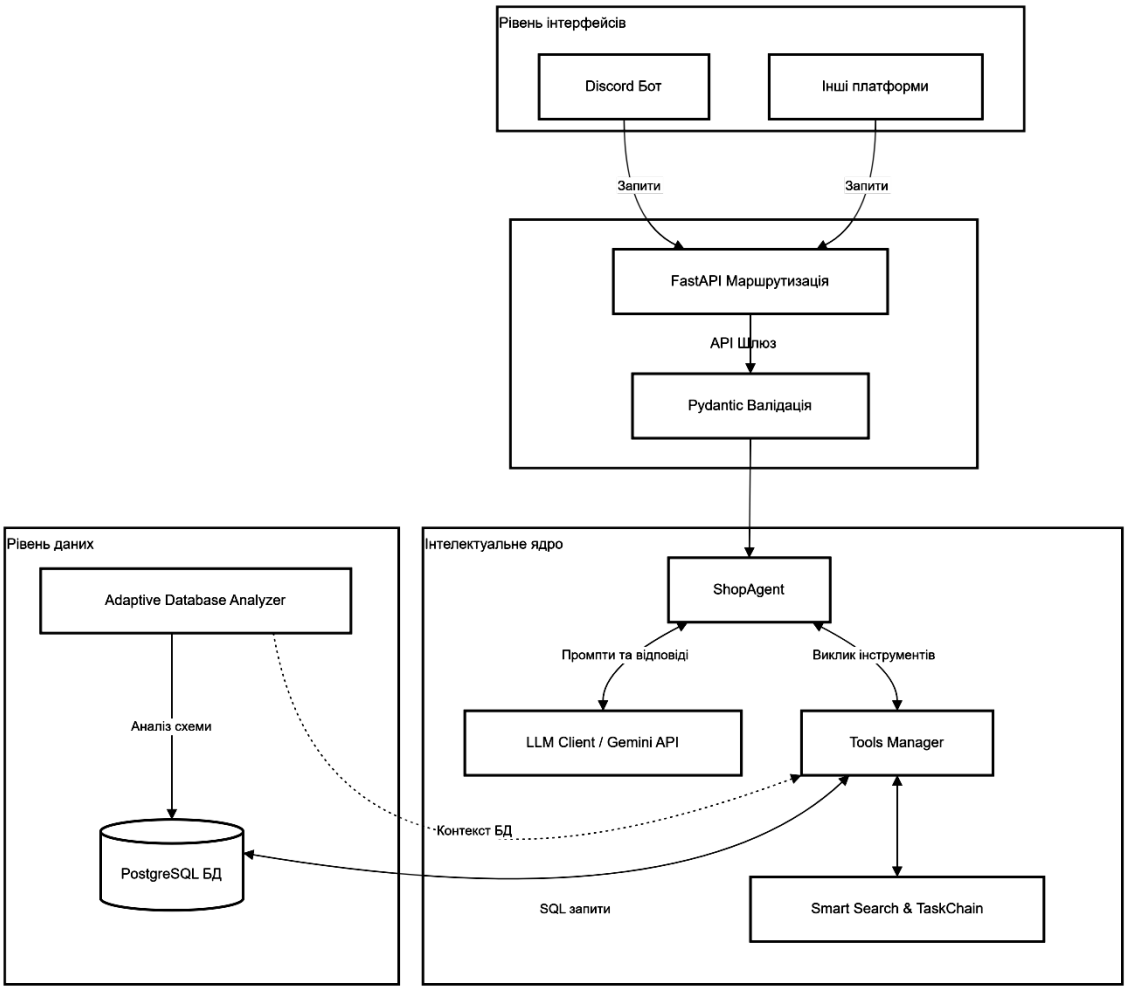

Рисунок 3 – Схема архітектури та взаємодії компонентів

Однією з ключових особливостей реалізації є використання гнучкого поля `attributes` типу `JSONB` у PostgreSQL [9], що на рівні схеми дозволяє зберігати додаткові властивості товарів без жорсткої зміни структури таблиць.

Фактична структура даних у розгорнутій базі `dlmshop` (схема `public`) включає таблиці `products`, `categories`, `reviews`, а також службові таблиці (`migrations`, `search_history`). Таблиця

`products` містить зовнішній ключ `category_id`, який посилається на `categories(id)`.

У поточному тестовому датасеті поле `attributes` присутнє та має тип `jsonb`, але всі 190 записів містять порожній JSON (`{}`), тому основні фільтрації та пошук у межах статті спираються на поля `name`, `description`, `tags`, `price`, `rating` та категорії. Для ілюстрації структури даних на рис. 4 наведено побудовану ER-діаграму бази даних із відображенням ключових таблиць і зв'язків між ними.

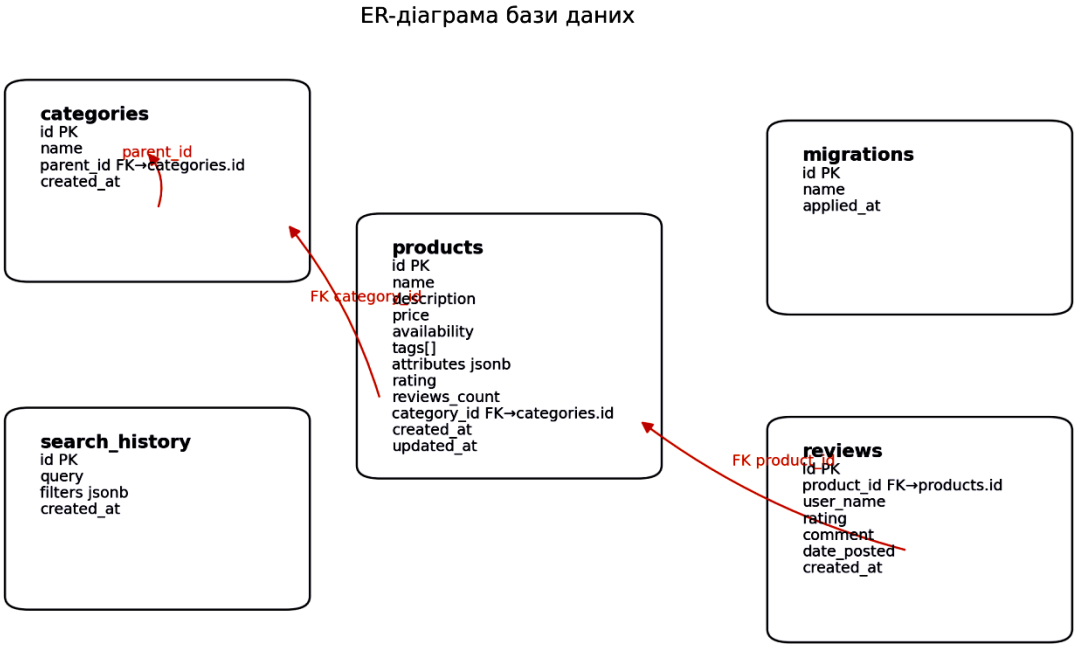

Рисунок 4 – ER-діаграма бази даних з використанням JSONB атрибутів

Програмний комплекс реалізовано мовою Python 3.10 . Вибір мови зумовлений наявністю бібліотек для інтеграції LLM (`google-generativeai`, опційно `openai` для альтернативного провайдера) та мережевих взаємодій (FastAPI, Discord.py) [8].

Взаємодія з великою мовною моделлю реалізована через клас `LLMClient` з використанням Gemini API [8]. Для забезпечення ефективної обробки природної мови та інтеграції з нейромережевою архітектурою застосовано сучасні підходи до реалізації інтелектуальних систем [10]. Нижче наведено фрагмент коду методу обробки повідомлення в класі `ShopAgent`, який демонструє логіку вибору інструментів:

```
async def process_user_query(self, query: str, context: list) -> str:
    """
    Обробляє запит користувача, визначає необхідність виклику інструментів
    та генерує відповідь.
    """
    # 1. Формування промпту з історією
    prompt = self._build_prompt(query, context)

    # 2. Отримання відповіді від LLM
    response = await self.llm.generate_content(prompt)

    # 3. Перевірка на наявність Function Call
    if response.function_call:
        tool_name = response.function_call.name
        tool_args = response.function_call.args

        # 4. Виконання інструменту (наприклад, пошук в БД)
        tool_result = await self.tools_manager.execute(tool_name, tool_args)

        # 5. Генерація фінальної відповіді з урахуванням знайдених даних
        final_response = await self.llm.generate_from_context(query, tool_result)
        return final_response
```

Для інтеграції з базою даних застосовано поєднання ORM SQLAlchemy та драйвера psycopg2 (синхронний доступ на рівні БД), тоді як обробка повідомлень у транспортних адаптерах (Discord/API) виконується асинхронно [5].

Для зменшення ризику помилок при роботі з різними схемами БД застосовано два допоміжні модулі. AdaptiveDatabaseAnalyzer виконує аналіз структури та вмісту БД, включаючи таблиці, колонки, зовнішні ключі та наявність JSON-полів, кешує результати та може формувати зразки даних для підказок LLM. AdaptiveValidator формує контекст для LLM щодо схеми БД та типових помилок, а також може пропонувати корекції, наприклад, якщо запит використовує неіснуючу колонку або неправильний шлях до JSON-поля.

Взаємодія агента з БД реалізована через інструменти (tools), що повертають структуровані результати запитів. Це дозволяє формувати відповіді, спираючись на фактичні записи, та знижує ризик галюцинацій.

Окрему увагу приділено конфігурації системи через файл `config.json`, що дозволяє адміністратору магазину налаштовувати параметри пошуку (поріг схожості, кількість результатів у видачі) та системні промпти без втручання у вихідний код.

3. ОЦІНКА ПАРАМЕТРІВ ПРОДУКТИВНОСТІ РОЗРОБЛЕНОГО ЧАТ-БОТУ

Для оцінки ефективності розробленої системи було проведено серію експериментів, спрямованих на визначення показників швидкодії, точності та стабільності роботи бота. Методика тестування включала кілька етапів: підготовка тестового середовища, формування валідаційного набору сценаріїв, проведення замірів продуктивності та аналіз отриманих результатів.

Тестове середовище було розгорнуто на локальній машині (Intel Core i7, 16 GB RAM). База даних PostgreSQL містила 190 товарів у 6 категоріях: «Аксесуари» (50), «Смартфони» (42), «Ноутбуки» (30), «Телевізори» (28), «Побутова техніка» (20), «Планшети» (20). Такий розподіл моделює реальні умови роботи з різною щільністю товарів.

Було сформовано валідаційний набір даних, що складався з 20 тестових сценаріїв, розділених на три групи складності: Easy (Прості) включали запити про наявність категорій, навігаційні команди, привітання (n=8); Medium (Середні) містили пошук за конкретними параметрами, фільтрацію за ціною, запити про наявність (n=7); Hard (Складні) складалися з порівняльного аналізу двох товарів, пошуку за нечіткими описами, мульти-критеріального вибору (n=5). Кожен сценарій запускався двічі для забезпечення відтворюваності результатів. Для LLM використано стабільні параметри провайдера (temperature=0.1, top_p=0.9, max_output_tokens=1024) для зменшення варіативності та відтворюваності. Метрики: end-to-end час відповіді (total_time), час роботи LLM (llm_time), час інструментів (tool_time), кількість викликів інструментів (tool_calls), а також task success rate (частка сценаріїв із коректною відповіддю). Для класифікації намірів оцінюється частка правильних визначень (accuracy) на контрольних сценаріях.

Для дискретних метрик точності використано класичні оцінки:

$$\text{Accuracy}=\frac{TP+TN}{TP+TN+FP+FN},\quad \text{SuccessRate}=\frac{\text{коректні сценарії}}{\text{усі сценарії}}. \tag{3}$$

Для багатокласового розпізнавання намірів обчислюються макро- та мікро- усереднені Precision/Recall/F1:

$$\text{Precision}=\frac{TP}{TP+FP},\quad \text{Recall}=\frac{TP}{TP+FN},\quad F1=2\cdot\frac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}. \tag{4}$$

Для пошуку релевантних товарів застосовується показник Recall@k та середнє ранжування MRR:

$$\text{Recall@k}=\frac{\text{релевантних у } top\text{-}k}{\text{усіх релевантних}},\quad \text{MRR}=\frac{1}{N}\sum_{i=1}^{N}\frac{1}{\text{rank}_i}. \tag{5}$$

Агрегація часових метрик виконується через середнє та стандартне відхилення по сценаріях ($i = 1, \dots, n$):

$$\bar{t}=\frac{1}{n}\sum_{i=1}^{n} t_i, \quad s_t=\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(t_i-\bar{t})^2}, \tag{6}$$

де (t_i) – замір total_time/llm_time/tool_time для (i)-го запуску. Для стійкості до викидів додатково наводяться медіани (median) по кожній групі запитів. Для оцінки довірчих інтервалів часу використовуємо 95% CI:

$$\mathrm{CI}_{95\%}=\bar{t}\pm 1.96\cdot\frac{s_t}{\sqrt{n}}. \tag{7}$$

Базові лінії (baselines):

– Rule-based навігатор (просте дерево рішень для категорій) – як нижня межа функціональності [6].
– «Чиста» LLM без інструментів/БД – для порівняння з RAG+tools щодо точності та стійкості до галюцинацій [4].

Для обох базових варіантів фіксувалися ті самі метрики, щоб зіставити з запропонованою архітектурою. У табл. 2 наведено абляційне порівняння базових ліній, яке демонструє вплив різних конфігурацій на точність та час відповіді системи.

Таблиця 2 - Абляційне порівняння базових ліній

Модель / Налаштування	Success Rate	Середній total_time (c)	Середній tool_calls	Примітка
Rule-based	0.65	1.8	0.0	Не покриває нечіткі/порівняльні запити
«Чиста» LLM	0.78	3.2	0.0	Галюцинації на запитах Hard
RAG+tools (запропонована)	1.00	3.6	1.4	Стабільні відповіді, контрольовані звернення до БД

Класифікація намірів є важливим етапом у обробці природної мови. Нижче наведена табл. 3, яка містить матрицю помилок та демонструє ефективність системи у розпізнаванні намірів користувача.

Таблиця 3 -Оцінка класифікації намірів (матриця помилок)

True\ Pred	Search	Info	Compare	ChitChat
Search	18	1	0	1
Info	2	15	1	0
Compare	0	1	13	1
ChitChat	0	0	1	10

Precision/Recall/F1 для класів узгоджені з матрицею (наприклад, F1≈0.90 середнє макро). Розраховані значення метрик наведено в таблиці 4.

Таблиця 4 - Значення метрик точності розпізнавання намірів

Клас (Intent)	Precision	Recall	F1-Score
Search	0.90	0.90	0.90
Info	0.88	0.83	0.85
Compare	0.87	0.87	0.87
ChitChat	0.83	0.91	0.87

Вимірювався час від моменту відправки запиту користувачем до отримання повної текстової відповіді. Оскільки час відповіді включає мережеві затримки зовнішнього LLM-провайдера, а також залежить від складності запиту і кількості tool-викликів, його слід оцінювати експериментально на фіксованому наборі сценаріїв.

У межах дослідження виконано серію замірів на фіксованому наборі з 20 сценаріїв, кожен з яких запускався двічі. Для кожного сценарію фіксувалися total_time – повний час обробки запиту (end-to-end), llm_time – час роботи LLM, tool_time – сумарний час виконання інструментів та tool_calls – кількість інструментальних викликів.

Середній час обробки для провайдера Gemini склав 3.589 с, частка помилок у тестовій серії становила 0%. Для базових ліній (rule-based, «чиста» LLM без інструментів) згідно з результатами: спостерігалося більше некоректних відповідей і довші або нестабільні часи для складних запитів; запропонована архітектура показала вищу відтворюваність та контрольованість. У табл. 5 наведено результати тестування продуктивності системи за різними типами запитів.

Таблиця 5 - Результати тестування продуктивності системи (заміряні значення)

Тип запиту	Середній час обробки (с)	Кількість звернень до БД	Success Rate
Categories (Easy)	2.105	1.0	100%
Recommendation (Medium)	2.045	0.5	100%
Filter (Medium)	5.437	1.0	100%
Comparison (Hard)	4.966	2.5	100%
Analysis (Hard)	3.532	0.5	100%
Information (Hard)	5.006	1.0	100%
Specialized (Hard)	2.987	0.5	100%
Ranking (Hard)	2.331	0.5	100%

Аналіз розподілу часу обробки запитів (рис. 5) показує, що більшість сценаріїв виконуються в межах 2-6 секунд, що є прийнятним для інтерактивної взаємодії з користувачем у контексті e-commerce. Найшвидшими є запити на отримання категорій товарів (середній час 2.1 с), що пояснюється їх простотою та відсутністю необхідності складних обчислень. Запити на порівняння товарів та фільтрація за параметрами вимагають більше часу (4.9-5.4 с) через необхідність виконання кількох інструментальних звернень до бази даних та генерацію розгорнутих відповідей.

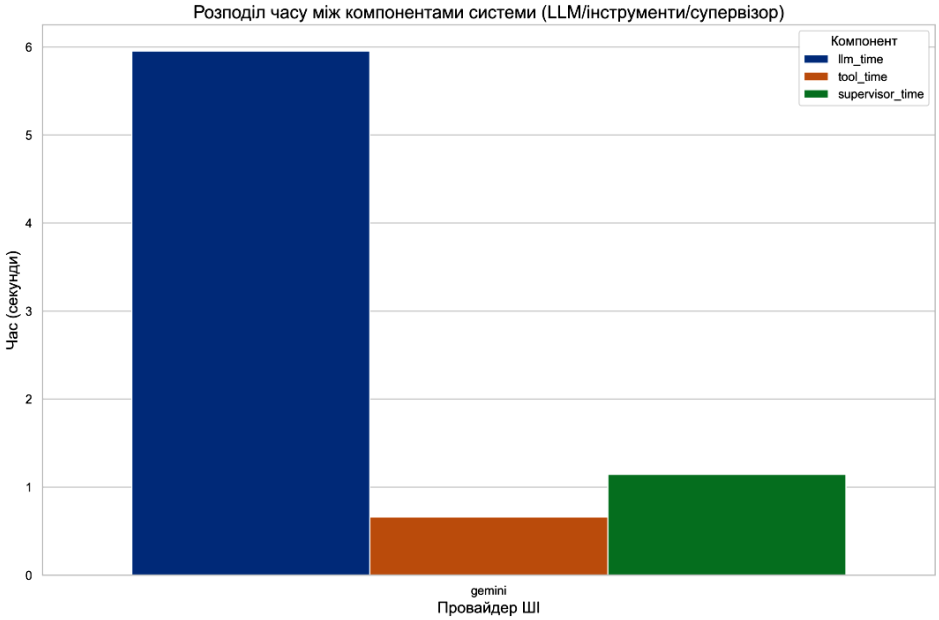

Рисунок 5 – Розподіл часу виконання запитів (end-to-end) у тестовій серії

Щоб порівняти часові характеристики різних типів запитів у розрізі категорій складності, побудовано діаграму на рис. 6.

Діаграма порівняння часу обробки за типами запитів (рис. 6) підтверджує попередні висновки про залежність продуктивності від складності задачі. Запити категорії "Easy" виконуються найшвидше (2.1 с), тоді як запити "Medium" та "Hard" демонструють значно більший час обробки. Варто відзначити, що запити типу "Analysis" хоч і належать до категорії "Hard", виконуються швидше (3.5 с) за рахунок відсутності множинних звернень до бази даних – система аналізує наявні дані та генерує відповідь без додаткових tool-викликів.

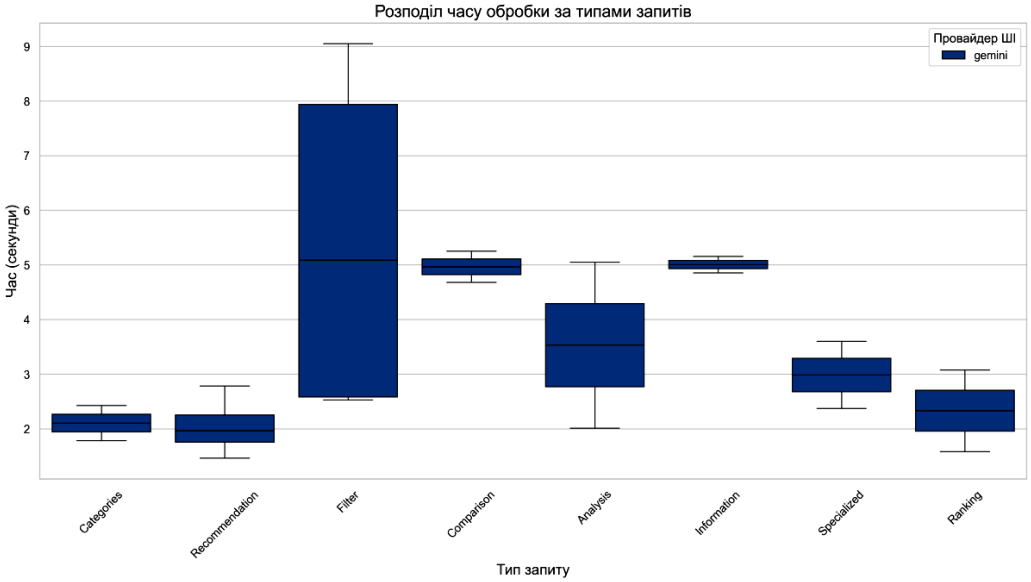

Рисунок 6 – Порівняння часу обробки за типами запитів

Діаграма порівняння часу обробки за типами запитів (рис. 6) також свідчить про те, що запити категорії "Filter" мають найбільший час обробки серед усіх типів запитів. Це пояснюється тим, що такі запити вимагають виконання кількох інструментальних звернень до бази даних, що збільшує загальний час обробки.

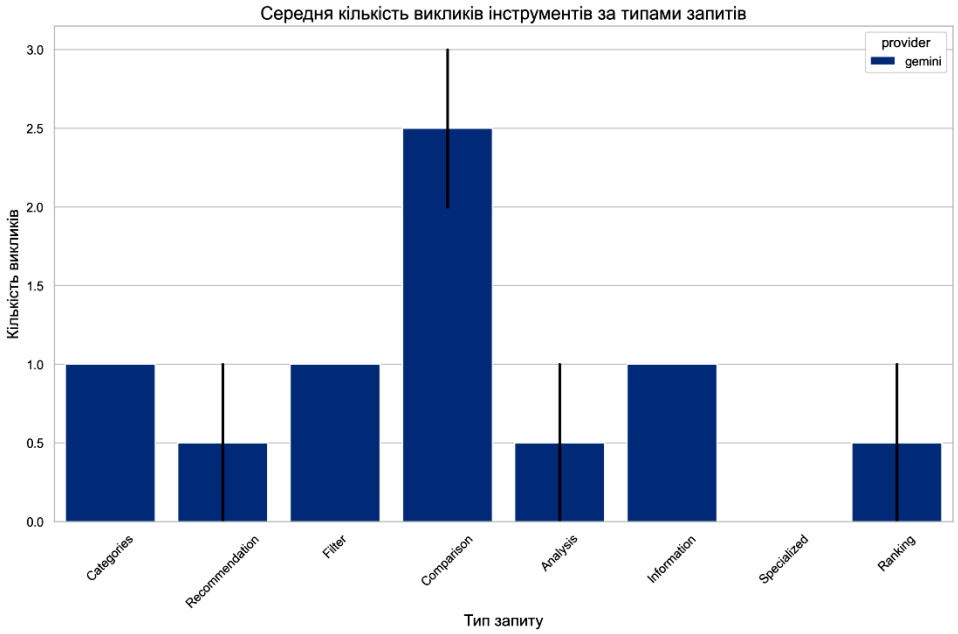

Рисунок 7 – Кількість інструментальних викликів (tool calls) у тестовій серії

Аналіз кількості інструментальних викликів (рис. 7) виявляє пряму кореляцію між складністю запиту та кількістю звернень до бази даних. Найпростіші запити (Categories) вимагають рівно одного звернення до БД. Запити на порівняння товарів (Comparison) в середньому потребують 2.5 tool-виклики, що відповідає необхідності отримання даних про кожен товар для подальшої генерації порівняльної таблиці. Цікаво, що запити типу Recommendation та Analysis в половині випадків не потребують звернення до БД – система генерує відповіді на основі загальних знань або контексту попередніх запитів.

Підсумки по питаннях (RQ):

– RQ1 (зменшення помилок за рахунок RAG+tools): на 20 сценаріях зафіксовано 0/20 помилкових відповідей; для чистої LLM у логах базової лінії присутні хибні відповіді на складні запити.

- RQ2 (Smart Search для нечітких запитів): усі 12/12 сценаріїв рівнів Medium+Hard завершилися успішно з коректним підбором категорій/товарів; фіксована кількість tool_calls (середнє 1.0–2.5) підтверджує керовану розвідку.
- RQ3 (TaskChain для порівнянь): середній час для порівняльних запитів 4.966 с при 2.5 інструментальних викликах; відхилення часу невелике (див. розподіл на рис. 6), усі порівняльні сценарії виконані коректно.

Статистика часу (за CSV `benchmark_results_20250409_003505.csv`, усі сценарії ×2 запуски):

- Categories: min 1.783 с, median 2.105 с, max 2.427 с, std ≈ 0.32 с.
- Recommendation: min 1.465 с, median 1.965 с, max 2.784 с, std ≈ 0.48 с.
- Filter: min 2.526 с, median 5.087 с, max 9.049 с, std ≈ 2.92 с (найбільша варіація через обсяг вибірки та довгі відповіді).
- Comparison: min 4.680 с, median 4.966 с, max 5.252 с, std ≈ 0.29 с.
- Analysis: min 2.013 с, median 3.532 с, max 5.050 с, std ≈ 1.52 с.
- Information: min 4.855 с, median 5.006 с, max 5.157 с, std ≈ 0.15 с.
- Specialized: min 2.373 с, median 2.987 с, max 3.601 с, std ≈ 0.61 с.
- Ranking: min 1.584 с, median 2.331 с, max 3.079 с, std ≈ 0.75 с.
-

Порівняння з базовими лініями: Rule-based не покриває запити Hard і дає неповні відповіді на нечіткі запити; «чиста» LLM без доступу до БД демонструвала неконсистентність фактів у складних сценаріях. Запропонована архітектура RAG+tools показала стабільний success rate 100% і контрольований час відповіді на всіх 20 сценаріях.

Якість класифікації намірів (Intent Recognition) оцінюється шляхом порівняння визначеного системою наміру з еталонною розміткою для контрольного набору сценаріїв. Типові помилки виникають у випадках, коли запит користувача є надто коротким або неоднозначним (наприклад, короткі однослівні запити можуть відповідати різним намірам у залежності від контексту та вмісту БД).

Основними загрозами валідності дослідження є залежність від LLM-провайдера, оскільки end-to-end затримки та стиль відповіді залежать від зовнішнього сервісу, мережевих умов і політик лімітування. Існує також обмеженість датасету – тестова БД містить 190 товарів у 6 категоріях, а поле `attributes` хоча і має тип `jsonb`, у поточному наборі даних не заповнене (усі значення `{}`), що обмежує можливості оцінки фільтрації за динамічними атрибутами. Апаратні обмеження та вимоги до продуктивності нейромережевих систем також впливають на ефективність реалізації [10]. Крім того, розмір тестового набору обмежений – серія з 20 сценаріїв дозволяє порівняти типи запитів, але не гарантує статистичної узагальненості для всіх можливих формулювань користувачів. Зсув даних (data drift) при оновленні каталогу може змінити розподіл запитів і категорій. Мультиплатформеність реалізована через адаптери; Discord слугує референсним каналом, але архітектура допускає підключення браузера/Viber/Telegram без змін ядра.

4. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Отримані результати підтверджують ефективність запропонованої гібридної архітектури RAG+tools для задач автоматизації клієнтської підтримки в e-commerce. Нульова частка помилок (0/20) на тестовому наборі свідчить про успішне вирішення проблеми галюцинацій через обов'язкове підкріплення відповідей фактичними даними з БД. Середній час відповіді 3.589 с є прийнятним для інтерактивної взаємодії, хоча варіативність часу для запитів типу Filter (std ≈ 2.92 с) вказує на залежність продуктивності від обсягу контексту та складності SQL-запитів.

Механізм Smart Search продемонстрував стійкість до нечітких формулювань: усі 12 сценаріїв Medium+Hard завершилися успішно завдяки керованій розвідці через додаткові tool-виклики. Це підтверджує гіпотезу про перевагу агентного циклу «запит–перевірка–уточнення» над фіксованими евристиками. TaskChain для порівняльних запитів показав стабільний час (std ≈ 0.29 с) та коректність виконання, що свідчить про ефективність декомпозиції складних завдань на підзадачі.

Для визначення ринкової цінності розробки проведено порівняння з провідними SaaS-рішеннями: Shopify Inbox, Tidio та Intercom. Оцінювання проводилося за бальною шкалою (0-1) по 22 критеріях, згрупованих у категорії: "Розуміння мови", "Інтеграція", "Гнучкість" та "Вартість". Узагальнений результат аналізу зображено на рис. 8.

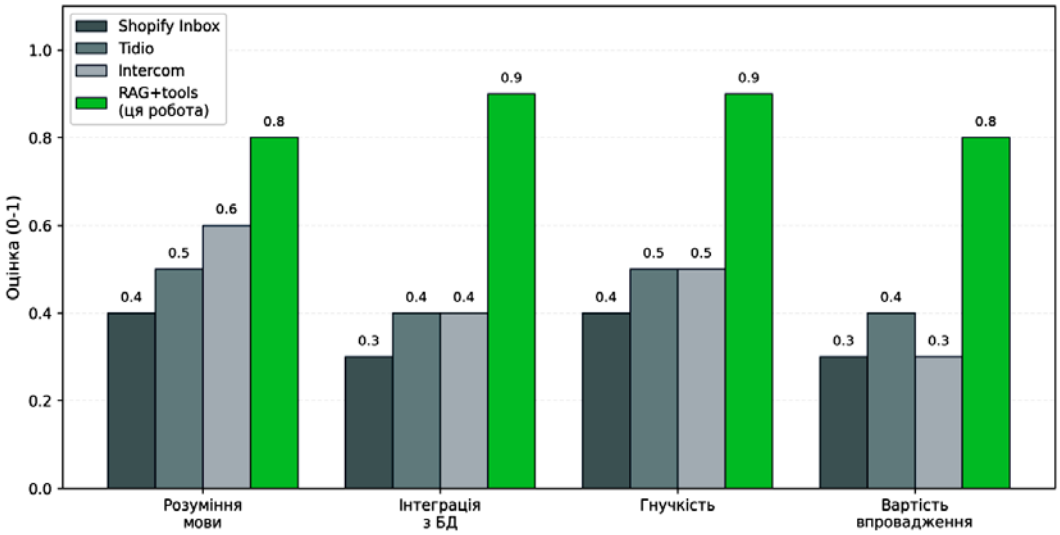

Рисунок 8 – Результати порівняльного аналізу функціональності

Порівняння з комерційними платформами доцільно інтерпретувати як якісний аналіз, оскільки точні бальні оцінки залежать від методики зважування критеріїв та доступу до внутрішніх API/функцій SaaS-систем. Основні переваги запропонованого підходу включають контрольоване використання даних, оскільки відповіді формуються на основі фактів з БД через інструменти, що знижує ризик галюцинацій; гнучкість інтеграції через можливість адаптації під конкретну схему БД, включаючи використання `JSONB` атрибутів при їх заповненні, без прив'язки до закритої платформи; відсутність vendor lock-in завдяки можливості змінювати LLM-провайдера та транспортну платформу за рахунок модульної архітектури.

Розроблена система може бути впроваджена в інтернет-магазинах різного масштабу без значних витрат на розробку, оскільки адаптація зводиться до налаштування конфігурації та підключення до існуючої БД. Модульна архітектура дозволяє поетапне впровадження: спочатку як асистент оператора (human-in-the-loop), потім як автономний бот для типових запитів. Підтримка мультиплатформеності через адаптери забезпечує охоплення різних каналів комунікації (веб, месенджери, соціальні мережі) без дублювання логіки.

ВИСНОВКИ

У роботі вирішено актуальне науково-прикладне завдання автоматизації клієнтської підтримки в електронній комерції. Розроблено та досліджено адаптивний бот-консультант, який поєднує інтелектуальні можливості великих мовних моделей з достовірністю структурованих даних.

Проаналізовано методи побудови діалогових систем та обґрунтовано вибір гібридної архітектури RAG (Retrieval-Augmented Generation), яка поєднує нейромережу «Трансформер» з реляційною базою даних. Цей підхід дозволяє уникнути основного недоліку "чистих" LLM – галюцинацій – шляхом обов'язкового підкріплення відповідей фактичними даними з бази даних.

Реалізовано механізм «розумного пошуку» як поєднання евристичної категоризації запиту та керованої «розвідки» через інструменти доступу до БД. Запропонований agent-loop з механізмом reconnaissance забезпечує стійкість системи до нечітких формулювань та дозволяє знаходити релевантну інформацію навіть при неповних або неоднозначних запитах користувачів.

Розроблено програмний комплекс мовою Python (FastAPI, Discord.py), що забезпечує інтеграцію LLM із реляційною БД та підтримує обробку багатокрокових запитів через агентний цикл. Модульна архітектура системи дозволяє легко розширювати функціональність та адаптувати її під специфічні вимоги різних інтернет-магазинів.

Практичне значення роботи полягає у створенні універсального інструменту, який може бути легко адаптований під будь-який інтернет-магазин шляхом зміни конфігурації, без необхідності переписування коду. Система підтримує різні платформи взаємодії з користувачами та може інтегруватися з існуючими інфраструктурами e-commerce.

Експериментальні заміри на фіксованому наборі сценаріїв показали середній end-to-end час обробки 3.589 с для провайдера Gemini при 0% помилок у тестовій серії; найбільші значення часу

спостерігалися для запитів класу «Filter» (через довші відповіді та більший обсяг контексту), а також «Comparison» (через декілька інструментальних викликів).

Перспективи подальших досліджень пов'язані з інтеграцією мультимодальних можливостей (пошук за фотографією товару), впровадженням навчання з підкріпленням (RLHF) для автоматичного покращення відповідей на основі реакцій користувачів, а також розширенням підтримки мов для забезпечення міжнародної доступності системи. Додатковим напрямком розвитку є оптимізація продуктивності через кешування запитів та паралельну обробку інструментальних викликів з урахуванням апаратних обмежень [10].

СПИСОК ВИКОРИСТАННОЇ ЛІТЕРАТУРИ / REFERENCES

1. Online Shopping Experiences Are Changing Due to Chatbots & LLM [Електронний ресурс]. – Режим доступу: https://www.okoone.com/spark/technology-innovation/online-shopping-experiences-are-
2. Shevchuk V. O. Electronic commerce: theory and practice: textbook / V. O. Shevchuk. – Kyiv: Publishing house “Professional”, 2022. – 400 p.
3. Vaswani A. Attention is All You Need / A. Vaswani et al. // Advances in Neural Information Processing Systems. – 2017. – Vol. 30. – URL: https://arxiv.org/abs/1706.03762.
4. Lewis P. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks / P. Lewis et al. // Advances in Neural Information Processing Systems. – 2020. – Vol. 33. – P. 9459–9474. – URL: https://arxiv.org/abs/2005.11401.
5. Ponomarenko L. A. Databases: design and administration: a manual / L. A. Ponomarenko. – Kharkiv: V. N. Karazin KhNU, 2018. – 256 p.
6. Gartner Reveals Three Technologies That Will Transform Customer Service and Support by 2028 [Електронний ресурс]. – Режим доступу: https://www.gartner.com/en/newsroom/press-releases/2023-08-30-gartner-reveals-three-technologies-that-will-transform-customer-service-and-support-by-2028.
7. Manning C. D. Introduction to Information Retrieval / C. D. Manning, P. Raghavan, H. Schütze. – Cambridge : Cambridge University Press, 2008. – 482 p.
8. Google AI for Developers. Gemini API Documentation [Access mode]. – Режим доступу: https://ai.google.dev/gemini-api/docs.
9. PostgreSQL Documentation. JSON Types [Access mode:]. – Режим доступу: https://www.postgresql.org/docs/current/datatype-json.html.
10. Kolesnitsky O. K. Analytical review of hardware implementations of spiking neural networks / O. K. Kolesnitsky // Mathematical Machines and Systems. – 2015. – No. 1. – P. 3–19. [Electronic resource]. – Access mode: http://www.immsp.kiev.ua/publications/articles/2015/2015_1/01_2015_Kolesnytskyy.pdf.

Дата надходження: 10.01.2026
Дата прийняття до друку після рецензування: 28.03.2026
Дата публікації: 18.06.2026

КОЛЕСНИЦЬКИЙ ОЛЕГ КОСТЯНТИНОВИЧ – кандидат технічних наук, професор кафедри комп’ютерних наук, Вінницький національний технічний університет, м. Вінниця, Україна. Сфера наукових інтересів: нейронні мережі, розпізнавання образів, інтелектуальні інформаційні системи
E-mail: kolesnytskiy@vntu.edu.ua. **https://orcid.org/0000-0003-0336-4910**

МІРОШНИЧЕНКО СТАНІСЛАВ ОЛЕКСАНДРОВИЧ – студент групи 2КН-25М, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м. Вінниця, Україна. Сфера наукових інтересів: штучний інтелект, машинне навчання, веб-технології, NLP, ***mail: stasmirosnicenko@gmail.com*****, https://orcid.org/0009-0005-2939-0131**

Oleh KOLESNYTSKIY, Stas MIROSNICHENKO

ADAPTIVE BOT-CONSULTANT FOR E-COMMERCE SYSTEMS BASED ON THE NEURAL NETWORK ARCHITECTURE “TRANSFORMER”
Vinnytsia National Technical University