НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Використання AI-асистента в якості експерта з технічних консультацій»

Виконав (-ла):
студент (-ка) IV курсу, групи КІ-03
Чайка Максим Васильович _____

Керівник:
доктор філософії з комп'ютерних наук,
доцент кафедри штучного інтелекту
Гуськова Віра Геннадіївна _____

Консультант з економічного розділу:
доцент, доктор філософії з економіки,
доцент кафедри економічної кібернетики ФММ,
Мажара Гліб Анатолійович _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ,
Кравець Павло Володимирович _____

Рецензент:
доктор філософії з системного аналізу,
старший викладач кафедри ММСА
Левенчук Людмила Борисівна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«31» січня 2024 р.

ЗАВДАННЯ
на дипломну роботу студенту
Чайка Максим Васильович

1. Тема роботи «**Використання AI-асистента в якості експерта з технічних консультацій**», керівник роботи доктор філософії, доцент, Гуськова Віра Геннадіївна, затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом роботи «13» червня 2024 року.

3. Вихідні дані до роботи: набір метрик обробки запитів користувачів сервісу технічної підтримки.

4. Зміст роботи: Поточні виклики у процесах підтримки; Велика мовна модель; Проєктування та розробка автономного асистента з використанням RAG Функціонально-вартісний аналіз програмного продукту.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): рисунки архітектури LLM, гістограми та графіки розподілу, схеми архітектури власного рішення.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Мажара Гліб Анатолійович, доцент, доктор філософії з економіки		

7. Дата видачі завдання «05» лютого 2024 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою	26.04.2024	Виконано
2	Підготовка першого розділу	29.04.2024	Виконано
3	Підготовка другого розділу	08.05.2024	Виконано
4	Розробка програмного продукту	23.05.2024	Виконано
5	Підготовка третього розділу	04.06.2024	Виконано
6	Підготовка економічної частини	08.06.2024	Виконано
7	Оформлення розділів відповідно до нормоконтролю	10.06.2024	Виконано
8	Підготовка презентації доповіді	11.06.2024	Виконано
9	Оформлення дипломної роботи	12.06.2024	Виконано

Студент Максим ЧАЙКА

Керівник Віра ГУСЬКОВА

РЕФЕРАТ

Дипломна робота: 142 с., 32 рис., 11 табл., 44 посилань, 3 додатки.

ТЕХНІЧНІ КОНСУЛЬТАЦІЇ, АВТОНОМНИЙ АСИСТЕНТ, ВЕЛИКІ МОВНІ МОДЕЛІ, RAG, GPT, ГЕНЕРАТИВНИЙ ШІ, ДОНАВЧАННЯ, LANGCHAIN.

Актуальність роботи: зростаюча потреба сучасних підприємств у забезпеченні високої якості обслуговування, зниженні витрат та підвищенні продуктивності шляхом впровадження автоматизованих систем. Аналіз можливостей штучного інтелекту як технічного експерта дозволяє не тільки глибше зрозуміти його потенціал, але й визначити методи його ефективної інтеграції у поточні бізнес-процеси.

Об'єктом дослідження є процес надання технічних консультацій.

Предметом дослідження є використання штучного інтелекту як експертної системи в процесі надання технічних консультацій.

Мета роботи – визначення стратегій інтеграції ШІ у процеси технічних консультацій та розробка автономного асистента для обробки звернень користувачів.

Результати роботи: аналіз метрик технічної підтримки виявив взаємозв'язок між якістю обслуговування та задоволеністю користувачів. Було порівняно використання RAG та донавчання для задачі синтезу відповіді. Розроблено архітектуру автономного асистента та інтегровано його у CRM-систему.

ABSTRACT

Bachelor's thesis: 142 p., 32 figures, 11 tables, 44 references, 3 appendixes.

TECHNICAL CONSULTATIONS, AUTONOMOUS ASSISTANT, LARGE LANGUAGE MODELS, RAG, GPT, GENERATIVE AI, FINE-TUNING, LANGCHAIN.

The relevance of the work lies in the increasing need of modern enterprises to ensure high-quality service, reduce costs, and increase productivity through the implementation of automated systems. Analyzing the possibilities of artificial intelligence as a technical expert allows not only an in-depth understanding of its potential but also the determination of methods for its effective integration into current business processes.

The object of the research is the process of providing technical consultations.

The subject of the research is the use of artificial intelligence as an expert system in the process of providing technical consultations.

The purpose of the work is to determine the strategies for integrating AI into technical consultation processes and to develop an autonomous assistant for handling user inquiries.

Results: an analysis of technical support metrics revealing a correlation between service quality and user satisfaction. The use of RAG and fine-tuning for response synthesis was compared. An architecture for an autonomous assistant was developed and integrated into a CRM system.

ЗМІСТ

ВСТУП

Розвиток інформаційних технологій і штучного інтелекту (ШІ) в останні десятиліття відкрив нові горизонти для їх застосування у різноманітних галузях людської діяльності. Однією з найбільш перспективних сфер використання ШІ є автоматизація процесів технічного обслуговування та надання консультаційних послуг. В цьому контексті, ШІ не просто виступає як інструмент оптимізації, але і як повноцінний експерт, здатний аналізувати значні обсяги даних, швидко ідентифікувати проблемні зони та надавати кваліфіковані рекомендації.

Актуальність даного дослідження полягає в наростальній потребі сучасних підприємств у забезпеченні високої якості обслуговування, мінімізації витрат і збільшенні продуктивності через впровадження автоматизованих систем. Вивчення можливостей використання ШІ в якості технічного експерта дозволяє не лише глибше зрозуміти його потенціал, але й визначити способи його ефективної інтеграції в наявні бізнес-процеси.

Метою є визначення стратегій інтеграції ШІ у сферу технічної підтримки та розробка асистента для автономної обробки звернень користувачів.

Об’єктом дослідження виступає процес надання технічних консультацій, а предметом – використання штучного інтелекту як експертної системи в цьому процесі.

Дослідження базується на аналізі наукових публікацій, вивченні наявних технологічних рішень та визначенні кращих практик застосування ШІ у контексті технічних консультацій.

РОЗДІЛ 1 РОЛЬ ШТУЧНОГО ІНТЕЛЕКТУ В ОПТИМІЗАЦІЇ ПРОЦЕСІВ ТЕХНІЧНИХ КОНСУЛЬТАЦІЙ

1.1 Актуальність

Вивчення можливостей ШІ наразі вкрай актуальне, адже світ цифрових технологій розвивається дуже швидко, а вимоги клієнтів до якості та швидкості сервісу постійно зростають.

Сучасний ринок виштовхує компанії на шлях невпинних технологічних нововведень, вимагаючи технічних рішень, які відповідають високому рівню потреб та бажань споживачів. Впровадження штучного інтелекту в процеси надання технічних консультацій може вагомо покращити ефективність обслуговування, спрощуючи численні процеси та забезпечуючи прийняття рішень на основі комплексного аналізу великих обсягів даних.

Розростання даних та збільшення складності систем формують потребу в інноваційних інструментах управління та аналізу даних. Системи, забезпечені штучним інтелектом, можуть автоматично адаптуватись до нових умов обслуговування, оптимізуючи основні процедури та знижуючи можливість помилок.

Крім того, ШІ робить значний внесок у персоналізацію сервісу, завдяки аналізу поведінки клієнтів та їх вподобань, надаючи індивідуалізовані рекомендації, що суттєво підвищує рівень задоволеності клієнтів. У світі, де персоналізація стає основною умовою успішної взаємодії зі споживачами, можливості штучного інтелекту для розуміння та задоволення індивідуальних потреб клієнтів можуть стати рішучим чинником у забезпеченні конкурентоспроможності компаній.

Таким чином, зростальна роль ШІ у процесах технічного обслуговування відображає ширший тренд цифровізації та автоматизації, який охоплює всі сфери бізнесу. Інвестиції в ШІ-технології не тільки підвищують ефективність обслуговування, а й формують основу для інноваційних підходів до клієнтської

взаємодії, що робить їх стратегічно важливими для будь-якої компанії, яка прагне залишатися на передовій технологічного прогресу.

1.2 Аналіз поточних викликів у процесах підтримки

Сфера підтримки зіштовхується з низкою складнощів, які істотно впливають на якість наданих послуг та рівень задоволеності клієнтів. Розуміння цих проблем є критично важливим для розробки ефективного рішення, здатного оптимізувати чинні процеси та підвищити ефективність взаємодії з користувачами.

Клієнти часто звертаються з різноманітними та технічно складними питаннями, що вимагають високої кваліфікації консультантів та глибокого розуміння продукту. Людський фактор може вносити суб'єктивність та непослідовність у відповіді, що призводить до можливих помилок та некоректних рішень.

Оскільки максимальна кількість одночасних клієнтів, яких може обробити оператор скінченна, збільшення обсягу запитів, особливо у часи високих навантажень, може призвести до затримок у відповідях [1]. Задоволеність клієнтів напряму залежить від темпу реагування на їхні проблеми, тому будь-які затримки можуть негативно позначитися на репутації компанії.

Технічна служба підтримки збирає та обробляє величезні обсяги інформації, що включає історію запитів, технічні характеристики продуктів, бази знань та інші технічні дані. Іноді ланцюжок обробки цих даних займає істотний час, що потенційно веде до затримок у наданні відповідей.

Системи технічної підтримки часто не пристосовані до раптового збільшення обсягу запитів, що може виникнути внаслідок масових несправностей або випуску нових продуктів. Це призводить до перевантаження

системи, додаткового стресу для персоналу та зниження загальної якості обслуговування [1].

Ці виклики вимагають впровадження технологічних рішень, що можуть забезпечити більшу ефективність, точність та швидкість у наданні технічних консультацій.

1.3 Застосування ШІ у сфері консультацій

Впровадження асистента на базі ШІ у процес надання консультаційних послуг відкриває нові горизонти для підвищення ефективності, точності та швидкості обслуговування клієнтів [2].

Штучний інтелект здатний автоматизувати рутинні завдання, такі як обробка запитів, сортування та класифікація звернень, що дозволяє зменшити навантаження на людський ресурс. Це особливо важливо у випадках пікових навантажень, коли обсяг запитів значно зростає. Автоматизація дозволяє швидко та ефективно обробляти великі обсяги даних, що знижує час реагування та підвищує загальну продуктивність. Автоматизація процесів за допомогою ШІ включає використання алгоритмів машинного навчання, які можуть навчатися на основі історичних даних і постійно вдосконалювати свої моделі. Це дозволяє швидко адаптуватися до змін у поведінці користувачів та нових типів запитів, що виникають з часом.

Наприклад, чат-боти на базі ШІ, можуть автоматично оновлювати свої відповіді на основі нових даних, що надходять, забезпечуючи актуальність і точність інформації.

Такі рішення можуть аналізувати великі обсяги даних з високою точністю, що дозволяє зменшити кількість помилок, пов'язаних з людським фактором [5]. Використання алгоритмів машинного навчання для виявлення та виправлення помилок у реальному часі забезпечує більш точні та послідовні відповіді на

запити клієнтів. Алгоритми машинного навчання можуть виявляти патерни та аномалії у великих наборах даних, що дозволяє швидко ідентифікувати потенційні проблеми та пропонувати оптимальні рішення [3].

Наприклад, вони можуть аналізувати історію запитів клієнтів і виявляти повторювані проблеми, що дає змогу розробити більш ефективні стратегії для їх вирішення. Крім того, ШІ може використовувати методи глибокого навчання для аналізу складних технічних даних, що забезпечує високу точність і надійність результатів.

Завдяки можливостям обробки природної мови (NLP) та машинного навчання, можна ідентифікувати та відреагувати на запити клієнтів. Це дозволяє значно скоротити час очікування відповіді, що підвищує рівень задоволеності клієнтів. Крім того, автономний асистент може працювати цілодобово, забезпечуючи безперервну підтримку без необхідності залучення додаткового персоналу [6]. Технології обробки природної мови дозволяють розуміти контекст і зміст запитів клієнтів, що забезпечує більш точні та релевантні відповіді.

Завдяки методам семантичного аналізу, чат-бот на базі ШІ може аналізувати текстові повідомлення клієнтів і надавати детальні та точні відповіді, що максимально відповідають потребам клієнта. ШІ-системи можуть аналізувати поведінку клієнтів та їхні вподобання, що дозволяє надавати індивідуалізовані рекомендації та рішення [4]. Це підвищує рівень задоволеності користувачів, оскільки вони отримують максимально релевантні до їх потреб та очікувань відповіді. Персоналізація стає ключовим фактором у забезпеченні конкурентоспроможності компаній. Персоналізація обслуговування включає використання алгоритмів рекомендацій, які можуть аналізувати історію взаємодії клієнтів з компанією та пропонувати індивідуальні рішення.

Системи ШІ можуть аналізувати попередні запити клієнтів і пропонувати рішення, що базуються на їхніх попередніх вподобаннях та потребах. Це

дозволяє забезпечити більш релевантні та персоналізовані відповіді, що підвищує рівень задоволеності клієнтів.

ШІ здатний швидко обробляти та аналізувати великі обсяги даних, що включають історію запитів, технічні характеристики продуктів, бази знань та інші технічні дані. Це дозволяє оперативно знаходити необхідну інформацію та надавати точні відповіді на запити клієнтів, що знижує час обробки даних та підвищує ефективність обслуговування. Також можна використовувати методи великих даних (Big Data) для аналізу великих обсягів інформації у реальному часі. Це дозволяє швидко виявляти тенденції та патерни, що можуть бути корисними для пошуку відповідей на технічні питання. Наприклад, системи можуть аналізувати дані про продуктивність продуктів і виявляти потенційні проблеми до того, як вони стануть критичними. Це дозволяє забезпечити проактивне обслуговування та знизити ризик виникнення серйозних проблем.

Впровадження ШІ дозволяє знизити витрати на технічне обслуговування внаслідок автоматизації рутинних завдань та зменшення потреби у великій кількості персоналу. Це дозволяє компаніям ефективніше використовувати свої ресурси та інвестувати у розвиток інноваційних технологій. Зниження витрат включає автоматизацію процесів, що дозволяє зменшити потребу у великій кількості персоналу та знизити витрати на навчання та підтримку [7].

Крім того, ШІ дає можливість використовувати методи прогнозування для аналізу даних про продуктивність обробки запитів і виявлення потенційних проблем, що дозволяє знизити витрати на ремонт та обслуговування.

Таким чином, інтеграція штучного інтелекту у механізми технічного обслуговування надає значні переваги, що дозволяють вирішити існуючі проблеми та підвищити загальну ефективність обслуговування клієнтів. ШІ стає ключовим інструментом для забезпечення високої якості сервісу, що є стратегічно важливим для будь-якої компанії, яка прагне залишатися на передовій технологічного прогресу.

1.4 Обмеження використання штучного інтелекту

Впровадження ШІ у сферу технічного обслуговування та надання консультаційних послуг, безсумнівно, відкриває нові горизонти для підвищення ефективності, точності та швидкості обслуговування клієнтів. Проте, як і будь-яка інноваційна технологія, ШІ має свої обмеження та виклики, які необхідно враховувати для забезпечення його успішного та етичного використання. Розглянемо основні обмеження, пов'язані з використанням ШІ у сфері обслуговування користувачів, а також нормативно-правові аспекти, що регулюють цю діяльність.

1.4.1 Етичні та соціальні обмеження

Одним з ключових обмежень використання ШІ є етичні та соціальні аспекти. Використання ШІ у взаємодії з клієнтами може викликати занепокоєння щодо приватності, безпеки даних та можливих упереджень у прийнятті рішень [7]. Тобто, алгоритми машинного навчання можуть почати виявляти упередження, якщо вони були навчені на даних, що містять ці історичні упередження. Це може призвести до дискримінації певних груп користувачів, що є неприпустимим з етичного погляду. Етичні аспекти також включають питання відповідальності за прийняття рішень, що базуються на результатах роботи ШІ.

Для мінімізації цих ризиків необхідно забезпечити прозорість даних якими користується ШІ та їхню відповідність етичним стандартам. Це включає регулярний аудит алгоритмів, аналіз їхньої роботи на предмет упереджень та розробку механізмів для їхнього виправлення. Крім того, важливо забезпечити

інформовану згоду користувачів на використання їхніх даних для навчання та роботи ШІ-систем.

Дослідники стверджують, що слід більше уваги приділяти соціальним і психологічним наслідкам систем штучного інтелекту, а також збільшувати фінансування програм розвитку соціальних навичок і підтримки для пом'якшення цих проблем [8].

1.4.2 Нормативно-правові обмеження

Сфера обслуговування користувачів регулюється низкою нормативно-правових актів, що спрямовані на захист прав користувачів та забезпечення безпеки їхніх даних. Одним з основних нормативних документів у цій сфері є Загальний регламент захисту даних (GDPR), який діє на території Європейського Союзу. GDPR встановлює суворі вимоги до обробки персональних даних, зокрема необхідність отримання інформованої згоди користувачів, забезпечення права на доступ до даних та їхнє видалення [9].

Для дотримання вимог GDPR компанії повинні забезпечити прозорість процесів обробки даних, розробити політики конфіденційності та забезпечити захист даних від несанкціонованого доступу. Крім того, необхідно забезпечити можливість користувачам реалізувати свої права на доступ до даних, їхнє виправлення та видалення.

Важливо також враховувати інші нормативно-правові акти, що регулюють використання ШІ у різних юрисдикціях, та забезпечити відповідність усім вимогам. Нормативно-правові обмеження також включають питання ліцензування та сертифікації ШІ-систем. У деяких галузях, таких як медицина або фінанси, використання ШІ може вимагати додаткових дозволів та сертифікацій, що підтверджують відповідність системи встановленим стандартам та вимогам.

1.4.3 Вимоги до безпеки та надійності

Безпека та надійність є критично важливими аспектами, особливо у сфері обслуговування користувачів. Використання ШІ у сфері технічного обслуговування вимагає забезпечення високого рівня безпеки даних та захисту від кібератак. Це включає розробку та впровадження заходів для захисту даних від несанкціонованого доступу, забезпечення цілісності даних та запобігання їхньому витоку.

Це дозволяє забезпечити стабільну та безперебійну роботу ШІ-систем, що є критичним для забезпечення та підтримки високої якості обслуговування користувачів. Надійність та безпека ШІ-систем є ключовими аспектами, що дозволяють забезпечити довгострокову ефективність та стабільність роботи ШІ у сфері обслуговування користувачів.

Заходи, які доцільно впроваджувати задля досягнення цих вимог:

- розробка заходів безпеки, що включають методи захисту даних від несанкціонованого доступу та кібератак;
- регулярне тестування та моніторинг роботи для виявлення та виправлення помилок;
- розробка методів захисту від зловмисних атак на навчальні дані та маніпуляцій з результатами роботи ШІ.

Висновки до розділу 1

У першому розділі було проведено аналіз ролі ШІ в оптимізації процесів технічних консультацій, що є надзвичайно актуальним, оскільки вимоги клієнтів до якості та швидкості сервісу постійно зростають. Впровадження ШІ у процеси надання відповідей дозволяє значно підвищити ефективність

обслуговування, спрощуючи численні процеси та забезпечуючи прийняття рішень на основі комплексного аналізу великих обсягів даних.

Аналіз поточних викликів у процесах технічної підтримки виявив низку проблем, таких як складність та гетерогенність запитів, час реагування, оперативність обробки великих обсягів даних та неоптимізованість до пікових навантажень. Ці виклики вимагають впровадження технологічних рішень, що можуть забезпечити більшу, швидкість, точність та ефективність.

Інтеграція ШІ у процеси технічного обслуговування відкриває нові горизонти для автоматизації та оптимізації процесів, підвищення точності та зменшення помилок, швидкого реагування на запити, персоналізації обслуговування, оперативного аналізу великих обсягів даних та зниження витрат. Це дозволяє компаніям ефективніше використовувати свої ресурси та більше інвестувати у розвиток.

Однак, впровадження ШІ також має свої обмеження і виклики, які необхідно враховувати для забезпечення його успішного та етичного використання. Серед них етичні та соціальні обмеження, нормативно-правові обмеження, проблеми з інтерпретацією та пояснюваністю, вимоги до безпеки та надійності, а також вимоги до інфраструктури. Для подолання цих обмежень необхідно впроваджувати відповідні заходи, такі як розробка етичних кодексів та стандартів, розробка заходів безпеки, регулярне тестування та моніторинг, а також розвиток інфраструктури.

Таким чином, роль ШІ у процесах підтримки відображає ширший тренд цифровізації та автоматизації, який охоплює всі сфери бізнесу. Інвестиції в ШІ-технології не тільки підвищують ефективність обслуговування, а й формують основу для інноваційних підходів до клієнтської взаємодії, що робить їх стратегічно важливими для будь-якої компанії, яка прагне залишатися на передовій технологічного прогресу.

РОЗДІЛ 2 РОЛЬ ШТУЧНОГО ІНТЕЛЕКТУ В ОПТИМІЗАЦІЇ ПРОЦЕСІВ ТЕХНІЧНИХ КОНСУЛЬТАЦІЙ

2.1 Велика мовна модель та переднавчання

Великі мовні моделі (LLM) є одними з найпотужніших інструментів у сучасному арсеналі штучного інтелекту. LLM, такі як GPT (Generative Pre-trained Transformer) та Gemini, здатні обробляти та генерувати текст на основі величезних обсягів даних, що дозволяє їм розуміти контекст та семантику природної мови на високому рівні.

2.1.1 Архітектура великих мовних моделей

В основі LLM лежить архітектура трансформерів, які були вперше запропоновані у роботі "Attention is All You Need" (Vaswani et al., 2017) [11]. Трансформери використовують механізм самоуваги (self-attention), що дозволяє моделі ефективно обробляти залежності між словами у тексті незалежно від їхньої відстані один від одного.

Трансформери є основою архітектури великих мовних моделей. Вони складаються з двох компонентів: енкодера та декодера. Енкодер обробляє вхідний текст, а декодер синтезує вихідний текст на основі обробленої інформації.

Одним з основних компонентів трансформера є механізм самоуваги (Self-Attention Mechanism): завдяки цьому механізму модель може визначати, які частини вхідного тексту є найбільш релевантними для поточного контексту (рис. 2.1). Він обчислює значущість уваги для кожного слова у тексті, що дозволяє моделі фокусуватися на важливих словах. Механізм самоуваги обчислює три матриці: запити (queries), ключі (keys) та значення (values). Ваги

уваги обчислюються як скалярний добуток запитів та ключів, нормалізований за допомогою softmax-функції:

$$Attention(Q,\ K,\ V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)v,$$

де Q — матриця запитів (queries);

K — матриця ключів (keys);

V — матриця значень (values);

d_k — розмірність ключів.

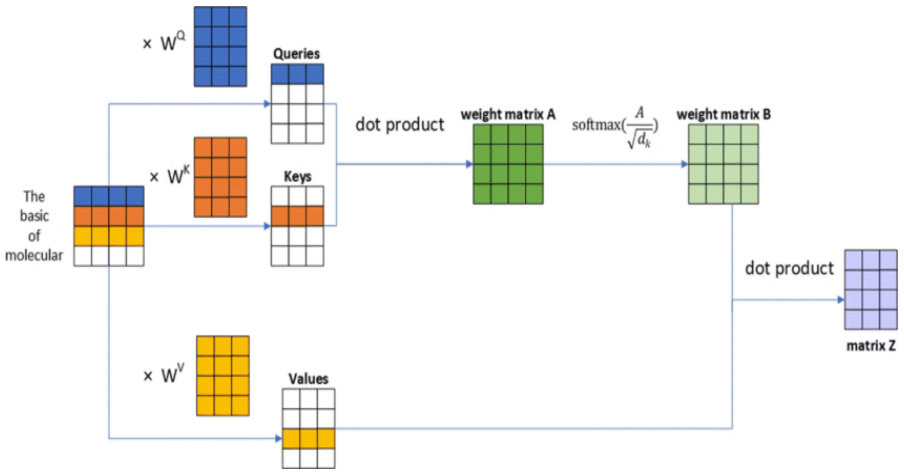

Рисунок 2.1 – Процес розрахунку механізму самоуваги

Для покращення здатності моделі до захоплення різних аспектів залежностей між словами, трансформери використовують мультиголову увагу (рис. 2.2). Це означає, що механізм самоуваги обчислюється кілька разів паралельно з різними наборами ваг, а результати об'єднуються. Завдяки цьому модель може одночасно фокусуватися на різних частинах вхідного тексту.

Кожен шар трансформера включає нормалізацію (layer normalization) та залишкові зв'язки (residual connections), що допомагає стабілізувати навчання та покращити передачу градієнтів. Нормалізація зменшує вплив зміни масштабу

вхідних даних, а залишкові зв'язки дозволяють зберігати інформацію з попередніх шарів.

Після механізму самоуваги кожен шар трансформера включає повнозв'язні шари, які виконують нелінійні перетворення вхідних даних. Ці шари складаються з двох лінійних перетворень з функцією активації між ними, зазвичай ReLU (Rectified Linear Unit).

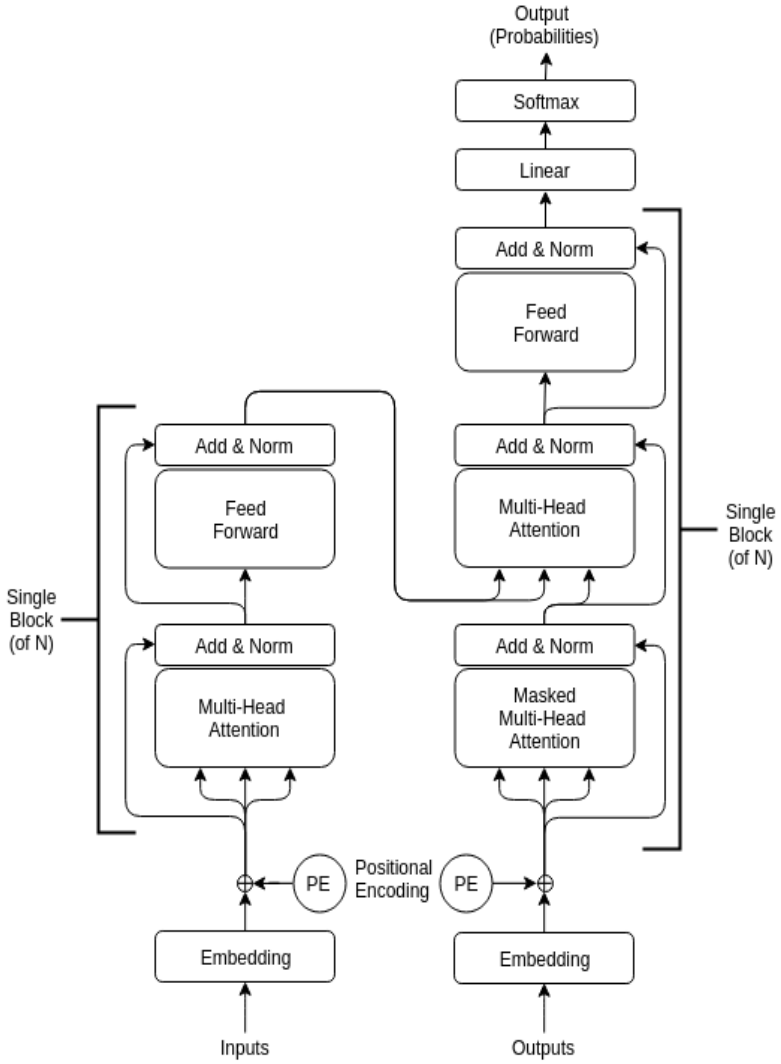

Рисунок 2.2 – Архітектура моделі трансформера

Оскільки трансформери не мають вбудованого механізму для обробки порядку слів у тексті, використовується позиційне кодування. Це додаткові вектори, які додаються до вхідних векторів слів, щоб зберегти інформацію про

їхню позицію у тексті. Позиційне кодування зазвичай обчислюється за допомогою синусоїдальних функцій різних частот.

Резюмуючи, архітектура трансформера складається з кодувальника та декодера, кожен з яких складається з N блоків. Вхідні дані — це послідовність подій, вихідні дані — це послідовність прогнозованих подій.

2.1.2 Архітектура GPT

Архітектура GPT та Gemini базується на трансформерах, але має свої особливості.

GPT (Generative Pre-trained Transformer) – є однією з найвідоміших великих мовних моделей. Її архітектура складається з декількох шарів трансформерів, кожен з яких включає механізми самоуваги та повнозв'язні шари.

Мовна модель GPT генерує текст послідовно, використовуючи попередні слова як контекст для наступного слова. Це дозволяє моделі генерувати зв'язний текст, але також означає, що генерація тексту може бути повільною, оскільки кожне слово генерується окремо.

Для генерації тексту використовується авторегресивний підхід, що означає, що модель генерує текст послідовно, слово за словом, використовуючи попередньо згенеровані слова як контекст для наступного слова. Це дозволяє моделі генерувати зв'язний та контекстуально релевантний текст.

GPT має велику кількість параметрів, що дозволяє моделі зберігати та обробляти велику кількість інформації. Наприклад, GPT-4 має 1.76 трильйонів параметрів, що робить її однією з найбільших мовних моделей на сьогодні [12].

Модель переднавчається на великому наборі даних, який включає тексти з різних джерел, таких як книги, статті, вебсторінки та інші текстові ресурси. Це дозволяє моделі отримати загальні знання про мову та контекст.

Таблиця 2.1 – Основні характеристики мовних моделей GPT [14]

Модель	*Архітектура*	*Кількість параметрів*	*Навчальні дані*
GPT-1	12-рівневий 12-головий декодер (без кодера), з linear-softmax	117 мільйонів	4,5 ГБ тексту із 7000 неопублікованих книжок різних жанрів
GPT-2	Аналогічна до GPT-1, проте зі зміненою нормалізацією	1.5 мільярди	40 ГБ тексту, 8 мільйонів документів із 45 мільйонів вебсторінок, які проголосували на Reddit
GPT-3	Аналогічна до GPT-2, проте з модифікацією для більшого масштабування	175 мільярдів [13]	499 мільярдів токенів, які складаються з CommonCrawl (570 ГБ), датасету вебсторінок, англійської Вікіпедії та двох датасетів з книгами
GPT-3.5 Turbo	Нерозголошено	175 мільярдів [13]	Нерозголошено
GPT-4	Нерозголошено	Нерозголошено, оцінюється в 1.7 трильйони [12]	Нерозголошено

2.1.3 Архітектура Gemini

Gemini – є ще однією потужною великою мовною моделлю, яка використовує архітектуру трансформерів. Відмінною рисою Gemini є її здатність до багатомодального навчання, що дозволяє моделі обробляти не лише текст, але й інші типи даних, такі як зображення та аудіо. Це дозволяє моделі

інтегрувати інформацію з різних джерел та генерувати більш комплексні відповіді.

Gemini включає механізми крос-модальних зв'язків, що дозволяють моделі обмінюватися інформацією між різними модальностями. Це покращує здатність моделі до інтеграції та аналізу різних типів даних.

Також Gemini використовує удосконалені механізми уваги, які дозволяють моделі ефективніше обробляти великі обсяги даних та знижувати обчислювальні витрати.

Модель Gemini складається з двох основних частин (рис. 2.3). Перший компонент fk(·) використовує локальну інформацію кожного класу, один рівень потоку [15,16] на клас. Потім глобально пов’язаний компонент g(·) намагається використати глобальну інформацію локальних представлень шляхом спільного використання деяких параметрів між різними класами, створюючи унікальне представлення даних і, таким чином, уникаючи жорстких обмежень рівня класифікації, наприклад, крос-ентропії.

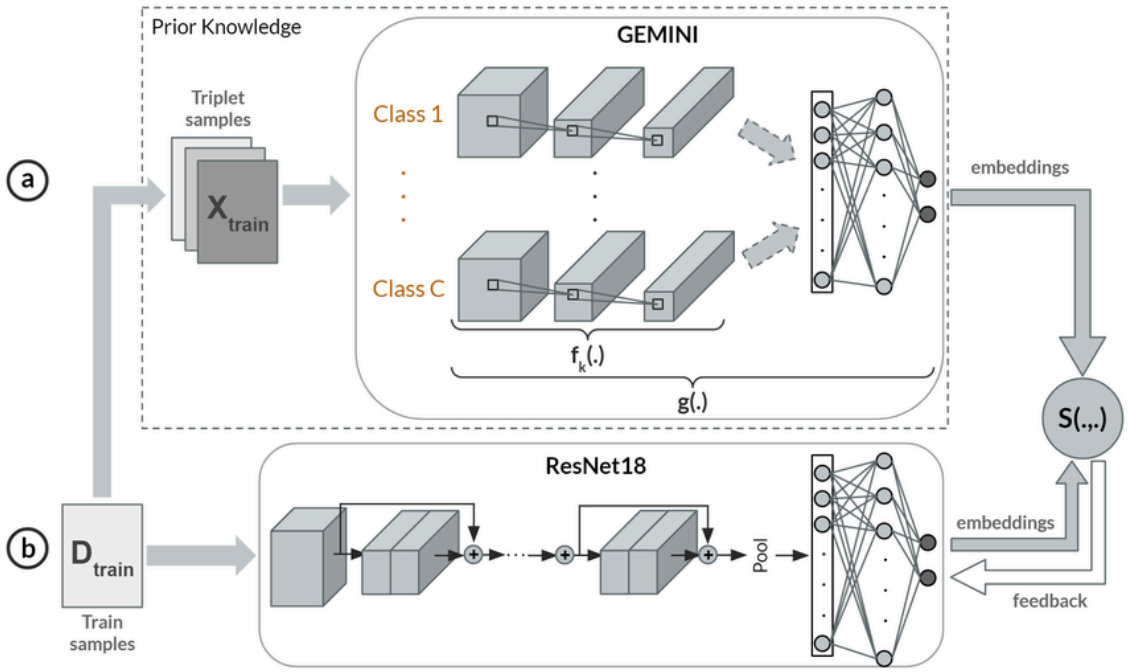

Рисунок 2.3 – Архітектура моделі Gemini.

Модель проходить навчання за допомогою триплетних зразків даних X_{train}, створених на основі початкового набору даних D_{train}. Після цього ми отримуємо вкладення(embedding) [17] відповідного навчального набору даних D_{train} з утвореного скороченого простору гіпотез. Низьковимірні вкладення Gemini спрямовують навчання ResNet через міру подібності $S(\cdot,\cdot)$ порівнюючи відстані між відповідними вкладеннями. Розбіжність забезпечує зворотний зв'язок для параметрів ResNet.

2.1.4. Переднавчання великих мовних моделей

Переднавчання – це процес попереднього навчання мовної моделі на великому корпусі тексту, що зазвичай містить мільярди слів. Цей етап сприяє формуванню у моделі глибокого розуміння структури мови, граматики та навіть деяких фактів про світ. Це можна порівняти з навчанням дитини основам мови через читання великої кількості книг, статей та вебсторінок. Модель вивчає синтаксис, семантику та загальні фрази, але може ще не розуміти конкретні технічні терміни чи знання, специфічні для предметної області.

Процес переднавчання включає кілька ключових етапів. Спочатку необхідно зібрати великий та різноманітний набір даних. Це можуть бути тексти з різних джерел, таких як книги, статті, вебсторінки та інші текстові ресурси. Зібрані дані повинні бути попередньо оброблені для видалення шуму та підготовки до навчання. Наприклад, видалення зайвих символів, нормалізацію тексту та токенізацію (розбиття тексту на токени).

Після попередньої обробки даних модель навчається на великому наборі даних. Навчання включає оптимізацію параметрів моделі за допомогою методів градієнтного спуску та інших алгоритмів оптимізації. Навчання може займати значний час та вимагати великих обчислювальних ресурсів.

Після навчання модель оцінюється на валідаційному наборі даних для перевірки її ефективності. Це дозволяє виявити можливі проблеми та налаштувати модель для покращення її продуктивності. Переднавчання великих мовних моделей включає кілька технічних аспектів, які є важливими для розуміння процесу.

Для навчання великих мовних моделей використовуються різні алгоритми оптимізації, такі як Adam (Kingma & Ba, 2014) та LAMB (You et al., 2020). Ці алгоритми дозволяють ефективно оптимізувати параметри моделі та знижувати обчислювальні витрати. Формула для оновлення параметрів у алгоритмі Adam виглядає наступним чином [18]:

$$[\theta_{t+1} = \theta_t - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}],$$

де α — швидкість навчання;

$\hat{m}_t$ та $\hat{v}_t$ – це скориговані моменти першого та другого порядку відповідно;

а ϵ – мале число для запобігання діленню на нуль.

Для запобігання перенавчанню (overfitting) використовуються методи регуляризації, такі як дроп-аут (dropout) та нормалізація (normalization) [19]. Це дозволяє моделі краще узагальнювати знання та покращувати її продуктивність на нових даних. Навчання LLM може вимагати значних обчислювальних ресурсів, тому часто використовується розподілене навчання (розподіл обчислень між кількома графічними процесорами (GPU) або обчислювальними кластерами), що дозволяє значно прискорити навчання.

2.2 Використання переднавчених моделей у технічній підтримці

Переднавчена модель може бути використана як каркас для більш специфічних рішень у сфері технічної підтримки. Тобто, базова модель, яка вже має загальні знання та розуміння мови, може бути адаптована для вирішення конкретних завдань, характерних для технічної підтримки..

Уявімо, що компанія надає технічну підтримку для програмного забезпечення. Переднавчена модель може бути донавчена (fine-tuned) на даних, що включають технічну документацію, історію запитів клієнтів, внутрішні бази знань та інші релевантні джерела. Це дозволяє моделі надавати точні та детальні відповіді на складні технічні питання, такі як налаштування програмного забезпечення, усунення помилок або оптимізація продуктивності.

Альтернативні методи використання переднавчених моделей включають Retrieval-Augmented Generation (RAG), який поєднує можливості пошуку інформації з генеративними можливостями моделі. RAG функціонує за принципом двоетапного процесу: спочатку модель здійснює пошук релевантної інформації у великій базі даних, а потім використовує цю інформацію для генерації відповіді. Це дозволяє надавати більш точні та детальні відповіді на запити клієнтів.

Таким чином, попередньо навчені моделі та їх донавчання є потужними інструментами для підвищення ефективності технічної підтримки, забезпечуючи швидке та точне вирішення запитів клієнтів. Використання таких моделей сприяє оптимізації процесів обслуговування та підвищенню рівня задоволеності клієнтів, що є стратегічно важливим для сучасних компаній.

2.3 Контекстно-орієнтована оптимізація мовних моделей

2.3.1 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) є інноваційним підходом, який поєднує можливості пошуку інформації з генеративними можливостями великих мовних моделей. Цей метод дозволяє надавати більш точні та релевантні відповіді на запити клієнтів, використовуючи актуальну інформацію з великих баз даних.

Суть методу RAG полягає в інтеграції двох основних компонентів: модуля пошуку (retriever) та модуля генерації (generator).[20] Модуль пошуку відповідає за знаходження релевантної інформації у великій базі даних, тоді як модуль генерації використовує цю інформацію для створення відповіді на запит клієнта. Такий підхід дозволяє доповнити контекст для моделі, витягуючи потрібні дані з бази знань, що значно підвищує точність та релевантність відповідей. Процес векторизації даних є ключовим етапом у роботі RAG, оскільки він дозволяє перетворювати текстові дані у вектори, які можуть бути ефективно порівняні між собою.

Векторизація включає кілька основних кроків. Спочатку текстові дані розбиваються на окремі слова або токени, що дозволяє моделі обробляти текст на рівні окремих слів або фраз. Потім кожен токен перетворюється у вектор високої розмірності за допомогою переднавченої мовної моделі. Ці моделі здатні захоплювати семантичні та синтаксичні властивості тексту, що дозволяє створювати вектори, які відображають значення та контекст слів. Вектори окремих токенів агрегуються для створення вектора, що представляє весь документ або фрагмент тексту. Це може бути зроблено за допомогою різних методів, таких як усереднення векторів або використання механізмів уваги. Вектори документів у базі даних порівнюються з вектором запиту клієнта для знаходження найбільш релевантних документів.

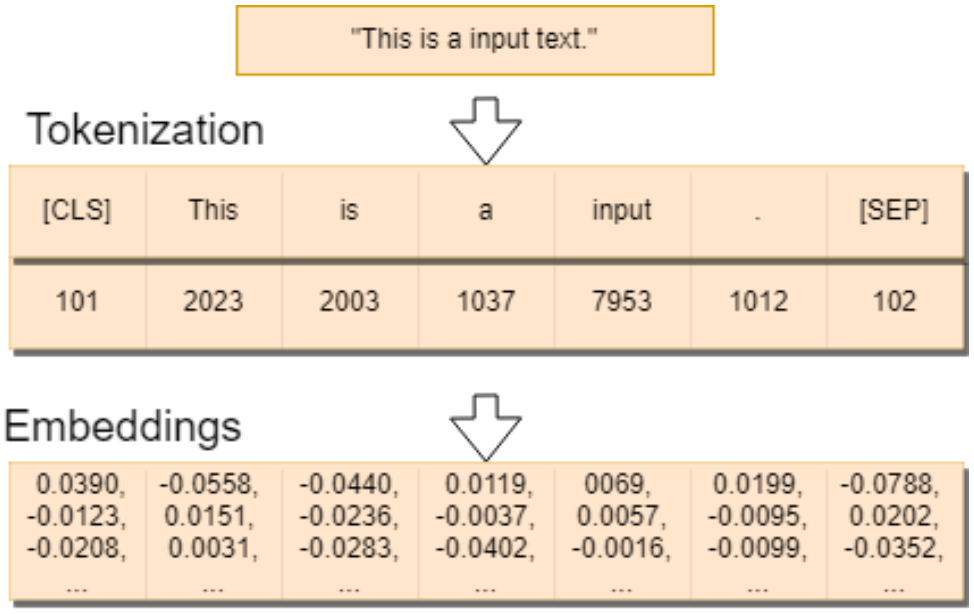

Рисунок 2.4 – Процес векторизації даних.

Це зазвичай робиться шляхом використання метрик подібності, таких як косинусна подібність або евклідова відстань. Одним з ефективних алгоритмів для порівняння векторів є FAISS (Facebook AI Similarity Search) [21]. FAISS є бібліотекою, розробленою Facebook AI Research, яка дозволяє ефективно здійснювати пошук подібних векторів у великих наборах даних. FAISS використовує різні методи для оптимізації пошуку, включаючи індексацію та зменшення розмірності. FAISS підтримує кілька типів індексів, які дозволяють оптимізувати пошук подібних векторів. Наприклад, індекси на основі к-найближчих сусідів (k-nearest neighbors, k-NN) дозволяють швидко знаходити найбільш подібні вектори до заданого вектора запиту. FAISS також підтримує індекси на основі кластеризації, такі як індекси на основі продукту квантизації (product quantization), які дозволяють зменшити розмірність векторів та знизити обчислювальні витрати. Процес пошуку подібних векторів у FAISS включає кілька етапів.

Спочатку вектори документів у базі даних індексуються за допомогою одного з підтримуваних методів індексації. Потім, коли надходить запит, його вектор порівнюється з індексованими векторами документів для знаходження

найбільш подібних векторів. Це дозволяє швидко знаходити релевантні документи, навіть у великих наборах даних.

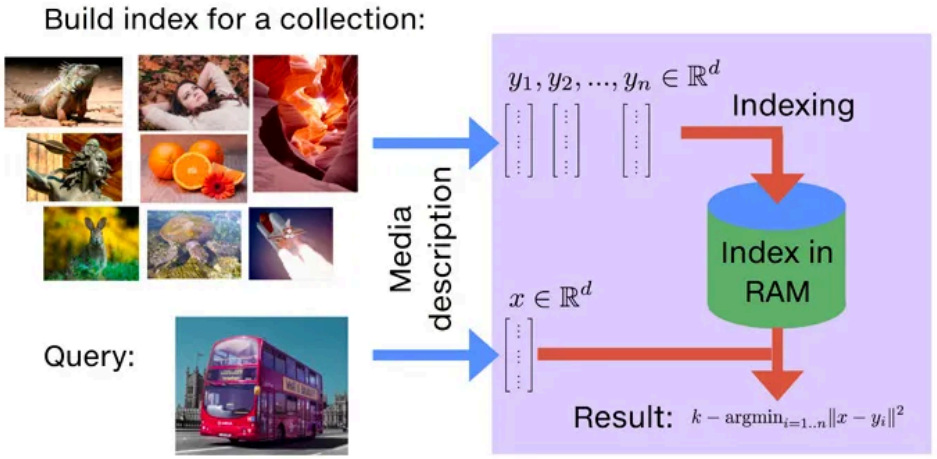

Рисунок 2.5 – Процес пошуку подібних векторів у FAISS

Загальний процес роботи RAG включає кілька етапів. Коли клієнт надсилає запит, він спочатку обробляється для токенізації та векторизації, що дозволяє створити вектор запиту, який може бути порівняний з векторами документів у базі даних. Модуль пошуку використовує вектор запиту для знаходження найбільш релевантних документів у базі даних, що включає порівняння векторів документів з вектором запиту та вибір документів з найвищою подібністю [22]. Після цього модуль генерації використовує знайдені документи для створення відповіді на запит клієнта. Це може включати інтеграцію інформації з кількох документів та генерацію зв'язного тексту, що відповідає на запит.

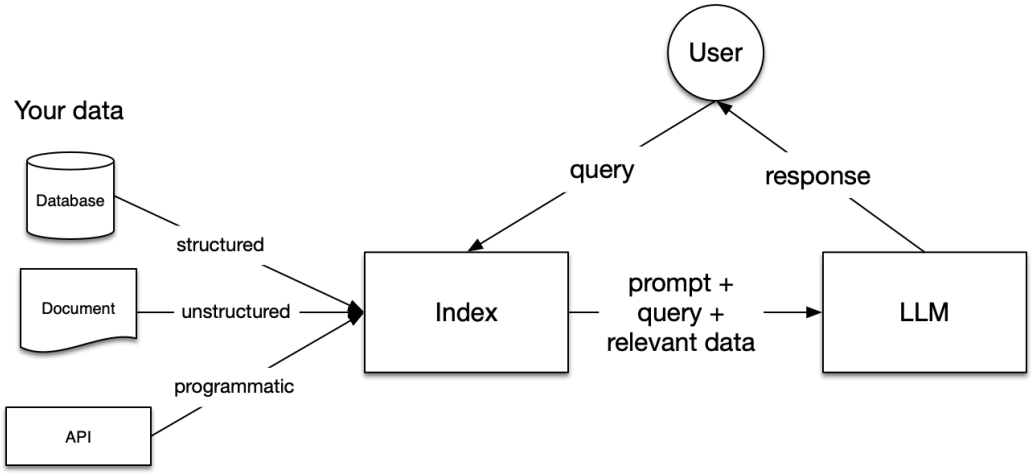

Рисунок 2.6 – Схема роботи RAG

Таким чином, RAG є потужним інструментом для підвищення ефективності технічної підтримки, забезпечуючи точні та релевантні відповіді на запити клієнтів. Використання векторизації даних та поєднання пошуку з генерацією дозволяє надавати високоякісні відповіді, що сприяє підвищенню рівня задоволеності клієнтів.

2.3.2 Донавчання (Fine-tuning)

Донавчання (fine-tuning) переднавчених великих мовних моделей є важливим етапом у процесі адаптації цих моделей до специфічних потреб технічної підтримки та консультацій. Цей процес дозволяє значно підвищити точність та релевантність відповідей, які надаються моделлю, шляхом навчання на спеціалізованих наборах даних, що відображають типові запити та проблеми клієнтів. Донавчання моделі включає кілька ключових етапів, кожен з яких має свої особливості та вимоги.

Першим етапом у процесі донавчання є збір та підготовка даних. Для того, щоб модель могла ефективно навчатися на специфічних запитах клієнтів, необхідно зібрати великий обсяг даних, що відображають реальні взаємодії між клієнтами та технічною підтримкою. Це можуть бути історії чатів, електронні листи, записи телефонних розмов та інші форми комунікації. Важливо, щоб ці дані були різноманітними та охоплювали широкий спектр запитань та проблем, з якими можуть стикатись клієнти.

Після збору даних необхідно провести їхню попередню обробку. Це включає видалення зайвих символів, нормалізацію тексту, токенізацію та інші методи підготовки даних. Наступним кроком є токенізація та векторизація тексту. (рис 2.4)

Навчання моделі включає оптимізацію її параметрів за допомогою методів градієнтного спуску та інших алгоритмів оптимізації [23]. Це дозволяє моделі навчатися на основі наданих даних та покращувати свою здатність до розуміння та генерації тексту. Важливо зазначити, що процес навчання може займати значний час та вимагати великих обчислювальних ресурсів, особливо якщо модель має велику кількість параметрів.

Після завершення навчання модель повинна бути оцінена на валідаційному наборі даних для перевірки її ефективності [24]. Це дозволяє виявити можливі проблеми та налаштувати модель для покращення її продуктивності. Оцінка моделі включає використання різних метрик, таких як точність, повнота, F1-міра та інші, що дозволяють оцінити якість відповідей, які надає модель. Важливо забезпечити, щоб модель не була перенавчена (overfitted) [25] на навчальних даних, оскільки це може призвести до погіршення її продуктивності на нових даних.

Важливо зазначити, що процес донавчання моделі не є одноразовим. Для забезпечення високої якості обслуговування клієнтів модель повинна постійно вдосконалюватися та оновлюватися на основі нових даних. Це включає регулярне збирання нових даних, їхню попередню обробку, токенізацію, векторизацію та навчання моделі. Постійне вдосконалення дозволяє моделі

адаптуватися до змін у поведінці клієнтів, нових типів запитів та інших факторів, що впливають на якість обслуговування.

Таким чином, донавчання переднавчених великих мовних моделей є складним та багатоступеневим процесом, що включає збір та підготовку даних, токенізацію та векторизацію, навчання моделі, оцінку та валідацію, інтеграцію з системами технічної підтримки, постійне вдосконалення та забезпечення безпеки даних. Використання таких моделей у сфері підтримки дозволяє підлаштувати поведінку і характер відповідей LLM під наявні стандарти. Інтеграція переднавчених LLM з системами управління знаннями, чат-ботами та іншими технологіями сприяє оптимізації процесів обслуговування та підвищенню рівня задоволеності клієнтів, що є стратегічно важливим для сучасних компаній.

2.4 Dialogflow

Dialogflow, розроблений компанією Google, є потужною платформою для створення чат-ботів та віртуальних асистентів, що використовують обробку природної мови (NLP) для взаємодії з користувачами. Ця платформа дозволяє розробникам створювати інтерактивні діалоги, які можуть відповідати на запити користувачів, виконувати дії та інтегруватися з різноманітними сервісами.

Dialogflow побудований на основі концепції агентів, які відповідають за обробку запитів користувачів. Основна ідея полягає в тому, щоб забезпечити можливість створення інтелектуальних діалогових систем, які можуть розуміти природну мову, витягувати релевантну інформацію та виконувати відповідні дії. Архітектура Dialogflow складається з кількох основних компонентів (рис. 2.7).

Одним з основних компонентів архітектури Dialogflow є агенти. Агенти є головними об'єктами у Dialogflow, що відповідають за обробку запитів

користувачів. Кожен агент може бути налаштований для виконання специфічних завдань та взаємодії з користувачами у певному контексті. Вони містять інтенції, сутності та контексти, які визначають їхню поведінку.

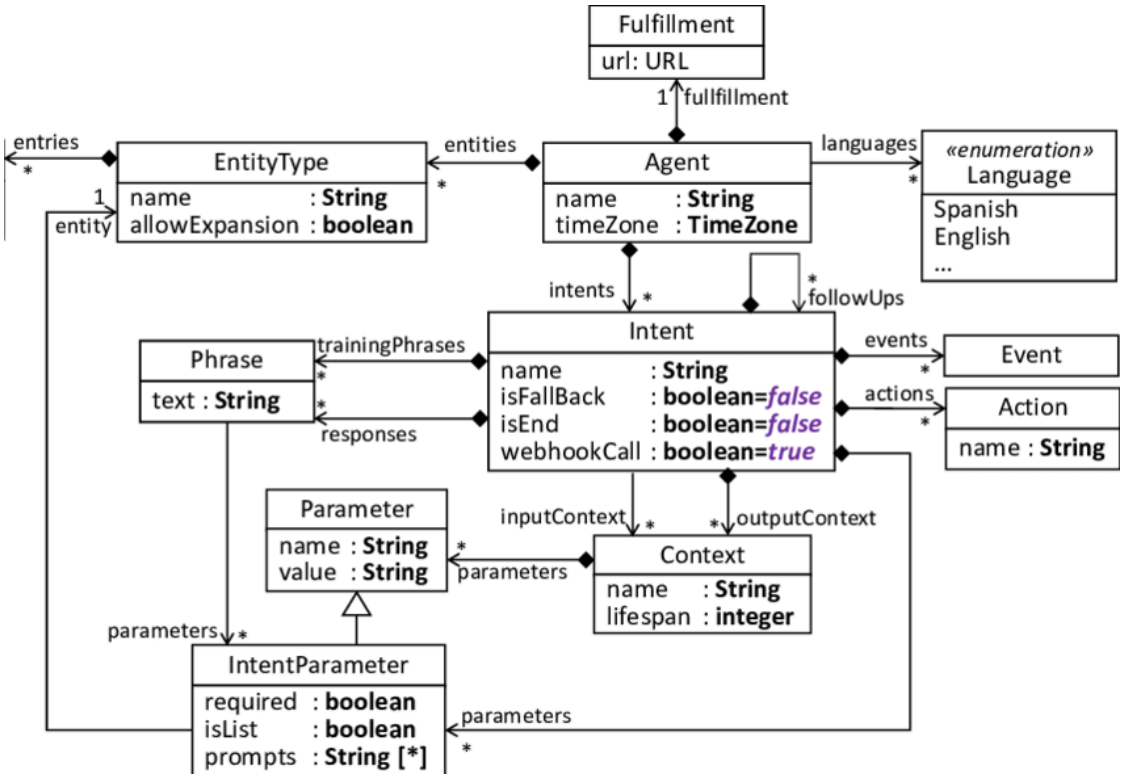

Рисунок 2.7 – Метамодель Google Dialogflow.

Інтенції визначають, як агент повинен реагувати на різні типи запитів користувачів. Кожна інтенція включає приклади фраз, які можуть бути використані для її активації, а також дії, які повинні бути виконані у відповідь на ці фрази. Інтенції також можуть містити параметри та контексти, що дозволяють агенту зберігати інформацію про поточний стан діалогу.

Сутності дозволяють агенту розпізнавати та витягувати специфічну інформацію з запитів користувачів. Це можуть бути імена, дати, місця та інші категорії даних, які є важливими для обробки запитів. Сутності можуть бути системними (вбудованими) або користувацькими (створеними розробником).

Контексти дозволяють агенту зберігати інформацію про поточний стан діалогу та використовувати її для обробки наступних запитів. Це дозволяє

створювати складніші та інтерактивні діалоги. Контексти можуть бути вхідними або вихідними та визначаються для кожної інтенції.

Dialogflow має широкий спектр застосувань у різних галузях, що робить його універсальним інструментом для створення інтерактивних діалогових систем. У контексті технічної підтримки Dialogflow може бути використаний для автоматизації обробки запитів клієнтів у службах технічної підтримки. Агенти можуть відповідати на типові питання, надавати інструкції щодо усунення несправностей та маршрутизувати складні запити до відповідних спеціалістів. Використання вебхуків дозволяє агентам отримувати доступ до баз знань компанії та надавати точні відповіді на основі актуальної інформації.

2.5 LangChain

LangChain був заснований у 2022 році співзасновниками Гаррісоном Чейзом (Harrison Chase) та Анкушем Гола (Ankush Gola) у 2022 році як проєкт з відкритим вихідним кодом, швидко здобувши визнання завдяки своїй здатності синтезувати LLM з різноманітними екзогенними компонентами. LangChain дозволяє програмістам інтегрувати потужні LLM, такі як GPT-3.5 та GPT-4 від OpenAI, з численними зовнішніми джерелами даних. Це сприяє розробці передових додатків для обробки NLP, які значно підвищують ефективність та точність обробки інформації.

Фреймворк [26] орієнтований на розробників програмного забезпечення, інженерів та аналітиків даних, які володіють мовами програмування Python, JavaScript або TypeScript. Завдяки спеціалізованим пакетам LangChain для цих мов користувачі можуть впроваджувати інноваційні рішення у свої проєкти, використовуючи новітні досягнення у сфері LLM. Це відкриває нові горизонти для створення високоефективних NLP-додатків, які можуть революціонізувати обробку та аналіз природної мови.

LangChain пропонує ряд компонентів [27], які дають змогу розробникам створювати складні LLM-додатки.

Шаблони підказок відіграють ключову роль у формуванні запитів до мовних моделей. Вони використовуються для перетворення сирих користувацьких запитів на підготовлені запити, які можуть бути ефективно оброблені моделями. Це дозволяє точніше та ефективніше передавати наміри користувача до LLM, забезпечуючи високу якість відповідей.

LangChain підтримує інтеграцію як з комерційними моделями, доступними через API (наприклад, OpenAI), так і з локальними моделями з відкритим вихідним кодом. Використання різних моделей дозволяє розробникам вибирати найбільш відповідні інструменти для конкретних задач, забезпечуючи гнучкість і масштабованість додатків.

Інструменти (Tools) – компонент LangChain, який розширює можливості системи шляхом додавання спеціалізованих функцій. Вони дозволяють агентам виконувати конкретні завдання, такі як пошук інформації або взаємодія з зовнішніми джерелами даних. Інструменти забезпечують гнучкість та адаптивність додатків, дозволяючи інтегрувати різні джерела даних та сервіси.

Ланцюги (Chains) є ключовим компонентом LangChain, що дозволяє створювати послідовність викликів, які комбінують різні компоненти для виконання комплексних обчислювальних задач. Ланцюги забезпечують модульність і гнучкість, дозволяючи інтегрувати різні моделі, алгоритми та джерела даних у єдину обчислювальну структуру. Це дозволяє оптимізувати процеси обробки даних та забезпечувати адаптацію ланцюгів до специфічних потреб користувача.

Для кращого розуміння можливостей фреймворку, розглянемо приклад з чатботом який був реалізований [28] за допомогою LangChain (рис 2.8.). Основна ідея цього чатботу полягає в тому, щоб, отримуючи різні типи запитів від користувачів, динамічно визначати найвідповідніший алгоритм або експерта для їх обробки, а потім зберігати результати для подальшого аналізу. Застосунок здатний обробляти широкий спектр запитів, включаючи програмування, поезію,

юридичні консультації та інше, використовуючи різні послідовні та паралельні ланцюги обробки даних.

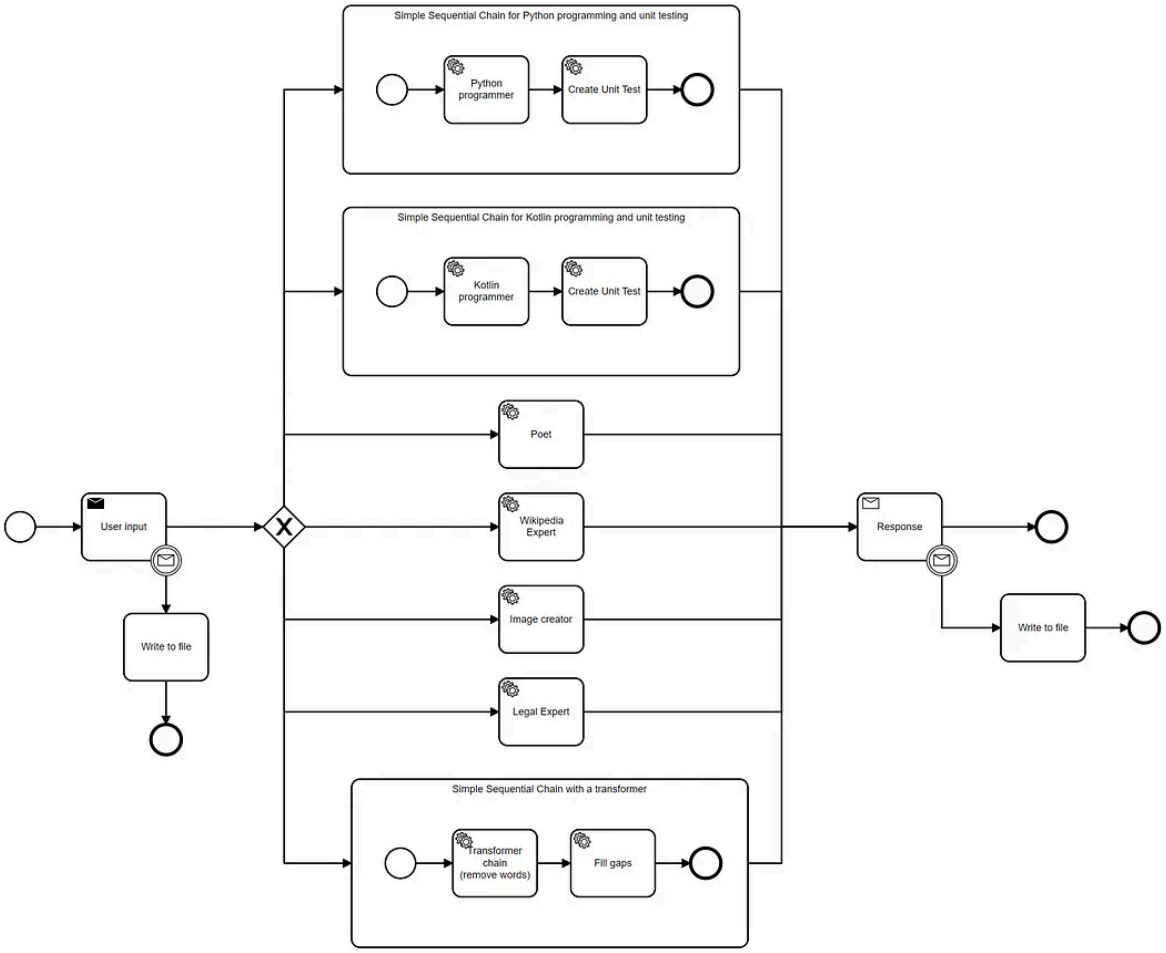

Рисунок 2.8 – Приклад чатбота на основі LangChain.

Алгоритм дій: Користувач вводить своє запитання, після чого вхідні дані записуються у файл для збереження та подальшого аналізу. Система використовує маршрутизатор для вибору найбільш відповідного ланцюга обробки, враховуючи тип запиту. Маршрутизатор може обрати один з семи ланцюгів: Python програміст, Kotlin програміст, поет, експерт з Вікіпедії, графічний художник, юридичний експерт або заповнювач пробілів у словах. Обрана модель генерує відповідь, яка потім записується у файл для збереження.

Висновки до розділу 2

Другий розділ був присвячений ролі великих мовних моделей (LLM) у сучасних інформаційних системах та їхньому застосуванню у сфері технічної підтримки та консультацій. Було розглянуто використання архітектури трансформерів, яка є основою великих мовних моделей. Завдяки механізмам самоуваги та мультиголової уваги трансформери демонструють високу ефективність в обробці природної мови.

Моделі, такі як GPT та Gemini, показали свою здатність генерувати високоякісний текст завдяки попередньому навчанню на різноманітних текстових даних. Переднавчання великих мовних моделей забезпечує модель загальним розумінням мови, тоді як донавчання на спеціально зібраних наборах даних дозволяє адаптувати моделі до специфічних завдань, таких як технічна підтримка.

Було розглянуто платформу Dialogflow, яка дозволяє створювати інтерактивні діалогові системи для обробки запитів користувачів. Використання агентів, інтенцій, сутностей та контекстів в Dialogflow сприяє автоматизації процесів технічної підтримки, надаючи інструкції щодо розв'язання проблем та маршрутизації складних запитів до відповідних спеціалістів.

LangChain, в свою чергу, є потужним інструментом, що інтегрує великі мовні моделі з зовнішніми джерелами даних та сервісами. Цей фреймворк забезпечує можливість створення складних LLM-додатків, використовуючи шаблони підказок, ланцюги, інструменти та інші компоненти. LangChain значно підвищує ефективність та точність обробки інформації, забезпечуючи адаптивність та модульність системи.

Узагальнюючи, можна зробити висновок, що використання переднавчених великих мовних моделей, їх донавчання та інші методи, такі як RAG, разом із платформами Dialogflow або LangChain, дозволяють створювати ефективні рішення для сфери технічної підтримки.

РОЗДІЛ 3 РОЗРОБКА ПРОДУКТУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Огляд і характеристика даних

Набір даних, або датасет [29], являє собою структуроване зібрання даних, організоване у спосіб, що дозволяє його ефективну обробку та аналіз. Дані у датасеті подані у вигляді таблиць або списків, де кожному елементу відповідає конкретна інформація, зазвичай у вигляді рядків і стовпців. Для подальшого аналізу ключових метрик був сформований та збережений у файл датасет «metrics.csv» (ДОДАТОК В). Датасет був створений на основі звернень у технічну підтримку, які були завантажені шляхом інкрементального вивантаження метрик з платформи обслуговування клієнтів Zendesk [30]. Датасет складається з 13 атрибутів (табл. 3.1).

Таблиця 3.1 – Атрибути даних в датасеті

timestamp	час створення звернення
response_time_first	час очікування до першої відповіді експерта
response_time_max	найбільший час очікування відповіді від експерта
response_time_avg	середній час очікування відповіді від експерта
duration	тривалість звернення
missed	чи було пропущено звернення
count_visitor	кількість повідомлень від користувача
count_agent	кількість повідомлень від експерта
rating	оцінка роботи експерта від користувача
abandon_time	час від початку звернення до виходу користувача
problem_solved	статус з яким було завершене звернення
problem_category	категорія звернення
priority	пріоритет обробки звернення
history	історія повідомлень та пов’язаних подій

В атрибуті датасету «rating» позначає оцінку відповіді експерта від користувача: позитивна оцінка – good, негативна оцінка – bad, варто врахувати, що значення атрибута «rating» є опціональним. Атрибут «abandon_time»

позначає час у секундах з моменту створення звернення до моменту, коли користувач покидає сервіс без відповіді.

Різні види сценаріїв завершення звернення представлені в атрибуті «problem_solved»: користувач покинув чат після отриманої відповіді – user_left_after_instructions, користувач підтвердив, що проблему було вирішено – solved, проблему було вирішено, проте QC спеціаліст не провів аналіз звернення – solved_no_review, не вийшло розв'язувати проблему користувача – not_solved, користувач покинув сервіс до отримання відповіді – user_left_before_instructions, недостатньо інформації від користувача – maybe, звернення користувача не відповідає критеріям технічної підтримки – not_related.

Атрибут «problem_category» містить в собі категорію запитання з яким користувач звернувся у підтримку.

Датасет містить бінарний атрибут «missed», який позначає чи пропустив експерт звернення: так – true, ні – false.

Атрибут «priority» містить значення пріоритету з яким потрібно обробити звернення у підтримку: обробити звернення в останню чергу – low, потрібно почати обробку, якщо немає звернень з вищим пріоритетом – normal, потрібно негайно почати обробку звернення – urgent.

У датасеті збережено дані з 70000 звернень у технічну підтримку. 2 колонок містять тип даних float – дійсні числа, 5 колонок типу int – цілі числа, 1 колонка типу timestamp – часова мітка, 1 колонка типу bool – бінарний тип даних, 4 колонок є рядками – string, тобто 7 цифрових колонок і 6 категоріальних колонок.

Отже, було розглянуто набір даних, над яким буде проведено дослідження для пошуку відповідності ключових метрик та рівня задоволеності клієнтів.

3.2. Візуалізація даних колонок датасету

Розглянемо детальніше атрибути та дані початкового необробленого датасету, щоб зрозуміти, яка саме інформація міститься у ньому і зробимо висновки на основі отриманої інформації.

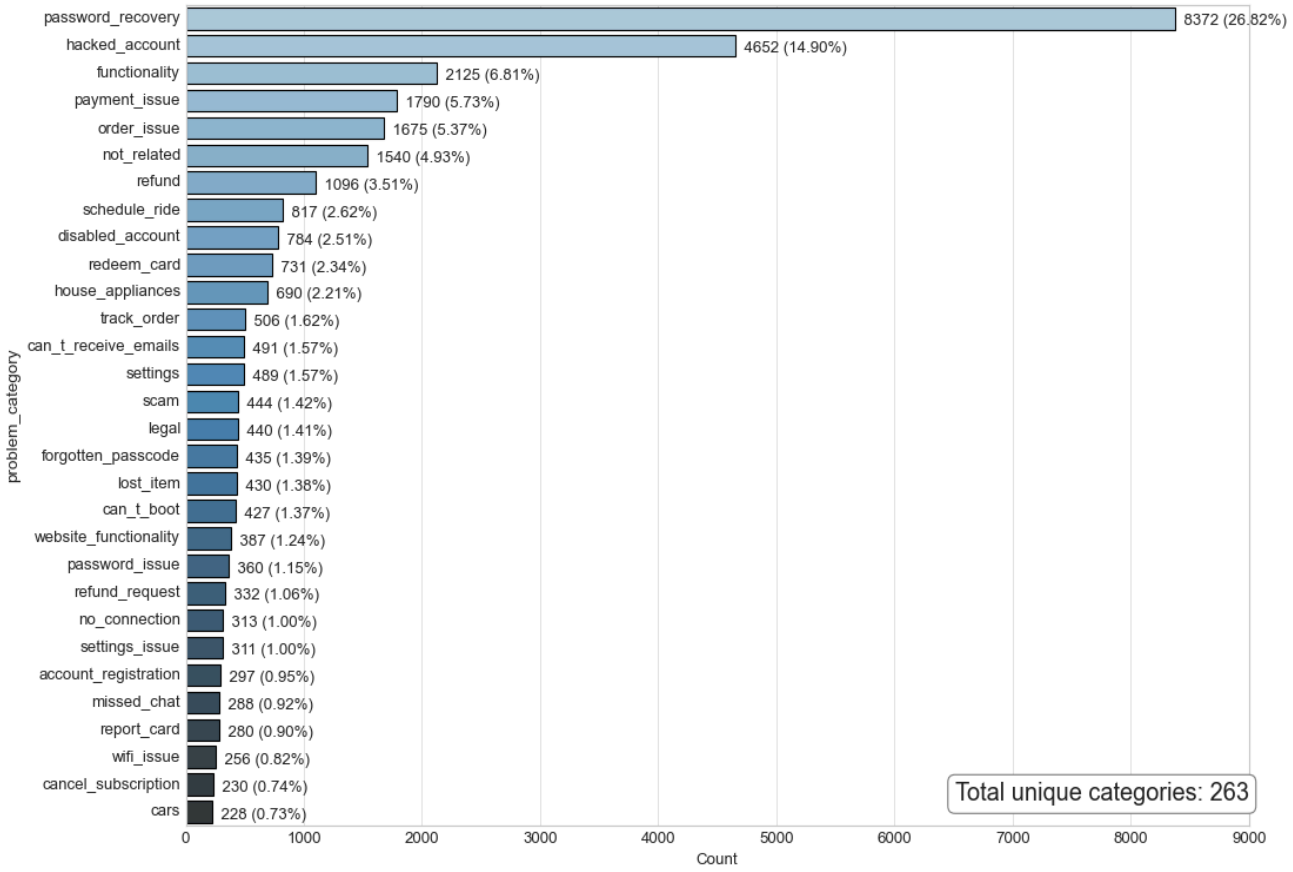

Рисунок 3.2 – Візуалізація 30 найпоширеніших значень атрибута «problem_category»

На гістограмі з розподілом значень атрибута «problem_category» (рис.3.2.) спостерігаємо, що категорія «password_recovery» переважає з абсолютним значенням у 8372 звернення, що складає 26.82% від загальної кількості звернень. Це свідчить про значну кількість випадків, коли користувачі не

можуть самостійно відновити свої паролі на сторонніх сервісах, і вказує на потенційно слабку реалізацію механізмів самостійного відновлення паролів.

Також можемо помітити, що значна кількість звернень мають категорію not_related – звернення користувача не відповідає критеріям технічної підтримки. Дізнавшись, що перевірку відповідності категорій запитів згідно з умовами сервісу здійснюють вручну технічні експерти, врахуємо цей аспект для подальшої автоматизації цього процесу.

Варто зазначити, що кількість унікальних категорій налічує 263 одиниці, що в свою чергу свідчить про широкий спектр потенційних запитів і підтверджує необхідність великої кількості даних для ефективного вирішення звернень користувачів.

Дослідимо залежність між категоріями запитів до системи технічної підтримки та відсотком їх успішного вирішення з метою виявлення закономірностей, які впливають на ефективність надання допомоги користувачам. Це дасть змогу зрозуміти, чи існують специфічні категорії запитів, що мають вищі або нижчі показники успішного вирішення, та які чинники можуть сприяти цьому.

З гістограми (рис. 3.2) можемо побачити, що «settings» (відсоток вирішення: 23.72%) і «scam» (відсоток вирішення: 23.87%) мають найвищий відсоток вирішення, що свідчить про добре налагоджені та ефективні процедури, які можливо були розроблені завдяки часто виникаючим запитам у цих областях. Особливу увагу слід звернути на «payment_issue», «password_recovery» і «hacked_account». Попри високий обсяг запитів, відсоток вирішення часто залишається проблематичним. Наприклад, «password_recovery» має низький показник успішності — 7.82%, хоча ця категорія має найбільшу кількість запитів.

Ці варіації є наслідком як технічної складності пошуку ефективного рішення, так і відмінностей в організації роботи та рівня підготовки фахівців технічної підтримки.

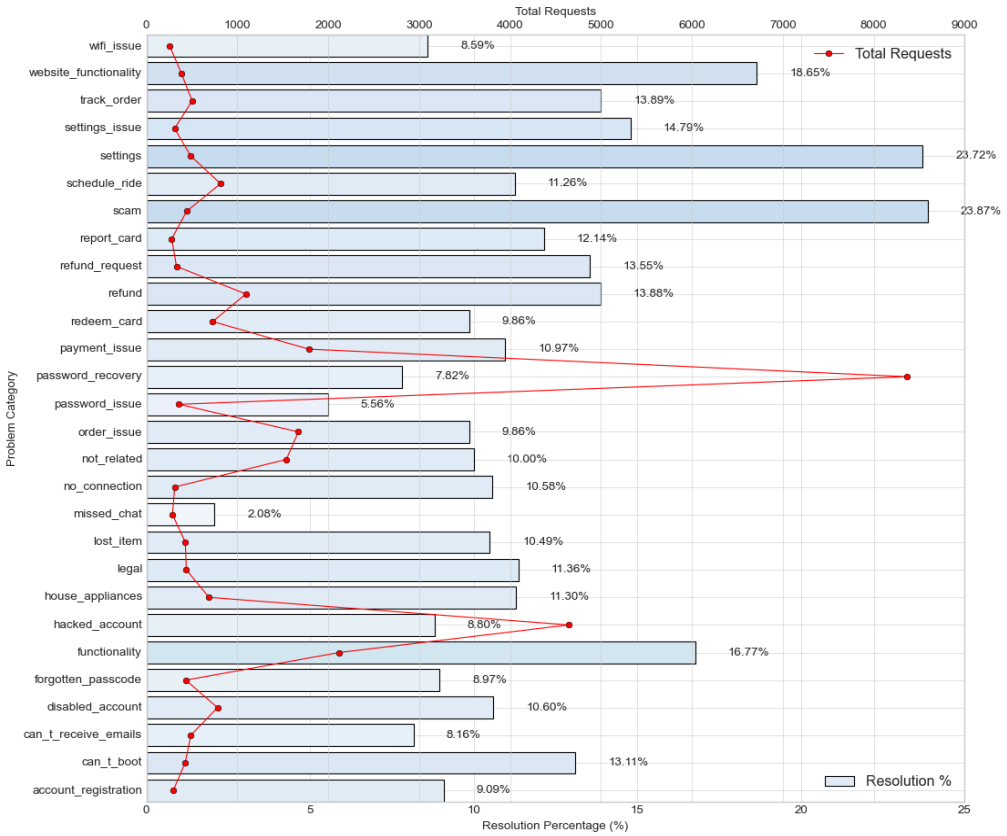

Рисунок 3.2 – Відповідність атрибута «problem_category» до відсотка успішного вирішення звернення.

Зображені розподільчі графіки (рис. 3.3.) дають змогу порівняти характеристики запитів, що отримали позитивні – good та негативні – bad оцінки. Кількість спостережень зі значенням good – 1497 записів, зі значенням bad – 420 записів.

Середні значення часу відповіді («response_time_first», «response_time_max», «response_time_avg») для позитивних та негативних оцінок є досить схожими, однак спостерігається незначне збільшення часу серед негативних оцінок:

- «response_time_first» для позитивної оцінки становить 43.83 секунд, для негативної – 46.87 секунд;
- «response_time_max» для позитивної оцінки – 100.58 секунд, для негативної – 103.11 секунд;

- «response_time_avg» становить 37.88 секунд – для позитивної та 48.10 секунд для негативної відповідно.

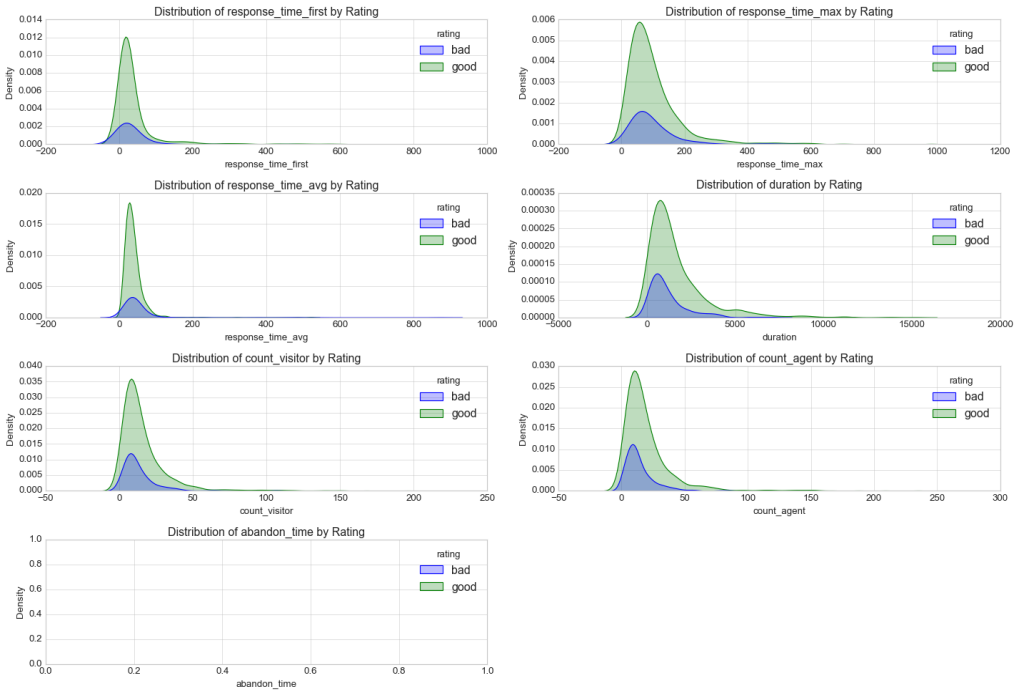

Рисунок 3.3 – Візуалізація розподілу метрик за значенням атрибута «rating»

Ці дані вказують на загальну тенденцію до уповільнення відповідей у випадках негативних оцінок.

Середня кількість повідомлень від користувача для позитивної оцінки становить 16.16, для негативної – 13.02. Середня кількість повідомлень від експерта для позитивного рейтингу становить 19.61, для негативного – 13.90. Більша кількість повідомлень від користувача та експерта при позитивній оцінці може свідчити про більш комплексний підхід до пошуку рішення, що відзначаються позитивними оцінками. Різниця між значеннями атрибута «duration» підтверджує цю гіпотезу. Тривалість обробки запиту для позитивної оцінки становить 1652.09 секунд, для негативної – 1172.96 секунд.

З діаграми співвідношення частки хороших та поганих оцінок для різних типів розв'язання проблем (рис. 3.4.) можемо побачити, що для підвищення задоволеності користувачів і зменшення кількості негативних оцінок слід

зосередитися на ефективному вирішенні запитів, оскільки саме для значень solved та solved_no_review бачимо найбільшу кореляцію з позитивною оцінкою. В свою чергу, «not_solved», «not_related», «user_left_before_instructions» найбільше корелюють з негативною оцінкою.

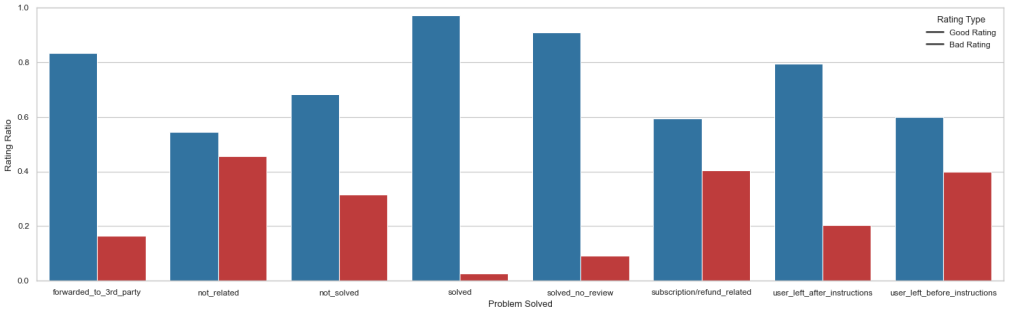

Рисунок 3.4. – Візуалізація співвідношення між атрибутами «problem_solved» та «rating»

Отже, для підвищення загального рівня задоволеності користувачів та зменшення кількості негативних оцінок слід зосередитися на кількох ключових аспектах:

- скорочення часу першої відповіді та часу обробки запитів;
- удосконалення процесу вирішення фінансових та технічних запитів за категоріями payment_issue, password_recovery та hacked_account;
- забезпечення чітких та зрозумілих інструкцій, щоб зменшити кількість негативних оцінок у категоріях user_left_after_instructions та user_left_before_instructions.

3.3. Використання донавчання для автоматизації відповідей

Для вирішення проблеми пікових навантажень, при масових несправностях є потреба в автоматизації відповідей. Для цієї задачі добре підходить велика мовна модель, проте необхідно врахувати декілька важливих аспектів:

- потрібно забезпечити, щоб модель надавала користувачам відповідні та коректні рекомендації, що відповідають їхнім запитам;
- необхідно постійно слідкувати та оцінювати ефективність роботи мовної моделі, проводити регулярний аналіз відповідей моделі, оцінку рівня задоволеності користувачів і коригування моделі відповідно до отриманих результатів.

Спробуємо використати підхід донавчання, щоб модель мала змогу відповідати на спеціалізовані запити користувачів технічної підтримки. Для цього підготуємо тренувальний та випробувальний датасети [31].

Для донавчання моделей GPT (Generative Pre-trained Transformer) кожен приклад у наборі даних повинен бути бесідою у форматі списку повідомлень, де кожне повідомлення має роль, зміст і, за опціонально ім'я. Деякі приклади тренувальних даних повинні безпосередньо вирішувати випадки, коли модель, налаштована за допомогою запиту, не поводиться належним чином. У наданих повідомленнях асистента повинні бути представлені ідеальні відповіді, які бажано отримувати від моделі після донавчання. Для донавчання GPT рекомендовано [32] надати принаймні 10 прикладів. Зазвичай розробники спостерігають явні покращення від донавчання за наявності 50-100 тренувальних прикладів для gpt-3.5-turbo.

Для підготовки тренувальних даних візьмемо значення колонки «history» з датасету «metrics.csv». Оскільки нам потрібно сформувати тренувальний набір даних у форматі бесіди користувача з експертом, потрібно буде відфільтрувати

всі значення які не є повідомленнями від користувача або експерта. Фільтрування буде відбуватись за наступними критеріями:

$$C(msg_i) = (type_i = \text{"chat.msg"}) \wedge (name_i != \text{"Customer service"})$$

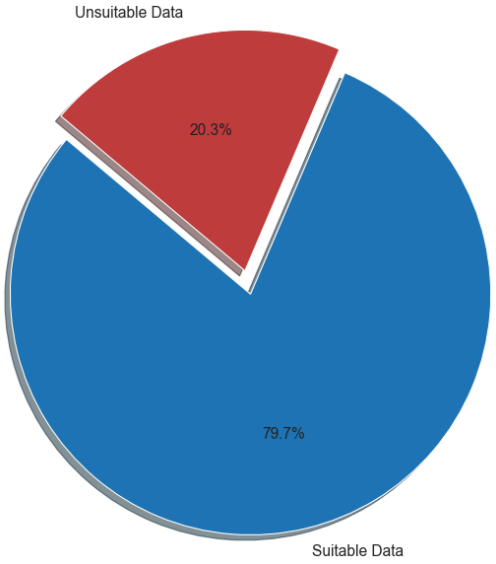

Рисунок 3.5 – Відношення придатних та непридатних даних для побудови тренувального набору даних

Відфільтрувавши лише придатні дані та розподіливши їх у пропорції, що представлені в формулах (3.1) та (3.2) отримаємо два датасети «train.jsonl» та «validation.jsonl».

$$D_{train} = 0.8 \times D \quad (3.1)$$

$$D_{validation} = 0.2 \times D \quad (3.2)$$

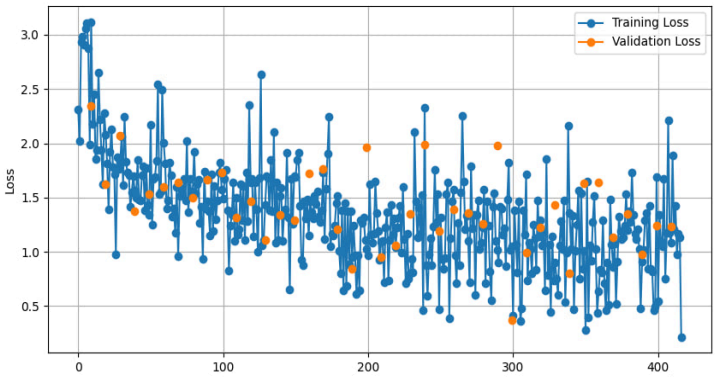

Рисунок 3.6 – Візуалізація процесу донавчання на попередньо відфільтрованих даних

Втрати в процесі навчання (сині точки) демонструє значні флуктуації протягом всього навчання. Це може бути ознакою нестабільності навчання, що вказує на проблеми з налаштуванням гіперпараметрів, таких як навчальна швидкість (learning rate).

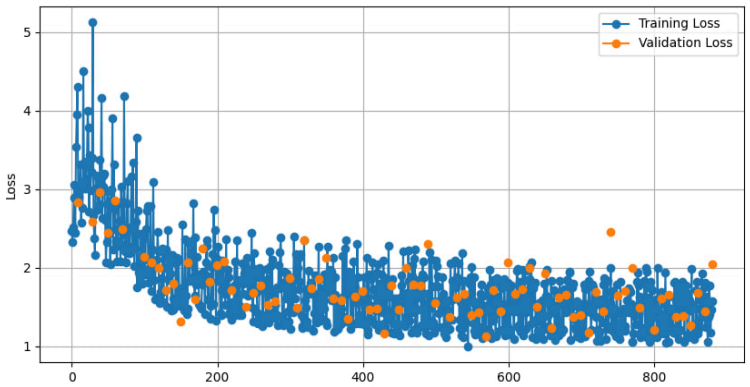

Рисунок 3.7 – Візуалізація процесу донавчання з меншим значенням learning rate

Спробуємо розв'язати цю проблему шляхом зменшення значення learning rate та збільшення кількості епох тренування. З графіка (рис.3.7.) бачимо, що флуктуації зменшились та втрати під час навчання поступово зменшуються,

проте помітно, що значення втрат під час валідації коливається навколо певного рівня. Це може бути ознакою того, що модель погано справляється з невідомими даними.

Вихідна модель, на основі якої проводився fine-tuning, базувалася на попередньо навченій LLM, яка вже мала базове розуміння характеру відповідей, проте під час подальшого навчання на нових даних виявилось, що її продуктивність на цих нових даних не є задовільною. У контексті конкретної проблеми, що виникла з тестовим датасетом, слід зазначити, що статичний датасет не може охопити всю різноманітність запитів. Сфера підтримки часто зустрічається з різноманітними та непередбачуваними запитами користувачів, які майже неможливо повністю покрити фіксованим набором даних. Навіть великий набір даних з часом втрачає актуальність. Інформація змінюється, з'являються нові факти, що не відображені у старих даних, що може призвести до неправильних відповідей моделі, що стає дуже критичним аспектом при використанні ШІ у якості технічного експерта.

3.4. Використання RAG для покращення результатів

3.4.1 Вимоги до реалізації RAG

Технологія RAG (Retrieval-Augmented Generation) пропонує ефективний підхід до покращення роботи моделей штучного інтелекту, особливо у випадках, коли статичний набір даних не може охопити всю різноманітність запитів користувачів та з часом втрачає свою актуальність (рис. 3.8). У випадку з донавчанням, вихідна модель, базована на попередньо навченій LLM, показала недостатню продуктивність під час навчання на нових даних. Використання RAG може допомогти обійти ці недоліки:

- модель з використанням RAG може динамічно отримувати інформацію з зовнішніх джерел під час обробки кожного запиту, таким чином вирішується проблема з фіксованим набором даних;
- завдяки механізму пошуку, модель може знаходити та використовувати новітню інформацію, що забезпечує актуальні та точні відповіді навіть у швидко змінюваних сферах.

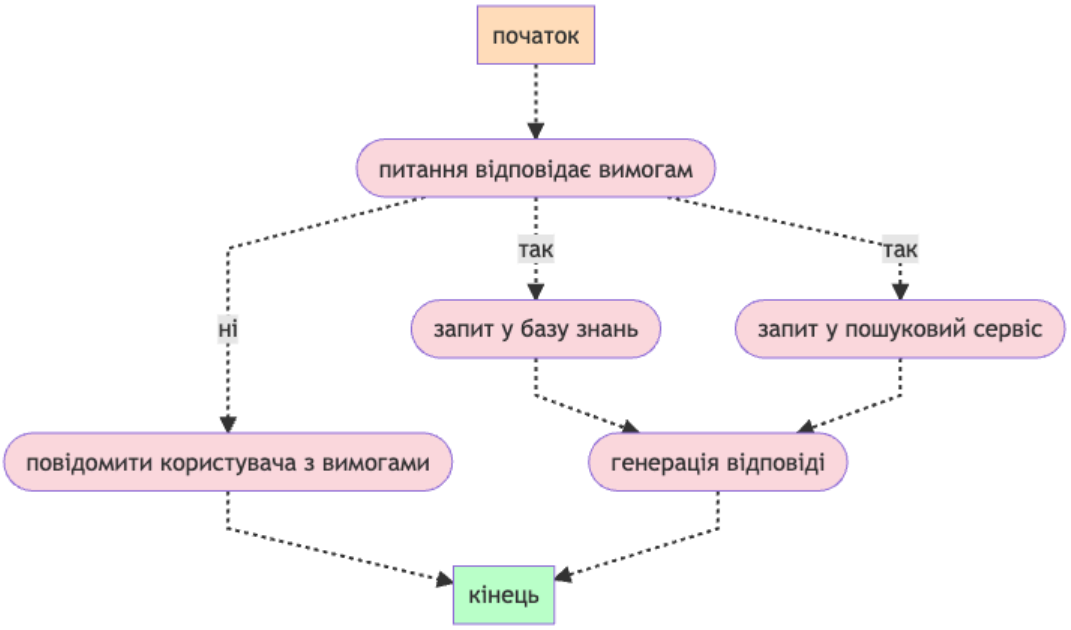

Рисунок 3.8 – Прототип алгоритму з використанням RAG.

Для забезпечення точності та надійності відповідей важливо обирати авторитетні та перевірені джерела. Бажано мати відповіді на типові запитання у базі знань експертів і регулярно їх оновлювати, щоб модель мала доступ до актуальних даних. Отримавши доступ до бази знань, її потрібно перевести у формат вкладень [17] шляхом векторизації.

Маючи доступ до актуальної бази знань, зможемо безперешкодно надавати високоякісні відповіді на типові запити. Однак виникає проблема обробки запитів, які вимагають даних, що відсутні у наявній базі знань. Цю

проблему можна вирішити шляхом звернення до відповідних реєстрів або використання пошукових систем, таких як Yahoo, Google, DuckDuckGo.

Важливо розробити алгоритм надання відповіді, який буде прозорим для легкого виявлення несправностей та стійким до таких несправностей, оскільки він включає інтеграції зі сторонніми сервісами. Також необхідно спроєктувати алгоритм відповідно до вимог бізнесу. Продумавши прототип рішення (рис. 3.8), ми можемо перейти до вибору технологій та методологій, які допоможуть побудувати якісне програмне забезпечення.

3.4.2 Проєктування автономного асистента з використанням RAG

Для забезпечення збереження та отримання даних із бази знань у векторному форматі, пропонується використання колонкової бази даних BigQuery від Google. Передусім необхідно створити проєкт в Google Cloud Platform. Після цього, створивши відповідну таблицю для зберігання векторних даних, виконаємо процес векторизації інформації з бази знань експертів. Важливо зазначити, що дані в базі знань підлягають регулярному оновленню, тому процес перезапису інформації в BigQuery має відбуватися на регулярній основі. Це завдання можна автоматизувати, використовуючи, наприклад, інструмент Cron Job [33].

Для розробки основного функціонала використаємо фреймворк LangChain, зокрема його бібліотеку LangGraph [34], яка дозволяє створювати додатки з використанням великих мовних моделей (LLM), де компоненти взаємодіють у форматі графа. Однією з основних концепцій у LangGraph є стан. Кожне виконання графа генерує стан, який передається між вузлами графа під час їх виконання. Кожен вузол оновлює цей внутрішній стан своїм результатом після виконання. Спосіб оновлення внутрішнього стану графа визначається або типом обраного графа, або спеціальною функцією.

Дуже спорідненою концепцією є концепція скінченних автоматів. Якщо розглядати цей граф з погляду скінчених автоматів, вузли незалежних агентів стають станами, а зв'язок цих агентів – це функції переходів. Скінчений автомат визначається як:

$$A = (S, X, Y, \delta, \lambda),$$

де S – кінцева множина станів автомата;

X та Y – вхідний і вихідний алфавіти;

$\delta : S \times X \rightarrow S$ – функція переходу;

$\lambda : S \times X \rightarrow Y$ – функція виходу.

Використовуючи RAG, модель може мати доступ до інформації яка не була використана під час навчання. Однак невибіркове отримання та включення фіксованої кількості отриманих уривків, незалежно від того, чи є пошук необхідним, чи уривки є релевантними, зменшує універсальність мовної моделі або може призвести до створення некорисної відповіді. На практиці було виявлено [35], що реалізація RAG вимагає логічного обґрунтування навколо цих етапів: наприклад, коли слід виконувати пошук (виходячи з питання та складу індексу), коли потрібно переформулювати питання для кращого пошуку, або коли відкидати нерелевантні знайдені документи та повторно виконувати пошук.

Введено термін саморефлексійний (Self-RAG), який охоплює ідею використання LLM для самокорекції низькоякісного пошуку та / або генерацій. Саморефлексійне отримання з доповненням генерації – це нова структура, яка покращує якість та фактичність мовної моделі за допомогою отримання інформації та саморефлексії.

Структура навчає одну довільну мовну модель, яка адаптивно здійснює пошук фрагментів на вимогу (наприклад, може виконувати пошук кілька разів під час генерації або повністю пропускати пошук), а також генерує та аналізує

знайдені фрагменти та власні генерації за допомогою спеціальних токенів, які називаються токенами рефлексії. Генерація токенів рефлексії дозволяє керувати мовною моделлю під час фази інференції, що дозволяє їй адаптувати свою поведінку до різноманітних вимог завдань.

Self-RAG дозволяє налаштовувати поведінку моделі для різноманітних завдань або уподобань, без необхідності навчання мовних моделей. Це відбувається шляхом введення токенів, які оцінюють різні аспекти якості генерації (наприклад, підтримка доказами, повнота покриття інформації).

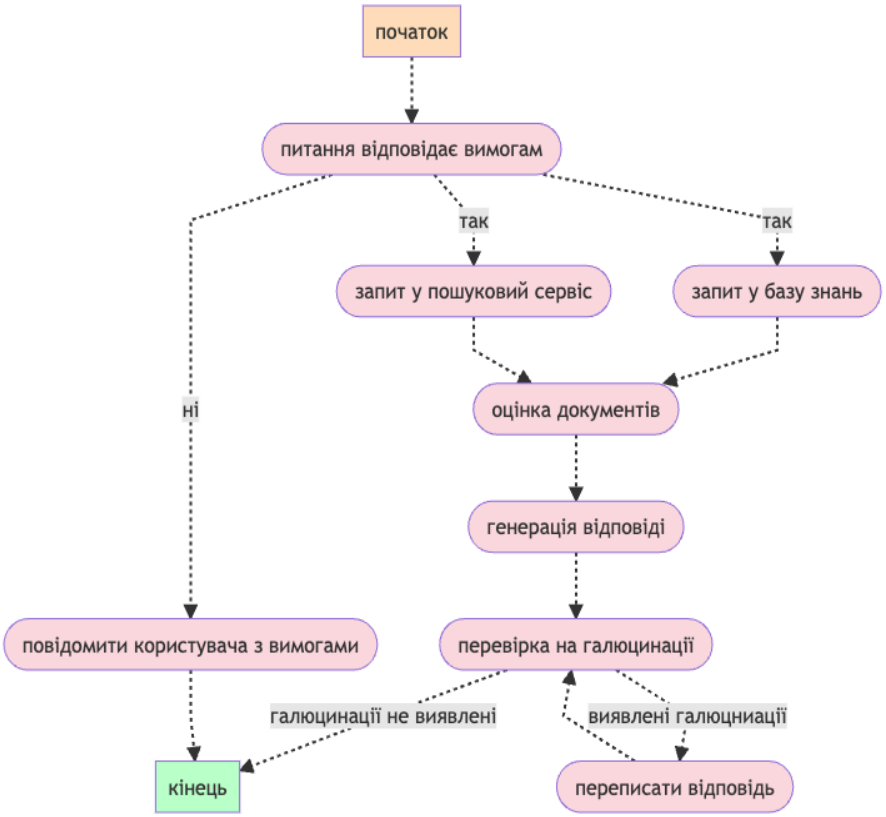

Рисунок 3.9 – Використання Self-RAG у прототипі застосунку.

Застосуємо цей підхід до нашого рішення з чат-ботом для технічних консультацій. Включення етапу оцінки документів дозволить відсіяти нерелевантні матеріали, отримані на основі ключових слів із бази знань. Це

забезпечить модель високоякісними документами для генерації відповідей, а також скоротить обсяг інформації у запиті, що робить процес більш економічним. Процес оцінки відповіді можна представити наступною формулою:

$$Assessment(Q) = S(\hat{A}, Q),$$

де Q – початкове питання, для якого потрібно згенерувати відповідь;

$(\hat{A}, Q)$ – згенерована відповідь на питання Q;

$S(\hat{A}, Q)$ – функція оцінки відповідності, яка повертає бінарний токен 'yes' або 'no', вказуючи на те, чи відповідає відповідь на питання.

Додатково було додано етап перевірки на галюцинації він сприятиме впевненості в тому, що відповідь моделі базується на фактах, отриманих із документів, а не на попередньо навчених даних. Такий підхід підвищує якість і релевантність інформації, що надається, та зменшує ризик генерації помилкових відповідей. Галюцинація [36] – феномен, коли мовна модель генерує інформацію, яка виглядає правдоподібною, але насправді є вигаданою або неправильною. Це може статися через те, що модель спирається на шаблони у даних, які вона бачила під час навчання, але не має фактичних доказів для підтвердження цієї інформації.

Для реалізації алгоритму у якості направленого графу виділимо основні вершини які будуть відповідати за конкретні інструкції в нашому графі (табл. 3.2). Також задля забезпечення розгалужень виділимо умовні ребра [37] (табл. 3.3), які будуть відповідальні за те, щоб поєднати різні вершини та побудувати шлях залежно від вхідних параметрів.

Таблиця 3.2. – Вершини в графі LangGraph

Вершина	Опис
greet	Вітання користувача. Початок спілкування.
familiarize	Ознайомити користувача з доступними категоріями запитань.
retrieve_knowledge	Отримання знань або інформації для відповіді на питання користувача.
clarify	Уточнення питання користувача або отримання додаткової інформації.
query_google	Пошук додаткової інформації за допомогою Custom Search API [43].
grade_documents	Оцінка документів або інформації, отриманої з пошуку або витягування знань.
generate_answer	Генерація відповіді на питання користувача на основі отриманих знань та інформації.
rewrite_answer	Перероблення відповіді у разі необхідності.

Таблиця 3.3. – Умовні ребра в графі LangGraph.

Назва	Опис
verify_category	Перевіряє, відповідає чи питання відповідає вказаній категорії.
decide_to_generate	Вирішує, чи генерувати відповідь на запитання, або перегенерувати запитання.
verify_urls	Перевіряє, чи містить відповідь на запитання дійсні URL-адреси.
grade_generation_v_documents_and_question	Вирішує, чи покладається генерація на документ та відповідає на запитання.
history_empty	Перевіряє, чи були повідомлення раніше у цьому зверненні.

Використовуючи вершини та ребра побудуємо результівну версію нашого графа (рис. 3.10.) проходження по якому запускатиме логіку, яка буде відповідати всім вище описаним вимогам. За допомогою додаткових перевірок, ми можемо забезпечити якість відповідей і оптимізувати логіки відповідей.

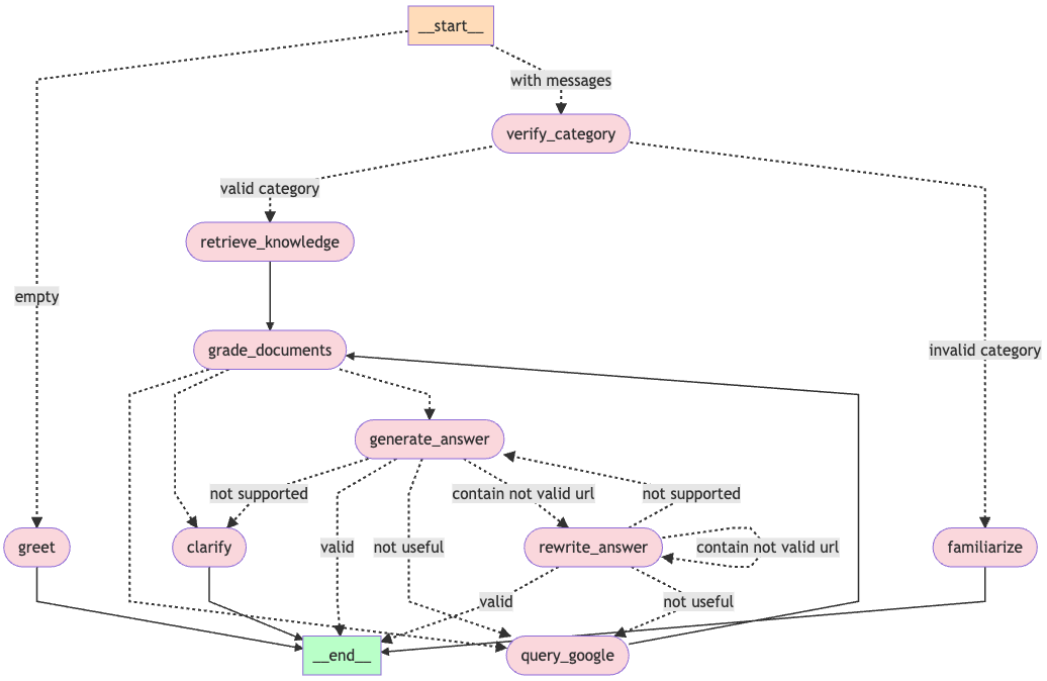

Рисунок 3.10 – Використання Self-RAG у прототипі застосунку.

Використавши підхід Self-RAG вдалось ліквідувати недоліки які були присутні з донавчанням, а також ми отримали гнучке до розширення рішення за допомогою бібліотеки LangGraph.

3.4 Інтеграція з CRM

Продумавши рішення яке зможе давати відповіді на запитання користувача, варто також подумати про інтеграцію цього рішення з Customer relationship management (CRM) системою, яка використовується для створення та обробку запитів користувачів. Zendesk [30] – CRM, якою користуються експерти, дає можливість інтеграції через Application Programming Interface (API). Для отримання інформації про звернення (ticket), можна використати

REST [38] API яке реалізує Zendesk. Для отримання інформації про повідомлення та історію чату, Zendesk пропонує використовувати Chat Conversations API [39], цей API використовує GraphQL[40] для виконання процедур та обміну інформацією.

Спроєктуємо сервіс [41], який зможе інтегруватись Zendesk за допомогою websocket з'єднання та реагувати на повідомлення сервера Zendesk, відправляючи вхідні повідомлення користувача на мовну модель з використанням RAG.

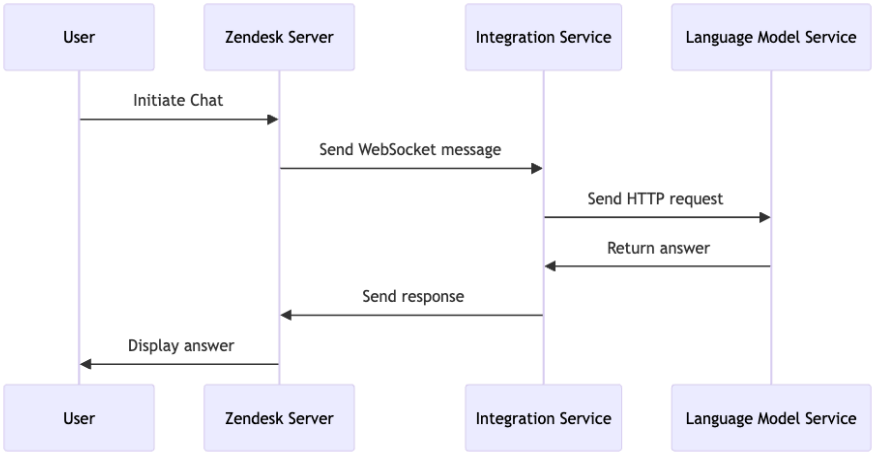

Рисунок 3.11. – Діаграма послідовності обробки повідомлення користувача.

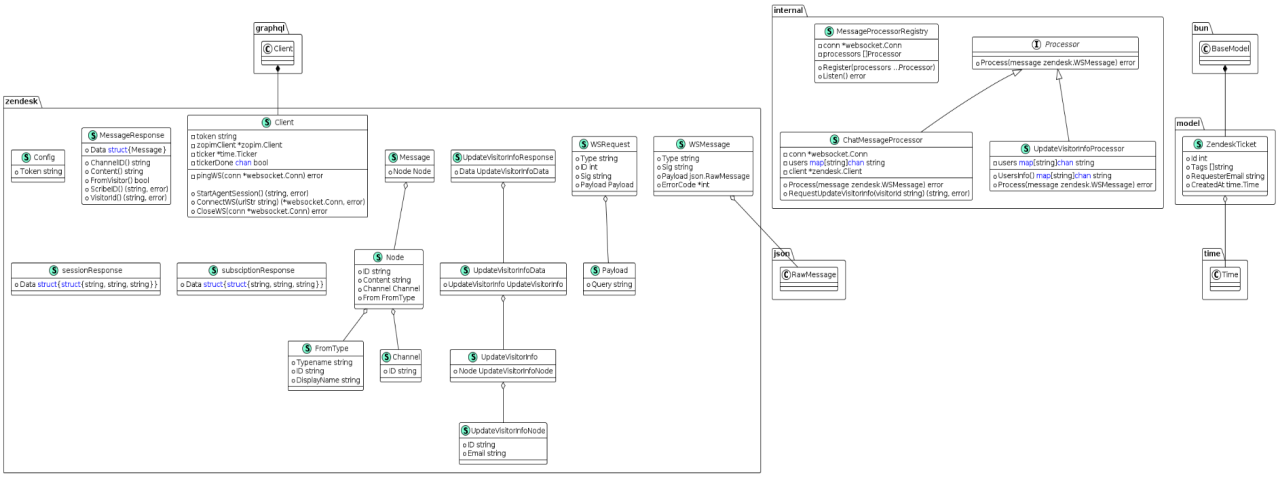

Рисунок 3.12 – UML діаграма класів.

Для збереження інформації про звернення та історії повідомлень, використаємо реляційну базу даних PostgreSQL, для цього створимо таблиці, «customers», «zopim_messages», «zendesk_chat_ticket» та «chat_history». (рис. 3.13.). Таблиця «zopim_messages» не має безпосередніх референційних зв'язків з іншими таблицями в базі даних. Проте, у ході виконання програми, сервер відправляє повідомлення з таким значенням channel_id, яке дозволяє встановити асоціативні зв'язки з даними цієї таблиці.

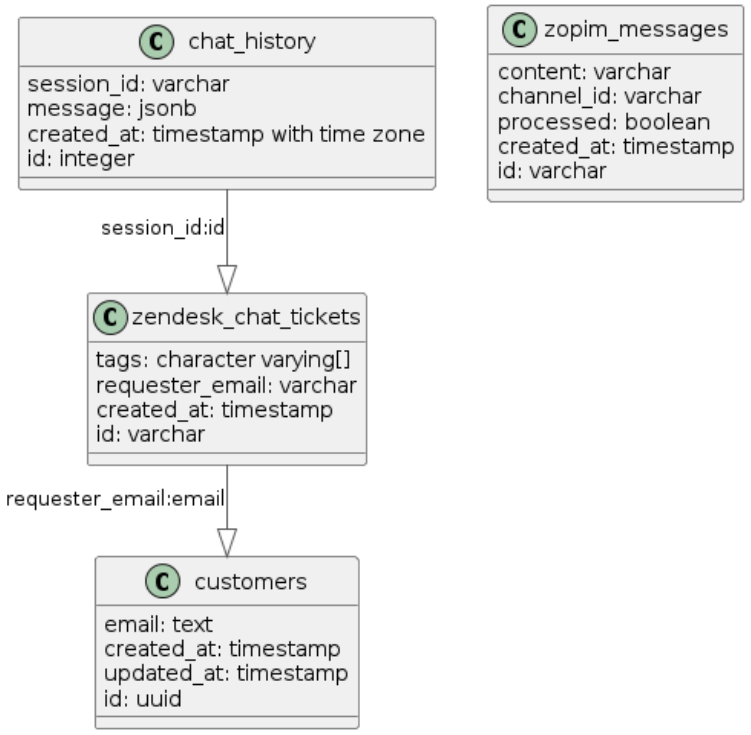

Рисунок 3.13 – Діаграма сутність-зв’язок.

3.6 Порівняння характеру відповідей

Розробивши автономного асистента для відповідей на технічні запитання, настав час дослідити процес генерації цих відповідей і визначити, наскільки вони відповідають нашим поставленим вимогам. Для цього ми розглянемо такі

аспекти: повноту та точність інформації, швидкість синтезу відповіді та загальну ефективність системи.

З конкретного прикладу діалогу між автономним асистентом та користувачем (рис. 3.14.) можемо побачити, що модель змогла уточнити необхідні деталі для вирішення проблеми користувача і надати структуровану відповідь.

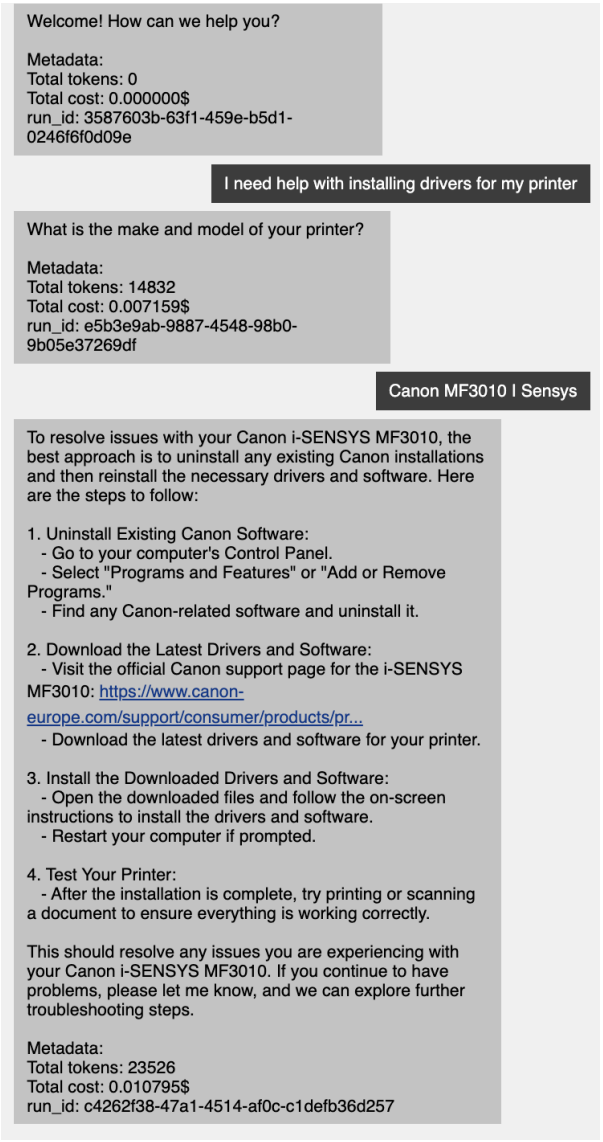

Рисунок 3.14 – Приклад діалогу користувача з автономним асистентом (повідомлення асистента зліва, користувача справа)

Завдяки всім вищеописаним етапам перевірки, кінцева відповідь була побудована на основі фактів які були знайдені у релевантних вкладеннях. Додатково представлено системну інформацію про кількість токенів (Total tokens), вартість генерації відповіді в доларах США, а також унікальний ідентифікатор (run_id), за допомогою якого можна буде покроково відстежити конкретний виклик алгоритму, знайти несправності та отримати системну інформацію включно з візуалізаціями показників системи.

На прикладі останнього повідомлення від моделі (див. рис. 3.14) проаналізуємо процес виконання графу. Порівнявши вузли графа та час їх виконання (табл. 3.4.) бачимо, що найбільш затратними по часу є вузли generate_answer та query_google. Всього на проходження графа було витрачено 38.60 секунд та 23526 токенів.

Таблиця 3.4. – Порядок виклику вузлів в графі LangGraph та їх час виконання

назва вузла	час виконання	порядковий номер
retrieve_knowledge	4.38 секунд	1
grade_documents	4.42 секунд	2
query_google	13.79 секунд	3
grade_documents	5.34 секунд	4
generate_answer	10.63 секунд	5

Бачимо, що у вузлі query_google основним обмежувальним фактором є тривалість збору та індексації даних з Custom Search API (рис. 3.15). Враховуючи значну часову затратність цієї операції, граф був оптимізований таким чином, щоб виконувати цю процедуру лише у випадку неуспішного пошуку документів в базі знань експертів.

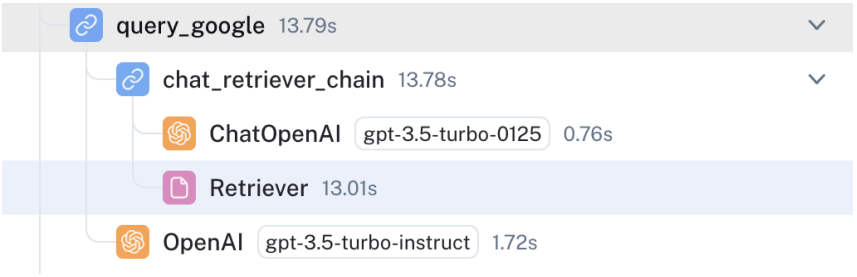

Рисунок 3.15 – Компоненти вузла query_google та їх час виконання

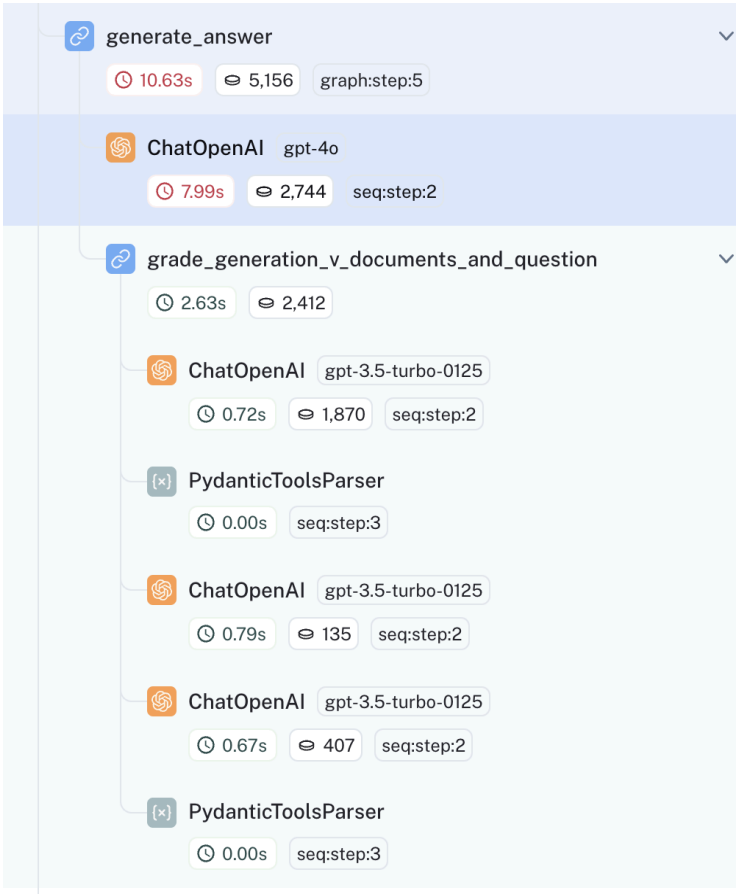

Рисунок 3.16 – Компоненти вузла generate_answer, їх час виконання та кількість використаних токенів

У вузлі generate_answer найбільше часу займає запит до мовної моделі gpt-4o, довгий час відповіді зумовлений обсягом згенерованого тексту та

пропускною здатністю (throughput) моделі [42]. Використання моделі з вищою пропускною здатністю може суттєво скоротити час генерації відповідей.

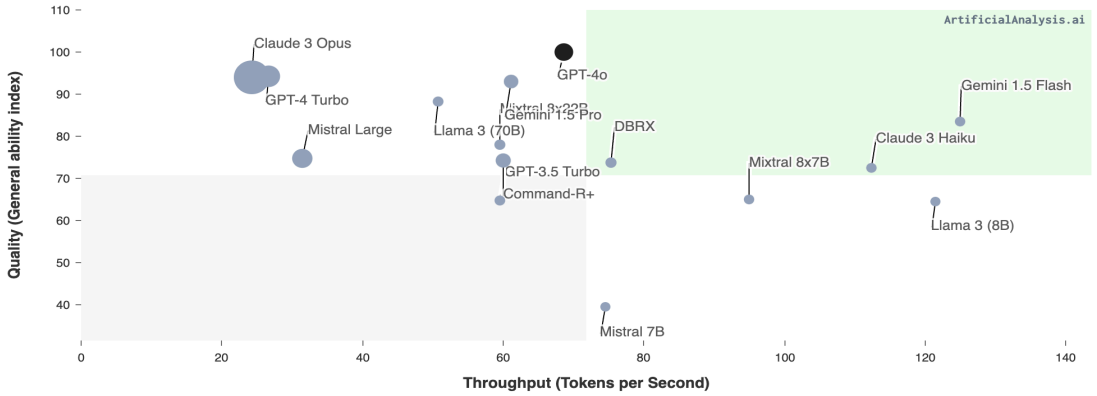

Рисунок 3.17 – Пропускна здатність великих мовних моделей[1]

З графіка (рис. 3.18.) видно, що тривалість генерації відповіді для 95-го процентиля не перевищує 75 секунд, що узгоджується з розподілом часу максимального відгуку (response_time_max) на графіку (рис. 3.3.) з датасету «metrics.csv». Час відповіді для 50-го процентиля також корелює з розподілом метрики response_time_avg.

Дослідимо які джерела (ДОДАТОК Б) були використані для генерації відповіді на запитання користувача:

- i-SENSYS MF3010 - Support - Download drivers, software and manuals - Canon Europe;
- Solved: Re: Error message 255,0,0 when trying to scan - Canon Community.

[1] Джерело зображення: https://artificialanalysis.ai/models/gpt-4o

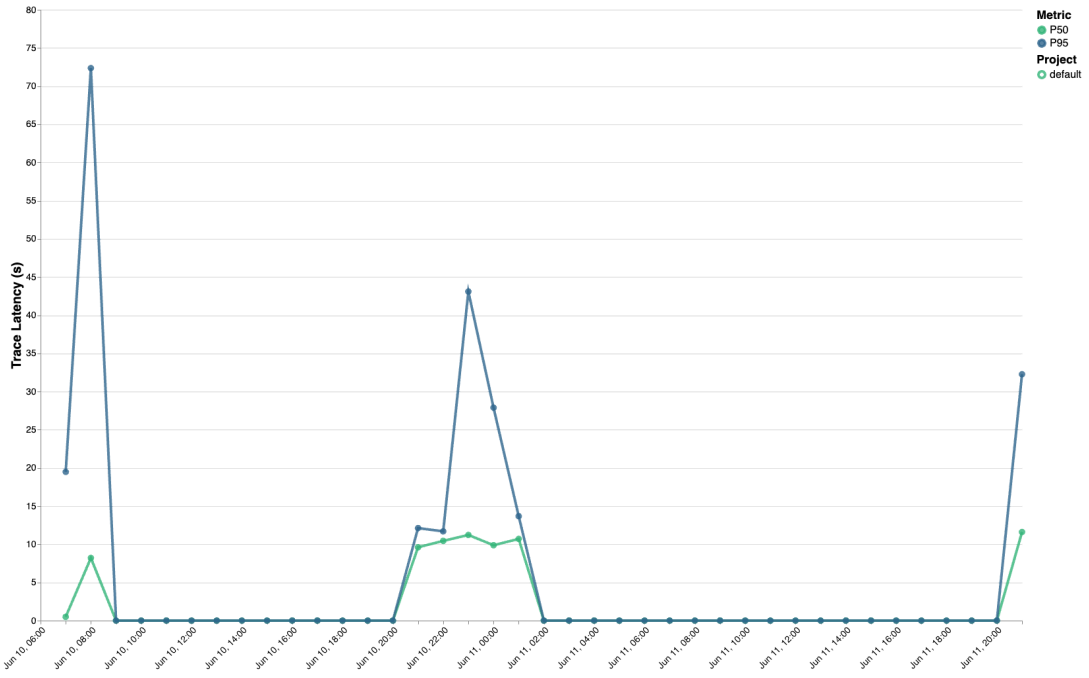

Рисунок 3.18 – Час, необхідний для генерації відповіді:
50-й та 95-й процентилі

Для створення контекстного вінка (context window) [44] були використані як офіційні сторінки вебсайту виробника, так і матеріали з форуму спільноти. Враховуючи, що офіційний ресурс є перевіреним та надійним джерелом інформації, ми можемо з високим ступенем впевненості стверджувати про якість і достовірність наданої відповіді.

Кількість токенів для генерації відповіді для 95-го досягає 36000 токенів, тоді як для 50-го процентиля – 16000 токенів (рис. 3.19.). В середньому витрачається 8511 токенів, що вказує на значний обсяг інформації в контекстному вікні.

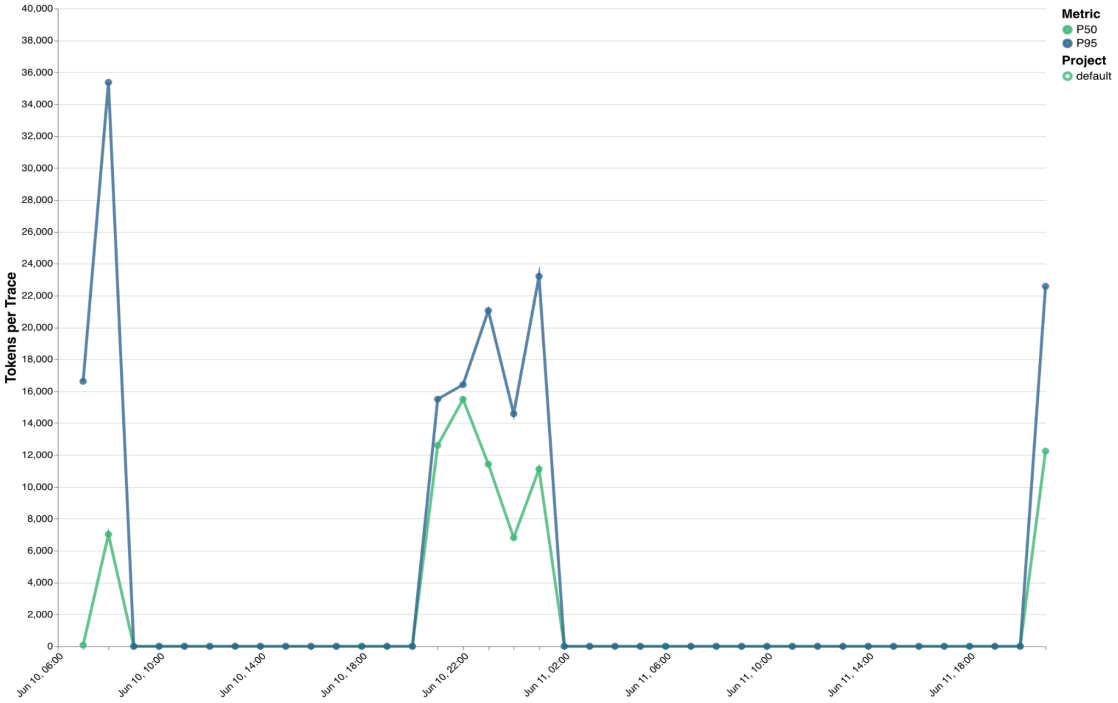

Рисунок 3.19 – Кількість токенів використаних для генерації відповіді: 50-й та 95-й процентилі.

Висновки до розділу 3

В ході дослідження було проведено ґрунтовний аналіз ключових метрик з технічної підтримки, що дозволило виявити взаємозв'язок між основними показниками та оцінкою якості обслуговування від користувачів. Було встановлено пряму залежність між тривалістю діалогу, кількістю повідомлень і позитивною оцінкою користувачів, що свідчить про їх задоволеність сервісом. Крім того, був виконаний аналіз запитань, які отримують найбільший попит від користувачів, з метою вдосконалення експерта на базі ШІ для задоволення цих потреб.

Було розглянуто підхід донавчання LLM для задачі автоматизації відповідей, де були виявлені певні обмеження та недоліки цього підходу для

конкретної задачі, як альтернативний підхід було запропоновано використання Retrieval-Augmented Generation (RAG).

Архітектура автономного асистента була спроєктована як скінчений автомат на основі орієнтованого графа. Вузли й шляхи в цьому графі представляли функції, що використовують мовні моделі для пошуку, класифікації, валідації та синтезу відповідей. Було здійснено векторизацію бази знань і розроблено механізм пошуку інформації з відкритих джерел у випадках, коли необхідна інформація відсутня в базі знань. Також був розроблений сервіс для інтеграції RAG у CRM-систему, яку використовують фахівці технічної підтримки. Аналіз показників, таких як швидкість відповіді, достовірність і інформативність, проведене на прикладі чату з ШІ-агентом, виявив кореляцію між показниками живих експертів та автономного агента на базі ШІ щодо позитивної оцінки клієнтів.

Таким чином, отримане рішення сприяє ефективному обробленню звернень користувачів у часи пікових навантажень завдяки постійній доступності й автоматизації відповідей на типові запитання, з використанням експертної бази знань та оптимізованої під специфіку технічних консультацій.

Майбутні кроки можуть включати A/B тестування на обмеженій когорті користувачів автономного асистента у якості основного експерта, де технічний фахівець буде виступати у якості оператора. Також передбачається оптимізація швидкості та якості відповідей шляхом постійного доповнення та актуалізації бази знань, що забезпечить актуальність і точність наданої інформації.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі проведено оцінку основних характеристик для майбутнього програмного продукту, що спеціалізується на дослідженні демографічного стану.

Реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі. Також буде показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього було застосовано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат шляхом ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу містить визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проєктування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проєктування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз являє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах зі стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір самої програми.

F_2 – якісний аналіз даних.

F_3 – графічні показники.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) C.

б) Python.

Функція F_2.

а) Застосування публічних бібліотек.

б) Самостійне написання алгоритмів.

Функція F_3:

а) Повна підтримка багатопоточності.

б) Обмежена підтримка багатопоточності.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

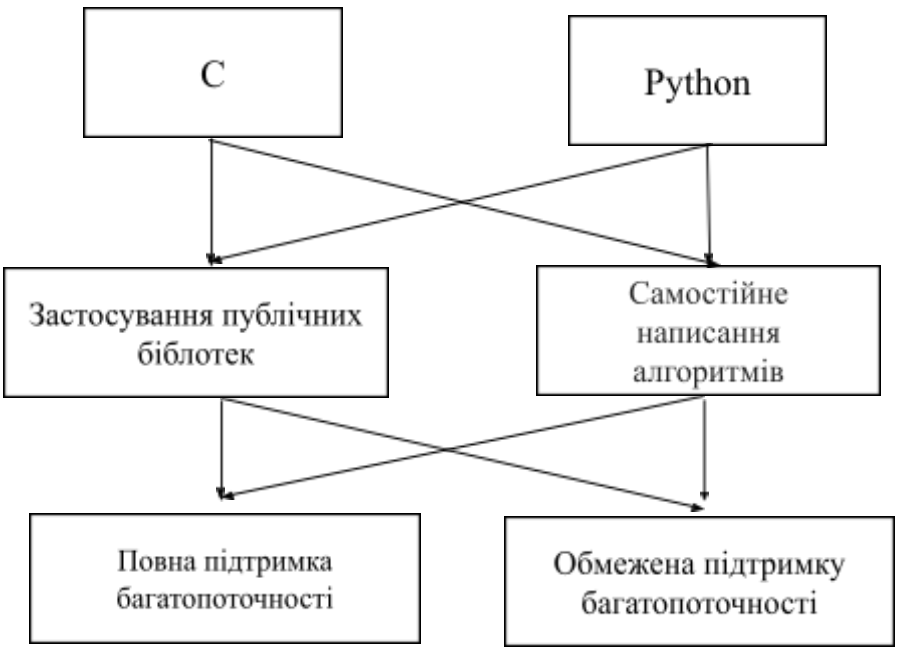

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Загальнодоступна програма, доступність багатьох бібліотек	Необхідність повної реалізації алгоритму
	Б	Доступна в реалізації програма для різних обчислень	На написання коду необхідно мати базові навички та вміння
F_2	*А*	Доступність та легкість при написанні	Іноді не відповідає задачі яку треба розв’язати
	Б	Ідеально описують усі необхідні характеристики	Достатньо затратно реалізовувати свої алгоритми для подальшої реалізації
F_3	*А*	Загально прийнята реалізація	Іноді не відповідає очікуваним значенням
	Б	При виконанні власних досліджень краще може передавати висновки	Необхідно достатньо багато часу для написання програми для побудови та знаходження всього необхідного в задачі.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто

відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу даємо загальнодоступності. Для спрощення роботи з написання коду варіант Б має бути відкинутий.

Функція F_2:

Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б.

Функція F_3:

Реалізація першого варіанту є сприйнятливою для програми. Це варіант А.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

$$F_1\text{а} - F_2\text{а} - F_3\text{а}$$

$$F_1\text{а} - F_2\text{б} - F_3\text{а}$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- $X1$ – швидкодія мови програмування;
- $X2$ – об’єм пам’яті для обчислень та збереження даних;

- *X*3 – час навчання даних;
- *X*4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у табл. 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва Параметра	*Умовні позна-чення*	*Одиниці виміру*	*Значення параметра*		
			гірші	*середні*	*кращі*
Швидкодія мови програмування	X1	оп/мс	60	80	110
Об’єм пам’яті	X2	Мб	60	50	30
Час попередньої обробки даних	X3	мс	80	70	60
Потенційний об’єм програмного коду	X4	кількість рядків коду	35	25	20

За даними табл. 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

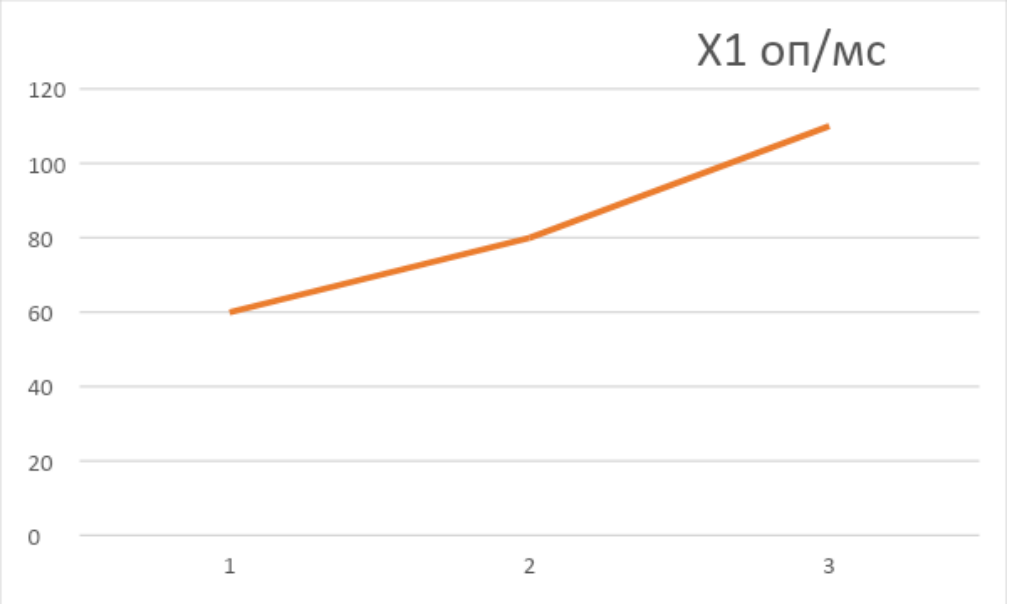

Рисунок 4.2 – X1, швидкодія мови програмування

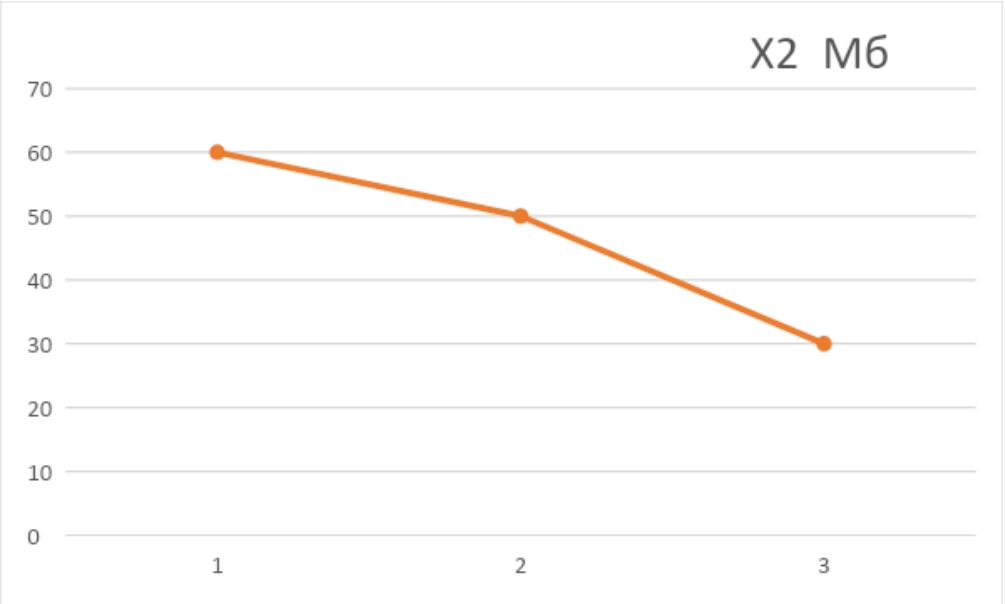

Рисунок 4.3 – X2, об’єм пам’яті

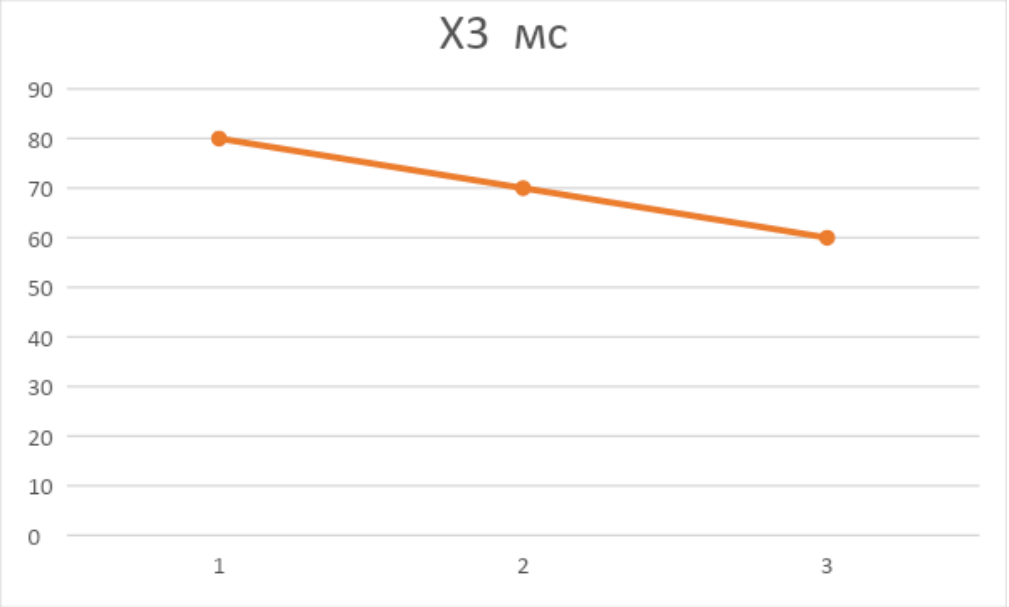

Рисунок 4.4 – X3, час попередньої обробки даних

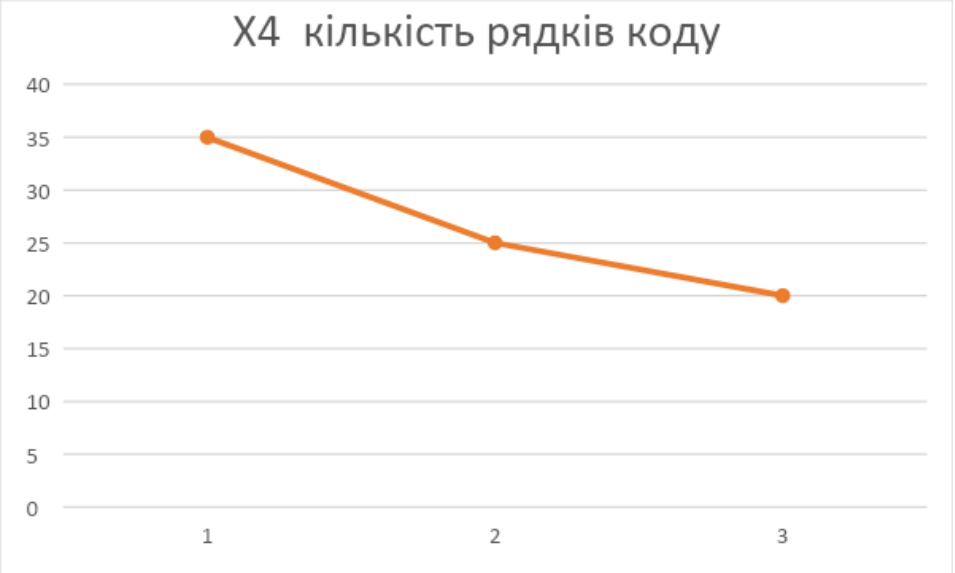

Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;

- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у табл. 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	1	2	2	1	1	1	2	10	-7,5	56,25
X2	Об’єм пам’яті	Mb	3	4	3	3	4	3	4	24	6,5	42,25
X3	Час попередньої обробки даних	мс	2	1	1	2	2	2	1	11	-6,5	42,25
X4	Потенційний об’єм програмного коду	Кількість рядків коду	4	3	4	4	3	4	3	25	7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	197

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2\left(n^3 - n\right)} = \frac{12 \cdot 197}{7^2\left(4^3 - 4\right)} = 0,754 > W_k = 0,67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у табл. 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
*X*1 і *X*2	<	<	<	<	<	<	<	<	0,5
*X*1 і *X*3	<	>	>	<	<	<	>	<	0,5
*X*1 і *X*4	<	<	<	<	<	<	<	<	0,5
*X*2 і *X*3	>	>	>	>	>	>	>	>	1,5
*X*2 і *X*4	<	>	<	<	>	<	<	<	0,5
*X*3 і *X*4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \{1.5 \text{ при } X_i > X_j \; 1.0 \text{ при } X_i = X_j \; 0.5 \text{ при } X_i < X_j$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}$$

$$b_i = \sum_{i=1}^{N} a_{ij}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'},$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з табл. 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	*Параметри* x_j				*Перша ітер.*		*Друга ітер.*		*Третя ітер*	
	*X*1	*X*2	*X*3	*X*4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
*X*1	1	0,5	0,5	0,5	2,5	0,16	9,25	0,16	34,125	0,16
*X*2	1,5	1	1,5	0,5	4,5	0,28	16,25	0,28	59,125	0,28
*X*3	1,5	0,5	1	0,5	3,5	0,22	12,25	0,21	41,875	0,2
*X*4	1,5	1,5	1,5	1	5,5	0,34	21,25	0,35	77,875	0,36
Всього:					16	1	59	1	213	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об’єм пам’яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j},$$

де n – кількість параметрів;

$K_{\text{в}i}$ – коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	*Варіант реалізації функції*	*Параметри*	*Абсолютне значення параметра*	*Бальна оцінка парамет-ра*	*Коефіцієнт вагомості параметра*	*Коефіцієнт рівня якості*
F1	А	*X*1	75	20	0,41	8,2
F3	А	*X*2	39	29	0,61	17,69
	Б	*X*3	78	14	0,22	3,08
F4	А	*X*4	90	24	0,45	10,8

За даними з табл. 4.6 за формулою:

$$K_K = K_{\text{ТУ}}\left[F_{1k}\right] + K_{\text{ТУ}}\left[F_{2k}\right] + \ldots + K_{\text{ТУ}}\left[F_{zk}\right],$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 8{,}2 + 17{,}69 + 10{,}8 = 36{,}69 ;$$
$$K_{K2} = 8{,}2 + 3{,}08 + 10{,}8 = 22{,}08 .$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти охоплюють два окремих завдання.

1. Розробка проекту програмного продукту.

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$Т_О = Т_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М},$$

де $Т_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р$ =37 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 37 \cdot 1.8 \cdot 0.9 = 59{,}94 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =29 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$T_2 = 29 \cdot 0.9 \cdot 0.8 = 20.88 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (59{,}94 + 20.88 + 4.8 + 20.88) \cdot 8 = 852 \text{ людино-годин.}$$

$$T_{II} = (59{,}94 + 20.88 + 6.91 + 20.88) \cdot 8 = 868{,}88 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 27000 грн., один аналітик в області даних з окладом 30000. Визначимо середню зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \cdot t} \text{ гр.,}$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$СЧ = \frac{27000+27000+30000}{3 \cdot 21 \cdot 8} = 166,6 \text{ грн}$$

Тоді, розрахуємо заробітну плату за формулою:

$$ЗП = С_ч \cdot Т_i \cdot КД,$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 166,6 \cdot 852 \cdot 1.2 = 170331,84$ грн.

II. $С_{ЗП} = 166,6 \cdot 868.88 \cdot 1.2 = 173706,49$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 170331,84 \cdot 0.22 = 37473$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 173706,49 \cdot 0.22 = 38215,43$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 27000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 27000 \cdot 0,2 = 64800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 64800 \cdot (1 + 0.2) = 77760 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 77760 \cdot 0,22 = 17107,2 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 10000 = 3500 \text{ грн},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.4 \cdot 10000 \cdot 0.08 = 1120 \text{ грн},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.35 =$$

$$= 627,2 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 627,2 \cdot 0,2 \cdot 0,3 \cdot 5,23 = 196,81 \text{ грн},$$

де N_C – середньо-споживча потужність приладу;

$K_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0,67 = 6700 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н,$$

$$С_{ЕКС} = 77760 + 17107,2 + 3500 + 1120 + 196,81 + 6700 = 106384,01 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 106384,01 / 627,2 = 169,62 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot T,$$

I. $С_М = 169,62 \cdot 852 = 144516,24$ грн

II. $С_М = 169,62 \cdot 868,88 = 147379,43$ грн

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67,$$

I. $С_Н = 144516,24 \cdot 0,67 = 96825,9$ грн

II. $С_Н = 147379,43 \cdot 0,67 = 98744,22$ грн

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н,$$

I. $С_{ПП} = 77760 + 17107,2 + 144516,24 + 96825,9 = 336209,34$ грн

II. $С_{ПП} = 77760 + 17107,2 + 147379,43 + 98744,22 = 340990,85$ грн

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{К j} / С_{Ф j},$$

$К_{ТЕР1} = 36,69 / 336209,34 = 10,913 \cdot 10^{-5}$,

$К_{ТЕР2} = 22,08 / 340990,85 = 6,4753 \cdot 10^{-5}$.

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 10,913 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У

нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 10,913 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- використання готових бібліотек;
- обмежена підтримка багатопоточності.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

Висновки до четвертого розділу

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

ПЕРЕЛІК ПОСИЛАНЬ

1. Common Customer Support Challenges. (2024). URL: https://devrev.ai/blog/common-customer-support-challenges (дата звернення 28.04.2024)
2. Brandtzaeg P. B., Følstad A. Why People Use Chatbots. SpringerLink International Conference on Internet Science, 2017, 10673: 377-392
3. Ariyaluran Habeeb, Riyaz Ahamed & Nasaruddin, Fariza & Gani, Abdullah & Hashem, Ibrahim & Ahmed, Ejaz & Imran, Muhammad. (2018). Real-time big data processing for anomaly detection: A Survey. International Journal of Information Management. 45. 10.1016/j.ijinfomgt.2018.08.006.
4. Zhou, Xin. (2023). Chatbot Improves the Customer Service in Four Important Industries. Highlights in Science, Engineering and Technology. 39. 339-346. 10.54097/hset.v39i.6551.
5. Mittal Amit, Agrawal Ayushi, Chouksey Ayushi, Shriwas Rachna, Agrawal Saloni. A Comparative Study of Chatbots and Humans. International Journal of Advanced Research in Computer and Communication Engineering, 2016, 5(3): 1055-1057.
6. Cheng Xusen, Bao Ying, Zarifis Alex, Gong Wangkun, Mou Jian. Exploring consumers' response to text-based chatbots in e-commerce: the moderating role of task complexity and chatbot disclosure. Emerald Insight Internet Research, 2021, 32(2)
7. Cui Lei, Huang Shaohan, Wei Furu, Tan Chuangqi, Duan Chaoqun, Zhou Ming. SuperAgent: A Customer Service Chatbot for E-commerce Websites. Association for Computational Linguistics Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics-System Demonstrations, 2017, 97-102
8. Saeed, M.G, Jasim, Y.A. and Raewf, M.B., 2022. Analyzing Social Media Sentiment: Twitter as a Case Study. ADCAIJ: Advances in

Distributed Computing and Artificial Intelligence Journal, 11(4), pp.427-450.

9. Tawfeeq, Tawfeeq & Al Awqati, Ali & Jasim, Yaser. (2023). The Ethical Implications of ChatGPT AI Chatbot: A Review. 49-56.
10. Regulation (EU) 2016/679 of the European Parliament and of the Council. URL: https://ogdpr.eu/en/gdpr-2016-679 (дата звернення 27.04.2024)
11. Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia. (2017). Attention Is All You Need.
12. Schreiner, Maximilian (July 11, 2023). "GPT-4 architecture, datasets, costs and more leaked". THE DECODER. Archived from the original on July 12, 2023. Retrieved July 12, 2023.
13. Ver Meer, Dave (June 1, 2023). "ChatGPT Statistics". NamePepper. Retrieved 2023-06-09.
14. GPT-1 to GPT-4. Each of OpenAI's GPT Models Explained and Compared. URL: https://www.makeuseof.com/gpt-models-explained-and-compared/ (дата звернення 01.05.2024)
15. Javad Abbasi Aghamaleki and Vahid Ashkani Chenarlogh.Multi-stream cnn for facial expression recognition in lim-ited training data. Multimedia Tools and Applications,78(16):22861–22882, 2019.
16. Vahid Ashkani Chenarlogh, Farbod Razzazi, and NajmehMohammadyahya. A multi-view human action recognition system in limited data case using multi-stream cnn. IEEE, 2019
17. Li, Saihan & Gong, Bing. (2021). Word embedding and text classification based on deep learning methods. MATEC Web of Conferences. 336. 06022. 10.1051/matecconf/202133606022.
18. The Math Behind the Adam Optimizer https://towardsdatascience.com/the-math-behind-adam-optimizer-c41407efe59b (дата звернення 01.05.2024)

19. Garbin, Christian & Zhu, Xingquan & Marques, Oge. (2020). Dropout vs. batch normalization: an empirical study of their impact to deep learning. Multimedia Tools and Applications. 79. 1-39. 10.1007/s11042-019-08453-9.
20. Zhao, Penghao & Zhang, Hailin & Yu, Qinhan & Wang, Zhengren & Geng, Yunteng & Fu, Fangcheng & Yang, Ling & Zhang, Wentao & Cui, Bin. (2024). Retrieval-Augmented Generation for AI-Generated Content: A Survey.
21. FAISS library for efficient similarity search and clustering of dense vectors. URL: https://github.com/facebookresearch/faiss (дата звернення 26.05.2024)
22. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" (Lewis et al., 2020).
23. Fu, Zihao & Yang, Haoran & So, Anthony & Lam, Wai & Bing, Lidong & Collier, Nigel. (2023). On the Effectiveness of Parameter-Efficient Fine-Tuning. Proceedings of the AAAI Conference on Artificial Intelligence. 37. 12799-12807. 10.1609/aaai.v37i11.26505.
24. Lalor, John & Wu, Hao & Yu, Hong. (2017). Improving Machine Learning Ability with Fine-Tuning.
25. Roth, Benjamin & Araujo, Pedro & Xia, Yuxi & Kaltenbrunner, Saskia & Korab, Christoph. (2024). Specification Overfitting in Artificial Intelligence. 10.21203/rs.3.rs-4106577/v1.
26. Partelow, Stefan. (2023). What is a framework? Understanding their purpose, value, development and use. Journal of Environmental Studies and Sciences. 13. 10.1007/s13412-023-00833-w.
27. LangChain Core Components. URL: https://python.langchain.com/v0.1/docs/integrations/components/ (дата звернення 18.05.2024)
28. Complex Chains with LangChain. URL: https://medium.com/@gil.fernandes/complex-chains-with-langchain-bee7e4f15c03 (дата звернення 31.05.2024)

29. Renear, Allen & Sacchi, Simone & Wickett, Karen. (2010). Definitions of dataset in the scientific and technical literature. Proceedings of the American Society for Information Science and Technology. 47. 1 - 4. 10.1002/meet.14504701240.
30. Zendesk Customer Service Software. URL: https://www.zendesk.com/service/ (дата звернення 31.05.2024)
31. Uçar, Muhammed & Nour, Majid & Sindi, Hatem & Polat, Kemal. (2020). The Effect of Training and Testing Process on Machine Learning in Biomedical Datasets. Mathematical Problems in Engineering. 2020. 1-17. 10.1155/2020/2836236.
32. Prepare dataset for GPT fine-tuning. URL: https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset (дата звернення 24.05.2024)
33. Garbarino, Ernesto. (2019). CronJobs. 10.1007/978-1-4842-5491-2_7.
34. LangGraph Python Official Documentation. URL: https://python.langchain.com/v0.1/docs/langgraph/ (дата звернення 28.05.2024)
35. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. DOI: https://doi.org/10.48550/arXiv.2310.11511 (дата звернення: 01.06.2024).
36. Xu, Ziwei et al. “Hallucination is Inevitable: An Innate Limitation of Large Language Models.” ArXiv abs/2401.11817 (2024): n. pag.
37. Junming, Xu. (2000). On conditional edge-connectivity of graphs. Acta Mathematicae Applicatae Sinica. 16. 414-419. 10.1007/BF02671131.
38. Rodriguez, Carlos & Baez, Marcos & Daniel, Florian & Casati, Fabio & Trabucco, Juan & Canali, Luigi & Percannella, Gianraffaele. (2016). REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. 21-39. 10.1007/978-3-319-38791-8_2.
39. Zendesk GraphQL Schema Documentation. URL: https://zendesk.github.io/conversations-api/ (дата звернення 02.06.2024)

40. GraphQL Query Language. URL: https://graphql.org/ (дата звернення 02.06.2024)
41. Vural, Hulya & Koyuncu, Murat & Guney, Sinem. (2017). A Systematic Literature Review on Microservices. 203-217. 10.1007/978-3-319-62407-5_14.
42. How to benchmark and optimize LLM inference performance. URL: https://ubiops.com/benchmark-and-optimize-llm-inference-performance/#:~:text=Throughput,one%20with%20a%20lower%20throughput (дата звернення 03.06.2024)
43. Google Custom Search API. URL: https://developers.google.com/custom-search/v1/overview (дата звернення 04.06.2024)
44. What is a LLM context window? URL: https://www.techtarget.com/whatis/definition/context-window (дата звернення 04.06.2024)

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ АВТОНОМНОГО АСИСТЕНТА

```
from dotenv import load_dotenv

load_dotenv()

from langchain_core.messages import HumanMessage, AIMessage
from langchain_community.callbacks import get_openai_callback

from flask import Flask, request, jsonify
from graph.workflow import complied_graph
import history
import uuid


app = Flask(__name__)


@app.route('/chatbot', methods=['POST'])
def chatbot():
  data = request.json
  if data is None:
     return jsonify({'error': 'Invalid JSON'}), 400

  print(data)

  customer_id = data.get('customer_id')
  messages = data.get('messages')
  tags = data.get('tags')

  chat_history = history.get_by_session_id(customer_id)

  run_id = uuid.uuid4()
```

```
  try:
    with get_openai_callback() as cb:
      res = complied_graph.invoke(
        {
          "question": messages,
          "chat_history": chat_history,
          "tags": tags,
        },
      )

    print("---TRY TO ADD MESSAGES---")
    chat_history.add_messages([HumanMessage(content=i) for i in messages] +
[AIMessage(content=res["generation"])])

    return {
      'response': res["generation"],
      'total_tokens': cb.total_tokens,
      'prompt_tokens': cb.prompt_tokens,
      'completion_tokens': cb.completion_tokens,
      'total_cost': cb.total_cost,
      'run_id': run_id
    }
  except Exception as e:
    print(e)
    return {
      'response': "Error:\n"+str(e),
      'run_id': run_id
    }

if __name__ == '__main__':
  app.run(port=8001, debug=True)
from langgraph.graph import END, StateGraph
from graph.edge import decide_to_generate, format_reponse,
grade_generation_v_documents_and_question, format_reponse
```

```
from graph.node import retrieve_knowledge, generate_answer, grade_documents, query_google,
rewrite_answer, history_empty, greet, clarify, familiarize
from graph.state import GraphState

workflow = StateGraph(GraphState)

workflow.add_node("greet", greet)

# Define the nodes
workflow.add_node("retrieve_knowledge", retrieve_knowledge)
workflow.add_node("clarify", clarify)
workflow.add_node("query_google", query_google)
workflow.add_node("grade_documents", grade_documents)
workflow.add_node("generate_answer", generate_answer)
workflow.add_node("rewrite_answer", rewrite_answer)
workflow.add_node("familiarize", familiarize)
workflow.add_node("format_response", format_reponse)

# Build graph
workflow.set_conditional_entry_point(
  history_empty,
  {
    "empty": "greet",
    "with messages": "retrieve_knowledge",
  },
)
workflow.add_edge("familiarize", END)
workflow.add_edge("greet", END)
workflow.add_edge("retrieve_knowledge", "grade_documents")
workflow.add_conditional_edges(
  "grade_documents",
  decide_to_generate,
  {
    "all not relevant": "query_google",
    "invalid category": "familiarize",
```

```
      "clarify": "clarify",
      "generate_answer": "generate_answer",
    },
)
workflow.add_edge("query_google", "grade_documents")
workflow.add_edge("clarify", "format_response")
workflow.add_conditional_edges(
    "generate_answer",
    grade_generation_v_documents_and_question,
    {
      "not based on documents": "query_google",
      "valid": "format_response",
      "not useful": "query_google",
      "contain not valid url": "rewrite_answer",
    },
)
workflow.add_conditional_edges(
    "rewrite_answer",
    grade_generation_v_documents_and_question,
    {
      "valid": "format_response",
      "not useful": "query_google",
      "contain not valid url": "rewrite_answer",
    },
)
workflow.set_finish_point("format_response")


complied_graph = workflow.compile()

print(complied_graph.get_graph().draw_mermaid())

from langchain_core.messages import BaseMessage

from typing_extensions import TypedDict
```

```
from typing import List


class GraphState(TypedDict):
  question: str
  generation: str
  documents: List[str]
  chat_history: List[BaseMessage]
  tags: List[str]

from chain.rewriter import answer_rewriter
from chain.answer_chain import answer_chain
from chain.grade_documents import document_grader
from chain.clarifying_chain import clarifying_chain
from chain.generalize_chain import generalize_chain
from retriever.google_search import google_search_retriever_chain
from retriever.experts_knowledge_base import knowledge_base_retriever_chain

from typing import Optional, List, Dict, Any
from toml import load as toml_load


def retrieve_knowledge(state):
  print("---RETRIEVE KNOWLEDGE---")
  question = state["question"]
  history = state["chat_history"]

  # Retrieval
  documents = knowledge_base_retriever_chain.invoke({"input": question, "chat_history":
history.messages})
  return {"documents": documents, "question": question}


def generate_answer(state):
  print("---GENERATE---")
```

```
    question = state["question"]
    documents = state["documents"]

    answer = answer_chain.invoke({"context": documents, "question": question})
    return {"documents": documents, "question": question, "generation": answer}


def grade_documents(state):
    print("---CHECK DOCUMENT RELEVANCE TO QUESTION---")
    question = state["question"]
    documents = state["documents"]
    history = state["chat_history"]

    problem_summary = generalize_chain.invoke({"question": question, "chat_history":
history.messages})
    print("---SUMMARIZE QUESTION----")
    print(problem_summary)

    filtered_docs = []
    for d in documents:
        score = document_grader.invoke(
            {"question": problem_summary.content, "document": d.page_content}
        )
        grade = score.binary_score
        if grade == "yes":
            print("---GRADE: DOCUMENT RELEVANT---")
            filtered_docs.append(d)
        else:
            print("---GRADE: DOCUMENT NOT RELEVANT---")
            continue
    return {"documents": filtered_docs, "question": question}


def query_google(state):
    print("---QUERY GOOGLE---")
```

```
    question = state["question"]
    history = state["chat_history"]

    documents = google_search_retriever_chain.invoke({"input": question, "chat_history":
history.messages})

    return {"documents": documents, "question": question}


def rewrite_answer(state):
    """
    Produce a better answer.

    Args:
        state (dict): The current graph state

    Returns:
        state (dict): Updates question key with a re-phrased question
    """

    print("---TRANSFORM QUERY---")
    answer = state["generation"]
    documents = state["documents"]

    fixed_answer = answer_rewriter.invoke({"answer": answer, "context": documents})
    return {"documents": documents, "generation": fixed_answer}


def clarify(state):
    """
    Asks question to retrieve required for solution details

    Args:
        state (dict): The current graph state
```

```
    Returns:
        state (dict): State after transformation
    """

    print("---CLARIFY---")
    question = state["question"]
    documents = state["documents"]
    history = state["chat_history"]

    clarifying_question = clarifying_chain.invoke({"question": question, "chat_history":
history.messages, "context": documents})

    return {"generation": clarifying_question.content}


def history_empty(state):
    """
    Route question to greeting or RAG.

    Args:
        state (dict): The current graph state

    Returns:
        str: Next node to call
    """

    history = state["chat_history"]

    if len(history.messages) == 0:
        return "empty"

    return "with messages"


def greet(state):
```

```
    """
    Greet customer with welcome message.

    Args:
        state (dict): The current graph state

    Returns:
        state (dict): State after transformation
    """

    print("---GREET----")
    print(state)

    tags = state["tags"]
    project_name = (extract_project_name(tags) or "default")

    with open('config/projects.toml', 'r') as f:
        config = toml_load(f)[project_name]

        if is_first_chat(tags):
            return {"generation": config["greeting"]}

        return {"generation": config["returning_greeting"]}

def familiarize(state):
    """
    Familiarize customer with allowed categories
    Args:
        state (dict): The current graph state

    Returns:
        state (dict): State after transformation
    """

    return {"generation": "I'm here to assist with any tech-related issues or inquiries you might
```

```
have.\nIf you have a question about your devices, software, or any tech topic, please let me know, and I'll be happy to help!"}


def is_first_chat(tags: List[str]) -> bool:
  return 'first_chat_for_user' in tags


def extract_project_name(tags: List[str]) -> Optional[str]:
  for tag in tags:
    if '_ticket' in tag:
      return tag.split('_ticket')[0]

  return None

from chain.grade_hallucinations import hallucination_grader
from chain.grade_answer import answer_grader
from chain.grade_details import details_grader
from chain.generalize_chain import generalize_chain
from chain.verify_category import category_verifier

import re
import requests

def decide_to_generate(state):
  """
  Determines whether to generate an answer, or re-generate a question.

  Args:
    state (dict): The current graph state

  Returns:
    str: Binary decision for next node to call
  """
```

```
    question = state["question"]
    history = state["chat_history"]


    score = details_grader.invoke({
        "question": question,
        "chat_history": history.messages,
    })
    grade = score.binary_score

    if grade == "yes":
        print("---ASSESS GRADED DOCUMENTS---")
        filtered_documents = state["documents"]

        if not filtered_documents:
            print(
                "---DECISION: ALL DOCUMENTS ARE NOT RELEVANT TO QUESTION, QUERY
GOOGLE---"
            )

            return "all not relevant"

        if len(history.messages) < 4:
            return "clarify"

        # We have relevant documents, so generate answer
        print("---DECISION: GENERATE---")
        return "generate_answer"
    else:
        score = category_verifier.invoke({
            "question": question,
            "category": "Tech, Services, Household electronics",
            "chat_history": history.messages,
        })
        grade = score.binary_score
```

```
        if grade == "yes":
            print("---DECISION: CLARIFY---")
            return "clarify"

        return "invalid category"


def verify_urls(state):
    """
    Determines whether answer contain valid URL's.

    Args:
        state (dict): The current graph state

    Returns:
        str: Decision for next node to call
    """

    print("---VERIFY RESPONSE URL's---")
    answer = state["generation"]
    print(answer)

    #
    regex = "https?://[-%_.!~*';/?:@&=+$,A-Za-z0-9]+"
    urls = re.findall(regex, answer)

    for url in urls:
        if requests.get(url).status_code == 404:
            print("---NO URL FOUND---")
            print(url, requests.get(url).status_code)
            return "contain not valid url"
        print(url, requests.get(url).status_code)

    return "valid"
```

```
def format_docs(docs):
  return "\n\n".join(doc.page_content for doc in docs)

def grade_generation_v_documents_and_question(state):
  """
  Determines whether the generation is grounded in the document and answers question.

  Args:
    state (dict): The current graph state

  Returns:
    str: Decision for next node to call
  """

  print("---CHECK HALLUCINATIONS---")
  question = state["question"]
  documents = state["documents"]
  generation = state["generation"]
  history = state["chat_history"]

  score = hallucination_grader.invoke(
    {"documents": format_docs(documents), "generation": generation}
  )
  grade = score.binary_score

  problem_summary = generalize_chain.invoke({"question": question, "chat_history":
history.messages})

  # Check hallucination
  if grade == "yes":
    print("---DECISION: GENERATION IS GROUNDED IN DOCUMENTS---")
    # Check question-answering
    print("---GRADE GENERATION vs QUESTION---")
    score = answer_grader.invoke({"question": problem_summary.content, "generation":
```

```
generation})
      grade = score.binary_score
      if grade == "yes":
         print("---DECISION: GENERATION ADDRESSES QUESTION---")
         return verify_urls(state)
      else:
         print("---DECISION: GENERATION DOES NOT ADDRESS QUESTION---")
         return "not useful"
   else:
      print("---DECISION: GENERATION IS NOT GROUNDED IN DOCUMENTS,
RE-TRY---")
      return "not based on documents"



def format_reponse(state):
   state["generation"] = re.sub(r'\*\*(.*?)\*\*', r'\1', state["generation"])

   return state

from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_core.pydantic_v1 import BaseModel, Field
from langchain_openai import ChatOpenAI



class VerifyCategory(BaseModel):
   """Binary score to assess answer addresses question."""

   binary_score: str = Field(
      description="Answer addresses the question, 'yes' or 'no'"
   )



system = """
You are a verifier of customer support service who determines whether a customer message
follows service chat categories. The goal is to filter out messages that do not relate to the specified
```

topics in any way.

1. If the customer message looks like an answer to a previous question, return "yes."
2. If the customer message involves seeking help, support, or information related to the specified categories, return "yes."
3. If you cannot determine if the customer message follows a specific category, use the provided chat history to find the context.
4. Evaluate the entire conversation, including previous interactions, to understand the context and correctly categorize the message.
5. Answer "no", only If you are 100% sure that message does not belog to a chat category and does not seems logical in chat context.

Give a binary rating of 'yes' or 'no' based on the rules above. "yes" means the message follows the specified categories."""

```
verify_prompt = ChatPromptTemplate.from_messages(
  [
    ("system", system),
    MessagesPlaceholder(variable_name="chat_history"),
    ("human", "Customer message: {question} \n\n Categories: {category}"),
  ]
)

llm = ChatOpenAI(model="gpt-4o", temperature=0)
structured_llm_verifier = llm.with_structured_output(VerifyCategory)

category_verifier = verify_prompt | structured_llm_verifier

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

rewrite_answer_prompt = PromptTemplate(
  input_variables=["answer", "context"],
  template="""
```

```
  You a answers re-writer that converts an input answer with invalid URL (that are not exist).
  You should rewrite invalid answer based on context that solved problem with another URL provided.
  If you can't replace URL in invalid answer because this is the only one that links with actual solution, omit it in fixed answer.
  Format the response in plain text without enhancements. Markdown format is not supported.

  Invalid answer: {answer}

  Context: {context}

  Fixed answer: """
)

llm = ChatOpenAI(model="gpt-3.5-turbo-0125")

answer_rewriter = rewrite_answer_prompt | llm | StrOutputParser()

from langchain_core.prompts import ChatPromptTemplate
from langchain_core.pydantic_v1 import BaseModel, Field
from langchain_openai import ChatOpenAI


class GradeHallucinations(BaseModel):
  binary_score: str = Field(
    description="Answer is grounded in the facts, 'yes' or 'no'"
  )


system = """
  You are a grader assessing whether an LLM generation is grounded in / supported by a set of retrieved facts.
  Give a binary score 'yes' or 'no'. 'Yes' means that the answer is grounded in / supported by the set of facts.
"""
```

```
hallucination_prompt = ChatPromptTemplate.from_messages(
  [
    ("system", system),
    ("human", "Set of facts: \n\n {documents} \n\n LLM generation: {generation}"),
  ]
)

llm = ChatOpenAI(model="gpt-3.5-turbo-0125")
grader = llm.with_structured_output(GradeHallucinations)

hallucination_grader = hallucination_prompt | grader

from langchain_core.prompts import ChatPromptTemplate
from langchain_core.pydantic_v1 import BaseModel, Field
from langchain_openai import ChatOpenAI



class GradeDocuments(BaseModel):
  binary_score: str = Field(
    description="Documents are relevant to the question, 'yes' or 'no'"
  )



system = """
  You are a grader assessing relevance of a retrieved document to a question summary. The goal is to filter out erroneous retrievals.
  If the document contains solution for customer's question, grade it as relevant.
  Document's can be in both English and Ukrainian, take it into account when grading.
  If customer is facing problem with some service, make sure that this service is mentioned in the documents.
  Give a binary score 'yes' or 'no' score to indicate whether the document is relevant to the question.
"""
grade_prompt = ChatPromptTemplate.from_messages(
  [
```

```
    ("system", system),
    ("human", "Retrieved document: \n\n {document} \n\n Question summary: {question}"),
  ]
)

llm = ChatOpenAI(model="gpt-3.5-turbo-0125")
grader = llm.with_structured_output(GradeDocuments)

document_grader = grade_prompt | grader

from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_core.pydantic_v1 import BaseModel, Field
from langchain_openai import ChatOpenAI



class GradeAnswer(BaseModel):
  binary_score: str = Field(
    description="Enough information to provide solution based on context, 'yes' or 'no'"
  )



system = """
  You are a grader assessing whether details about the problem is enough to provide solution based on messages history.
  Make sure that all required information is gathered and you can identify concrete devices models or versions of software.
  Give a binary score 'yes' or 'no'. 'Yes' means that information in chat is enough to identify key problem and specific details. 'No' means that you found at least one ambiguity that interfere exact solution.
"""
details_grader_prompt = ChatPromptTemplate.from_messages(
  [
    ("system", system),
    MessagesPlaceholder(variable_name="chat_history"),
    ("human", "{question}"),
```

```
  ]
)

llm = ChatOpenAI(model="gpt-3.5-turbo-0125")
grader = llm.with_structured_output(GradeAnswer)

details_grader = details_grader_prompt | grader

from langchain_core.prompts import ChatPromptTemplate
from langchain_core.pydantic_v1 import BaseModel, Field
from langchain_openai import ChatOpenAI


class GradeAnswer(BaseModel):
  binary_score: str = Field(
    description="Answer addresses the question, 'yes' or 'no'"
  )


system = """
  You are a grader that assessing whether an answer addresses / resolves a question.
  Give a binary score 'yes' or 'no'. yes' means that the answer resolves the question.
"""
answer_prompt = ChatPromptTemplate.from_messages(
  [
    ("system", system),
    ("human", "Question summary: \n\n {question} \n\n LLM generation: {generation}"),
  ]
)

llm = ChatOpenAI(model="gpt-3.5-turbo-0125", temperature=0)
grader = llm.with_structured_output(GradeAnswer)

answer_grader = answer_prompt | grader
```

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.pydantic_v1 import BaseModel, Field
from langchain_openai import ChatOpenAI


class GradeAnswer(BaseModel):
  binary_score: str = Field(
    description="Answer addresses the question, 'yes' or 'no'"
  )


system = """
  You are a grader that assessing whether an answer addresses / solves a question.
  Give a binary score 'yes' or 'no'. yes' means that the answer solves the question.
"""
answer_prompt = ChatPromptTemplate.from_messages(
  [
    ("system", system),
    ("human", "Question summary: \n\n {question} \n\n LLM generation: {generation}"),
  ]
)

llm = ChatOpenAI(model="gpt-3.5-turbo-0125")
grader = llm.with_structured_output(GradeAnswer)

answer_grader = answer_prompt | grader

from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder

prompt = ChatPromptTemplate.from_messages(
  [
    (
      "system",
      """
```

You are an technical expert on a consultations website.

The goal is to collect information about the customer's problem, so effective solution can be provided later.

Craft clear and concise questions that efficiently collect the necessary information.

Strictly follow this rules:

1. Ask clarifying questions in a logical sequence to gather all required information. Ensure that each question builds upon the previous responses to speed up the process.

2. Ask ONLY ONE question per time.

3. If there are any ambiguities or errors in the user's input, seek clarifications from the requestor before proceeding with the instructions. It is crucial to have accurate and appropriate instructions to guide the AI effectively.

4. Ask questions that required for providing solution, don't overcomplicate.

5. Avoid ambiguity in clarifying questions to improve understanding and accuracy. Try to ask specific quesitons.

6. Try to avoid questions about contacting another customer support service.

Important! Questions should be easy to answer, even for people who not have knowledge about problem resolution.

In order to form clarifying question you can use the following information:

Context: {context}
"""),
),
MessagesPlaceholder(variable_name="chat_history"),
("human", "{question}")
]
)

```
llm = ChatOpenAI(model="gpt-4o", temperature=0.2)

clarifying_chain = prompt | llm
```

```
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import PromptTemplate
from langchain_openai import ChatOpenAI

solution_prompt = PromptTemplate(
  input_variables=["question", "context"],
  template="""
  You are a world-class expert in technical support, provided with knowledge on how to resolve customer's problem, \
  and you need to extract information to make a clear and concise answer. Form a complete answer based on the given knowledge. \
  Your answer should be in a human-like manner. With positive attitude.

  Make sure that your answer grounded in / supported by a set of retrieved facts

  Rules:
  1. Try to give simple solutions, that easy to implement and ideally avoid contacting 3rd party.
  2. If you found shortcut (шорткат) in context that solves the problem, you can use it as an answer.
  3. NEVER provide generic answers on how issue can be solved, try to provide a consise personalized answer. If you ask the customer to perform some action, describe in detail how to perform it.
  4. If you include URLs in the answer, make sure they are relevant and closely related to the piece of context you used to create your answer.
  5. Do NOT use any markup languages. any text enchancemenents, markdown or other formats are prohibited.

  Your answer should follow rules above, be consise and contain only required details about the specific problem customer wan't to solve and ideally be in the form of steps required to complete the solution.
  Remember! Asking customer to contact another customer support service is last-last-resort, do it, only if there is no another option to solve an issue.

  Question: {question}
  Context: {context}
```

```
  Answer: """
)

llm = ChatOpenAI(model_name="gpt-4o", temperature=0)

answer_chain = solution_prompt | llm | StrOutputParser()


import psycopg
import logging
from typing import List, Optional, Sequence
from langchain_core.chat_history import BaseChatMessageHistory
from langchain_core.messages import BaseMessage, message_to_dict, messages_from_dict
from psycopg import sql
import json

db_params = {
  'dbname': 'example',
  'user': 'example',
  'password': 'example',
  'host': 'localhost',
  'port': '5432'
}

conn_info = "dbname='{dbname}' user='{user}' password='{password}' host='{host}'
port='{port}'".format(**db_params)
sync_connection = psycopg.connect(conn_info)

table_name = 'chat_history'
schema = 'chat'

logger = logging.getLogger(__name__)


def _get_messages_query(table_name: str, schema: str) -> sql.Composed:
```

```
  return sql.SQL(
    "SELECT message "
    "FROM {schema}.{table_name} "
    "WHERE session_id = %(session_id)s "
    "ORDER BY id;"
  ).format(
    table_name=sql.Identifier(table_name),
    schema=sql.Identifier(schema)
  )


def _insert_message_query(table: str, schema: str) -> sql.Composed:
  return sql.SQL(
    "INSERT INTO {schema}.{table} (session_id, message) VALUES (%s, %s)"
  ).format(
    table_name=sql.Identifier(table_name),
    schema=sql.Identifier(schema)
  )


class PostgresChatMessageHistory(BaseChatMessageHistory):
  def __init__(
    self,
    table_name: str,
    schema: str,
    session_id: str,
    /,
    *,
    sync_connection: Optional[psycopg.Connection] = None,
  ) -> None:
    self._connection = sync_connection

    self._session_id = session_id
    self._table_name = table_name
    self._schema = schema
```

```
def add_messages(self, messages: Sequence[BaseMessage]) -> None:
    """Add messages to the chat message history."""
    if self._connection is None:
        raise ValueError(
            "Please provide a PostgreSQL conection",
        )

    values = [
        (self._session_id, json.dumps(message_to_dict(message)))
        for message in messages
    ]

    query = _insert_message_query(self._table_name, self._schema)

    with self._connection.cursor() as cursor:
        cursor.executemany(query, values)
    self._connection.commit()

def get_messages(self) -> List[BaseMessage]:
    """Get chat messages history."""
    if self._connection is None:
        raise ValueError(
            "Please provide a PostgreSQL conection",
        )

    query = _get_messages_query(self._table_name, self._schema)

    with self._connection.cursor() as cursor:
        print("---EXECUTE GET MESSAGES---")
        cursor.execute(query, {"session_id": self._session_id})
        items = [record[0] for record in cursor.fetchall()]

    messages = messages_from_dict(items)
    print("---END GET MESSAGES---")
```

```
        return messages

    @property
    def messages(self) -> List[BaseMessage]:
        return self.get_messages()


def get_by_session_id(session_id: str) -> BaseChatMessageHistory:
    print("---GET BY SESSION ID---")
    return PostgresChatMessageHistory(
        table_name,
        schema,
        session_id,
        sync_connection=sync_connection
)

from langchain_community.utilities import GoogleSearchAPIWrapper
from langchain_core.prompts import PromptTemplate
from langchain.retrievers.web_research import WebResearchRetriever
from langchain_community.vectorstores import Chroma
from langchain_openai import OpenAI, ChatOpenAI
from langchain_openai import OpenAIEmbeddings
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain.chains import create_history_aware_retriever

DEFAULT_SEARCH_PROMPT = PromptTemplate(
    input_variables=["question"],
    template="""You are an assistant tasked with requesting clear information Google search. \
        Generate ONLY ONE (1) Google search query in imperative form. Do not use question formats. \
        You should analyze question provided to you, in order to generate query. \
        Query should be concise, built in imperative format, and contain all necessary information to get the most quality responses.
        Do NOT include any personal information about customer.
        The output should be a numbered list of queries.
```

```
        Question: {question}
        """,
)

chroma_vectorstore = Chroma(embedding_function=OpenAIEmbeddings())

google_search_retriever = WebResearchRetriever.from_llm(
  vectorstore=chroma_vectorstore,
  llm=OpenAI(model_name="gpt-3.5-turbo-instruct"),
  search=search,
  prompt=DEFAULT_SEARCH_PROMPT,
)

generalize_prompt = ChatPromptTemplate.from_messages(
  [
    MessagesPlaceholder(variable_name="chat_history"),
    ("human", "{input}"),
    (
      "system",
      """Given conversation with customer and technical expert, summarize the problem
customer is facing into consise sentence. In order to efficient search of related information.
      Do NOT answer to customer's question, they provided only for context.
      """,
    ),
  ]
)

llm = ChatOpenAI(model="gpt-3.5-turbo-0125")

google_search_retriever_chain = create_history_aware_retriever(
  llm, google_search_retriever, generalize_prompt
)

from langchain_openai import OpenAIEmbeddings
```

```
from langchain_community.vectorstores import BigQueryVectorSearch
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain.chains import create_history_aware_retriever
from langchain_openai import ChatOpenAI

embedding_model = OpenAIEmbeddings(model="text-embedding-3-large")

vectorstore = BigQueryVectorSearch(
  project_id='test',
  dataset_name='test',
  table_name='test',
  location='US',
  embedding=embedding_model,
)

experts_knowledge_base_retriever = vectorstore.as_retriever(k=3)

generalize_prompt = ChatPromptTemplate.from_messages(
  [
    MessagesPlaceholder(variable_name="chat_history"),
    ("human", "{input}"),
    (
      "system",
      """Given conversation with customer and technical expert, summarize the problem
customer is facing into consise sentence. In order to efficient search of related information.
      Do NOT answer to customer's question, they provided only for context.
      """,
    ),
  ]
)

llm = ChatOpenAI(model="gpt-3.5-turbo-0125")
knowledge_base_retriever_chain = create_history_aware_retriever(
  llm, experts_knowledge_base_retriever, generalize_prompt
)
```

ДОДАТОК Б ЛІСТИНГ ПРОГРАМИ ІНТЕГРАЦІЇ З CRM

```
package main

import (
  "log"
  "net/http"
  "os"

  "aichatservice/internal"
  "aichatservice/pkg/zendesk"

  _ "github.com/joho/godotenv/autoload"

  zendesk_pkg "gitlab.com/integrator/be/libs/pkg/zendesk"
)

func main() {
  client := zendesk.NewClient(zendesk.Config{
    Token: os.Getenv("ZOPIM_OAUTH_TOKEN"),
  })

  wsURL, err := client.StartAgentSession()
  if err != nil {
    log.Fatalf("Failed to start agent session: %v", err)
  }

  conn, err := client.ConnectWS(wsURL)
  if err != nil {
    log.Fatal(err)
  }
  defer client.CloseWS(conn)

  registry := internal.NewMessageProcessorRegistry(conn)

  zendeskClient := zendesk_pkg.NewClient(
    http.DefaultClient,
```

```
    os.Getenv("ZENDESK_LOGIN"),
    os.Getenv("ZENDESK_PASSWORD"),
    os.Getenv("ZENDESK_API_BASE_URL"),
  )

  visitorInfoProcessor := internal.NewUpdateVisitorInfoProcessor()
  chatMsgProcessor := internal.NewChatMessageProcessor(conn,
visitorInfoProcessor.UsersInfo(), zendeskClient)

  registry.Register(visitorInfoProcessor, chatMsgProcessor)

  if err := registry.Listen(); err != nil {
    log.Fatal(err)
  }
}
```

```
package zendesk

import (
  "bytes"
  "encoding/json"
  "errors"
  "fmt"
  "io"
  "log"
  "net/http"
  "time"

  "github.com/gorilla/websocket"
  "github.com/machinebox/graphql"
  "gitlab.com/integrator/be/libs/pkg/zendesk/zopim"
)

const _apiURL = "https://chat-api.zopim.com/graphql/request"

type Client struct {
  *graphql.Client

  token      string
  zopimClient *zopim.Client
```

```
  ticker      *time.Ticker
  tickerDone  chan bool
}

func NewClient(cfg Config) *Client {
  return &Client{
    Client:      graphql.NewClient(_apiURL),
    zopimClient: zopim.NewClient(http.DefaultClient, "https://www.zopim.com/api/v2",
cfg.Token),
    token:       cfg.Token,
    ticker:      time.NewTicker(3 * time.Second),
    tickerDone:  make(chan bool),
  }
}

func (c *Client) StartAgentSession() (string, error) {
  query := `mutation($access_token: String!) {
    startAgentSession(access_token: $access_token) {
      websocket_url
      session_id
      client_id
    }
  }`

  variables := map[string]interface{}{"access_token": c.token}
  body, err := json.Marshal(map[string]interface{}{"query": query, "variables": variables})
  if err != nil {
    return "", err
  }

  req, err := http.NewRequest("POST", _apiURL, bytes.NewBuffer(body))
  if err != nil {
    return "", err
  }

  req.Header.Set("Content-Type", "application/json")

  client := &http.Client{}
  resp, err := client.Do(req)
  if err != nil {
```

```
    return "", err
  }
  defer resp.Body.Close()

  responseData, err := io.ReadAll(resp.Body)
  if err != nil {
    return "", err
  }

  var sessionResp sessionResponse
  if err := json.Unmarshal(responseData, &sessionResp); err != nil {
    return "", err
  }

  return sessionResp.Data.StartAgentSession.WebsocketURL, nil
}

func (c *Client) ConnectWS(urlStr string) (*websocket.Conn, error) {
  conn, _, err := websocket.DefaultDialer.Dial(urlStr, nil)
  if err != nil {
    return nil, err
  }

  log.Println("Successfully connected to WebSocket")

  query := `subscription {
    message {
      node {
        id
        content
        channel {
          id
        }
        from {
          __typename
          id
          display_name
        }
      }
    }
```

```
}
`

subscriptionMessage := WSRequest{
  Type: "request",
  ID:   1,
  Payload: Payload{
    Query: query,
  },
}

subscriptionMessageJSON, err := json.Marshal(subscriptionMessage)
if err != nil {
  return nil, err
}

err = conn.WriteMessage(websocket.TextMessage, subscriptionMessageJSON)
if err != nil {
  return nil, err
}

_, respBody, err := conn.ReadMessage()
if err != nil {
  return nil, err
}

respMessage := WSMessage{}
if err := json.Unmarshal(respBody, &respMessage); err != nil {
  return nil, err
}

if respMessage.ErrorCode != nil {
  return nil, errors.New("unable to subscribe for chat messages")
}

go func() {
  for {
    select {
    case <-c.ticker.C:
      if err := c.pingWS(conn); err != nil {
```

```
            log.Fatal(err)
          }
        case <-c.tickerDone:
          fmt.Println("Ticker stopped")
          return
        }
      }
    }()

    return conn, err
}

func (c *Client) CloseWS(conn *websocket.Conn) error {
    c.ticker.Stop()
    c.tickerDone <- true

    return conn.Close()
}

func (c *Client) pingWS(conn *websocket.Conn) error {
    subscriptionMessage := WSRequest{
        Sig: "PING",
        Payload: Payload{
            Query: "",
        },
    }

    pingMessageJSON, err := json.Marshal(subscriptionMessage)
    if err != nil {
        return err
    }

    if err := conn.WriteMessage(websocket.TextMessage, pingMessageJSON); err != nil {
        return err
    }

    fmt.Println("PING sent")
    return nil
}
```

```
type WSMessage struct {
  Type      string          `json:"type,omitempty"`
  Sig       string          `json:"sig,omitempty"`
  Payload   json.RawMessage `json:"payload,omitempty"`
  ErrorCode *int            `json:"errorCode,omitempty"`
}

type WSRequest struct {
  Type    string  `json:"type"`
  ID      int     `json:"id"`
  Sig     string  `json:"sig"`
  Payload Payload `json:"payload"`
}

type Payload struct {
  Query string `json:"query"`
}

type subsciptionResponse struct {
  Data struct {
    StartAgentSession struct {
      WebsocketURL string `json:"websocket_url"`
      SessionId    string `json:"session_id"`
      ClientId     string `json:"client_id"`
    } `json:"startAgentSession"`
  } `json:"data"`
}

type sessionResponse struct {
  Data struct {
    StartAgentSession struct {
      WebsocketURL string `json:"websocket_url"`
      SessionId    string `json:"session_id"`
      ClientId     string `json:"client_id"`
    } `json:"startAgentSession"`
  } `json:"data"`
}

package zendesk
```

```
type UpdateVisitorInfoResponse struct {
  Data UpdateVisitorInfoData `json:"data"`
}

type UpdateVisitorInfoData struct {
  UpdateVisitorInfo UpdateVisitorInfo `json:"updateVisitorInfo"`
}

type UpdateVisitorInfo struct {
  Node UpdateVisitorInfoNode `json:"node"`
}

type UpdateVisitorInfoNode struct {
  ID    string `json:"id"`
  Email string `json:"email"`
}

package zendesk

import (
  "encoding/base64"
  "encoding/json"
)

type MessageResponse struct {
  Data struct {
     Message Message `json:"message"`
  } `json:"data"`
}

func (r *MessageResponse) ChannelID() string {
  return r.Data.Message.Node.Channel.ID
}

func (r *MessageResponse) Content() string {
  return r.Data.Message.Node.Content
}

func (r *MessageResponse) FromVisitor() bool {
  return r.Data.Message.Node.From.Typename == "Visitor"
```

```
}

func (r *MessageResponse) ScribeID() (string, error) {
  id, err := base64.StdEncoding.DecodeString(r.Data.Message.Node.Channel.ID)
  if err != nil {
    return "", err
  }

  var meta [][]string
  if err := json.Unmarshal(id, &meta); err != nil {
    return "", err
  }

  return meta[1][1], nil
}

func (r *MessageResponse) VisitorId() (string, error) {
  id, err := base64.StdEncoding.DecodeString(r.Data.Message.Node.From.ID)
  if err != nil {
    return "", err
  }

  var meta [][]string
  if err := json.Unmarshal(id, &meta); err != nil {
    return "", err
  }

  return meta[0][0], nil
}

type Message struct {
  Node Node `json:"node"`
}

type Node struct {
  ID      string   `json:"id"`
  Content string   `json:"content"`
  Channel Channel  `json:"channel"`
  From    FromType `json:"from"`
}
```

```
type Channel struct {
  ID string `json:"id"`
}

type FromType struct {
  Typename    string `json:"__typename"`
  ID          string `json:"id"`
  DisplayName string `json:"display_name"`
}

package zendesk

type Config struct {
  Token string
}

package internal

import (
  "aichatservice/pkg/zendesk"
  "encoding/json"
)

type UpdateVisitorInfoProcessor struct {
  users map[string]chan string
}

func NewUpdateVisitorInfoProcessor() *UpdateVisitorInfoProcessor {
  return &UpdateVisitorInfoProcessor{
     users: make(map[string]chan string),
  }
}

func (p *UpdateVisitorInfoProcessor) UsersInfo() map[string]chan string {
  return p.users
}

func (p *UpdateVisitorInfoProcessor) Process(message zendesk.WSMessage) error {
  var data zendesk.UpdateVisitorInfoResponse
```

```
  if err := json.Unmarshal(message.Payload, &data); err != nil {
    return err
  }

  node := data.Data.UpdateVisitorInfo.Node
  if node.Email == "" {
    return nil
  }

  p.users[node.ID] <- node.Email

  return nil
}

package internal

import (
  "aichatservice/pkg/zendesk"
  "encoding/json"
  "log"

  "github.com/gorilla/websocket"
)

type MessageProcessorRegistry struct {
  conn       *websocket.Conn
  processors []Processor
}

func NewMessageProcessorRegistry(conn *websocket.Conn) *MessageProcessorRegistry {
  return &MessageProcessorRegistry{
    conn: conn,
  }
}

func (p *MessageProcessorRegistry) Register(processors ...Processor) {
  for _, processor := range processors {
    p.processors = append(p.processors, processor)
  }
```

```
}

func (p *MessageProcessorRegistry) Listen() error {
  for {
    _, respBody, err := p.conn.ReadMessage()
    if err != nil {
      return err
    }

    message := zendesk.WSMessage{}
    if err := json.Unmarshal(respBody, &message); err != nil {
      return err
    }

    for _, processor := range p.processors {
      go func(proc Processor) {
        if err := proc.Process(message); err != nil {
          log.Println(err)
        }
      }(processor)
    }
  }
}

package internal

import "aichatservice/pkg/zendesk"

type Processor interface {
  Process(message zendesk.WSMessage) error
}

package internal

import (
  "aichatservice/pkg/zendesk"
  "bytes"
  "context"
  "encoding/json"
  "fmt"
```

```
  "io"
  "net/http"
  "strconv"

  "github.com/gorilla/websocket"
  zendesk_pkg "gitlab.com/integrator/be/libs/pkg/zendesk"
)

type ChatMessageProcessor struct {
  conn   *websocket.Conn
  users  map[string]chan string
  client *zendesk_pkg.Client
}

func NewChatMessageProcessor(
  conn *websocket.Conn,
  users map[string]chan string,
  zendeskClient *zendesk_pkg.Client,
) *ChatMessageProcessor {
  return &ChatMessageProcessor{
    conn:   conn,
    users:  users,
    client: zendeskClient,
  }
}

func (p *ChatMessageProcessor) Process(message zendesk.WSMessage) error {
  var data zendesk.MessageResponse

  if err := json.Unmarshal(message.Payload, &data); err != nil {
    return err
  }

  if data.Content() == "" || !data.FromVisitor() {
    return nil
  }

  visitorEmail, err := p.RequestUpdateVisitorInfo(data.Data.Message.Node.From.ID)
  if err != nil {
    return err
```

```
  }

  user, err := p.client.SearchUserByEmail(context.TODO(), visitorEmail)
  if err != nil {
    return err
  }

  ticket, err := p.client.GetLatestChatTicketByRequesterId(context.TODO(), "chat", user.Id)
  if err != nil {
    return err
  }

  reqBody := struct {
    Channel string   `json:"customer_id"`
    Message string   `json:"message"`
    Tags    []string `json:"tags"`
  }{
    Channel: data.ChannelID(),
    Message: data.Content(), Tags: ticket.Tags}

  bodyJson, err := json.Marshal(reqBody)
  if err != nil {
    return err
  }

  resp, err := http.Post("http://localhost:8000/chatbot", "application/json",
bytes.NewReader(bodyJson))
  if err != nil {
    return err
  }

  respBody, err := io.ReadAll(resp.Body)
  if err != nil {
    return err
  }

  aiAnswer := struct {
    Response string `json:"response"`
  }{}
```

```
  if err := json.Unmarshal(respBody, &aiAnswer); err != nil {
    return err
  }

  queryTemplate := `mutation {
       sendMessage(channel_id:"%s", msg: %s) {
         success
       }
    }`

  escapedAIAnswer := strconv.Quote(aiAnswer.Response)

  query := fmt.Sprintf(queryTemplate, data.ChannelID(), escapedAIAnswer)

  replyMessage := zendesk.WSRequest{
    Type: "request",
    ID:   1,
    Payload: zendesk.Payload{
      Query: query,
    },
  }

  replyMessageJSON, err := json.Marshal(replyMessage)
  if err != nil {
    return err
  }

  return p.conn.WriteMessage(websocket.TextMessage, replyMessageJSON)
}

func (p *ChatMessageProcessor) RequestUpdateVisitorInfo(visitorId string) (string, error) {
  updateVisitorInfoTemplate := `mutation {
       updateVisitorInfo(visitor_id:"%s") {
         node {
           email
           id
         }
       }
    }`
```

```
  updateVisitorInfoQuery := fmt.Sprintf(updateVisitorInfoTemplate, visitorId)

  updateVisitorInfo := zendesk.WSRequest{
    Type: "request",
    ID:   1,
    Payload: zendesk.Payload{
      Query: updateVisitorInfoQuery,
    },
  }

  updateVisitorInfoJSON, err := json.Marshal(updateVisitorInfo)
  if err != nil {
    return "", err
  }

  if err := p.conn.WriteMessage(websocket.TextMessage, updateVisitorInfoJSON); err != nil {
    return "", err
  }

  visitorChannel := make(chan string)
  p.users[visitorId] = visitorChannel

  return <-visitorChannel, nil
}
package model

import (
  "time"

  "github.com/uptrace/bun"
)

type ZendeskTicket struct {
  bun.BaseModel `bun:"table:chat.tickets"`

  Id             int `bun:",pk"`
  Tags           []string
  RequesterEmail string
  CreatedAt      time.Time
}
```

```
func NewZendeskTicket(
  id int,
  tags []string,
  requesterEmail string,
) *ZendeskTicket {
  return &ZendeskTicket{
    Id:             id,
    Tags:           tags,
    RequesterEmail: requesterEmail,
    CreatedAt:      time.Now().UTC(),
  }
}
```

ДОДАТОК В ВСМІСТ ДАТАСЕТУ «metrics.csv»

timestamp,response_time_first,response_time_max,response_time_avg,duration,missed,count_visitor,count_agent,rating,abandon_time,dropped,problem_solved,problem_category,priority

2022-09-01 20:52:37.000000,24,24,24.243000030517578,147,false,1,3,,,,user_left_after_instructions,services__social_media__facebook__hacked_account,normal

2022-09-29 00:55:16.000000,28,30,28.039999961853027,382,false,3,6,,,,user_left_before_instructions,services__apple_services__app_store_/_itunes,normal

2022-12-13 14:52:27.000000,19,19,16.37749993801117,144,false,3,3,,,,user_left_before_instructions,other__no_info,normal

2023-05-07 21:51:42.000000,27,50,38.64099991321564,162,false,2,3,good,,false,forwarded_to_3rd_party,tech_food_services_delivery_doordash,normal

2023-05-07 21:45:50.000000,17,52,34.26400005817413,463,false,4,5,,102.23000001907349,true,user_left_after_instructions,services__marketplace_/_stores__other,urgent

2022-09-01 20:40:22.000000,12,81,23.561400000254313,896,false,18,32,good,,,user_left_after_instructions,services__social_media__facebook__hacked_account,normal

2022-09-01 20:26:20.000000,11,52,22.460699963569642,842,false,10,13,,,,user_left_after_instructions,services__social_media__facebook__hacked_account,normal

2022-09-01 19:43:14.000000,7,68,16.65478571823665,3921,false,29,53,,,,not_solved,software__windows__windows_10__other,normal

2022-12-13 14:02:19.000000,18,46,27.86940007209778,1768,false,9,15,,,,user_left_after_instructions,services__social_media__facebook,normal

2022-11-11
17:02:55.000000,71,71,71.30799984931946,71,false,2,1,,,,user_left_before_instructions,other__no_info,normal

2023-05-07
21:56:08.000000,9,9,7.422999978065491,316,false,3,4,,,false,user_left_before_instructions,services__marketplace_/_stores__other,normal

2022-09-01
20:58:16.000000,35,35,34.81000018119812,35,false,1,1,,,,solved_no_review,services__social_media__facebook__hacked_account,normal

2023-05-07
21:42:56.000000,9,29,15.906749904155731,383,false,8,5,,,false,forwarded_to_3rd_party,services__internet_/_mobile_provider__boost_mobile,normal

2023-01-25
15:37:56.000000,14,84,32.673099994659424,1033,false,14,21,good,,,user_left_after_instructions,services__travel_services__airlines,normal

2022-11-11
16:19:58.000000,15,72,27.259235311956967,3386,false,19,25,,,,user_left_before_instructions,devices__pc_/_laptop__windows,normal

2022-11-11
17:16:49.000000,9,79,34.641400003433226,461,false,7,6,,,,solved_no_review,services__entertainment__peacock,normal

2023-05-07
22:03:16.000000,62,62,62.37299990653992,62,false,2,1,,,false,user_left_before_instructions,other__no_info,normal

2023-05-07
22:11:17.000000,7,47,17.05450001358986,487,false,9,11,,,false,user_left_after_instructions,services__payments_/_finances__paypal,normal

2023-05-08
06:35:32.000000,17,36,26.35350012779236,71,false,2,2,,,false,user_left_before_instructions,other__no_info,normal

2023-05-08
06:21:01.000000,9,33,17.30188881026374,461,false,12,17,,,false,user_left_before_instructions,services__social_media__facebook,normal

2023-05-07
22:12:59.000000,9,23,14.598499953746796,528,false,5,6,,,false,forwarded_to_3rd_party,services__streaming_services__dish,normal

2022-11-11
17:18:44.000000,16,59,31.06333335240682,462,false,8,8,,,,user_left_before_instructions,other__no_info,normal

2023-05-07
22:23:28.000000,19,56,29.37550002336502,223,false,5,4,,,false,user_left_before_instructions,services__internet_/_mobile_provider__other,normal

2023-05-07
22:19:30.000000,12,58,27.497999986012776,462,false,5,7,,,false,forwarded_to_3rd_party,services__marketplace_/_stores__other,normal

2023-05-07
22:05:43.000000,10,148,55.63366675376892,655,false,4,6,,,false,user_left_after_instructions,devices__streaming_devices__roku,normal

2023-01-25
15:15:31.000000,24,60,21.115480022430418,2381,false,26,37,good,,,,services__other,normal

2023-05-07
22:10:35.000000,8,62,32.24959998130798,541,false,6,10,bad,,false,user_left_before_instructions,devices__mobile_phone__ios,normal

2023-05-07
22:25:47.000000,42,141,78.66049998998642,598,false,5,8,,,false,user_left_after_instructions,devices__mobile_phone__ios,normal

2023-05-07
22:39:35.000000,18,36,26.655500054359436,151,false,2,4,,,false,user_left_after_instructions,services__internet_/_mobile_provider__verizon,normal

2023-05-07
22:37:01.000000,7,7,6.77400016784668,7,false,2,1,,,false,user_left_before_instructions,other__no_info,normal

2023-05-07
22:35:11.000000,13,37,23.66333333651225,226,false,3,4,,,false,user_left_before_instructions,other__no_info,normal

2023-05-07
22:31:45.000000,14,55,26.047833402951557,461,false,10,8,,39.98699998855591,true,user_left_before_instructions,other__no_info,normal

2022-11-11
17:22:57.000000,30,30,29.52999997138977,172,false,1,5,,,,user_left_before_instructions,devices__other,normal

2022-09-27
20:58:09.000000,25,27,23.943749964237213,253,false,7,6,,,,user_left_after_instructions,services__email_account__gmail,normal

2022-06-20 05:15:30.000000,,,,0,false,0,1,,,,maybe,other__no_info,normal

2023-05-27
14:24:03.000000,26,67,34.038500010967255,280,false,5,5,,,false,user_left_before_instructions,services__entertainment__paramount_,normal

2022-11-11
17:11:38.000000,22,46,23.846333318286472,782,false,10,12,,,,user_left_after_instructions,devices__pc_/_laptop__windows,normal

2022-09-27
21:16:19.000000,28,50,29.55042859486171,405,false,8,11,,,,user_left_after_instructions,services__entertainment__steam_giftcard,normal

2022-12-13
14:56:43.000000,7,7,7.346999883651733,7,false,2,1,,,,user_left_before_instructions,other__no_info,normal

2023-05-07
22:48:43.000000,20,20,19.591000080108643,20,false,2,1,,,false,user_left_before_instructions,services__apple_services__apple_tv,normal

2022-11-11
17:23:25.000000,11,11,11.384000062942505,11,false,2,1,,,,user_left_before_instructions,services__samsung_services,normal

2023-01-25
15:40:47.000000,9,35,18.3374286038535,763,false,9,14,,,,subscription/refund_related,services__other,normal

2022-09-27
21:13:58.000000,20,20,19.937999963760376,812,false,2,4,good,,,user_left_before_instructions,services__entertainment__other,normal

2022-06-03
23:25:10.000000,363,363,146.8710000038147,1360,false,13,7,,,,solved,services__email_account__gmail__password_recovery,normal

2022-09-27
20:57:55.000000,15,65,23.304500007629393,1120,false,12,17,,,,forwarded_to_3rd_party,services__uber_services__uber_eats,normal

2023-01-25
15:50:54.000000,10,10,8.672500014305115,42,false,4,3,,,,user_left_before_instructions,other__no_info,normal

2022-06-03
23:10:28.000000,488,488,185.59900005658469,713,false,5,6,,,,user_left_after_instructions,services__social_media__instagram__hacked_account,normal

2022-06-03
23:20:22.000000,623,623,171.64416674772897,1285,false,17,8,,,,user_left_after_instructions,hardware__tv_attachments__other__other,normal

2022-06-04
00:00:05.000000,9,65,31.593000014623005,171,false,6,4,,,,user_left_after_instructions,services__other__other,normal

2023-05-27
14:31:12.000000,15,166,78.00979995727539,1373,false,9,9,,,false,not_solved,devices__mobile_phone__ios,normal

2022-06-03
23:47:40.000000,71,71,70.86800003051758,71,false,1,1,,,,user_left_after_instructions,services__social_media__instagram__hacked_account,normal

2022-12-13
14:56:08.000000,14,23,18.742666562398274,227,false,4,4,,,,user_left_before_instructions,services__payments_/_finances__banking,normal

2022-09-27
20:59:49.000000,41,41,41.151000022888184,50,false,1,2,,,,user_left_after_instructions,services__marketplace_/_stores__wish,normal

2022-06-03
23:51:46.000000,11,41,30.85699995358785,241,false,3,5,,,,user_left_after_instructions,services__social_media__instagram__hacked_account,normal

2022-10-30
11:53:10.000000,,,,0,false,0,1,,,,solved,services__google_services__google_pay,normal

2022-09-27
21:14:08.000000,21,48,19.65014283997672,664,false,9,12,,,,user_left_before_instructions,services__other,normal

2023-01-25
15:55:46.000000,12,12,12.2260000705719,12,false,1,1,,,,user_left_before_instructions,other__no_info,normal

2022-06-03
23:48:30.000000,18,21,16.039333264033,92,false,3,4,,,,solved,hardware__tv_attachments__other__other,normal

2023-05-27
15:24:07.000000,30,30,29.961999893188477,37,false,1,2,,,false,user_left_after_instructions,services__uber_services__uber_taxi,normal

2022-06-03
23:58:17.000000,20,176,76.77900004386902,285,false,8,5,,,,maybe,services__entertainment__hulu__other,normal

2022-05-26
15:19:13.000000,9,10,9.4743332862854,187,false,4,4,,,,user_left_after_instructions,services__uber__uber_driver__account_registration,normal

2023-01-25
15:57:09.000000,32,32,31.79200005531311,32,false,1,1,,,,forwarded_to_3rd_party,services__social_media__facebook,normal

2022-06-03
23:47:23.000000,11,131,67.86859993934631,664,false,10,6,,,,maybe,other__no_info,normal

2022-06-03
23:56:26.000000,39,129,83.26599997282028,669,false,8,7,,,,solved,services__amazon__prime_membership__other,normal

2023-05-07
22:52:23.000000,8,40,18.785000026226044,202,false,5,5,,,,false,user_left_before_instructions,services__marketplace_/_stores__other,normal

2022-05-26
15:47:46.000000,22,263,72.11885717936924,909,false,10,10,,,,user_left_after_instructions,services__uber__uber_driver__other,normal

2022-05-26
15:02:26.000000,21,88,22.95736359466206,1451,false,26,27,,,,user_left_after_instructions,software__whatsup__recover_chat_history,normal

2022-05-26
15:59:14.000000,14,14,13.703999996185303,56,false,1,2,,,,user_left_after_instructions,hardware__other,normal

2022-05-24
15:52:06.000000,13,13,13.128000020980835,13,false,3,1,,,,maybe,services__google__youtube_tv__functionality,normal

2022-05-26
16:03:20.000000,78,78,43.99074989557266,405,false,11,5,,,,user_left_after_instructions,hardware__laptop__can_t_boot,normal

2022-09-27
21:27:11.000000,49,49,48.78800010681152,69,false,1,2,,,,user_left_before_instructions,services__microsoft__microsoft_account,normal

2023-01-25
15:48:06.000000,19,70,46.600599956512454,614,false,6,10,,,,user_left_before_instructions,services__other,normal

2022-05-26
16:02:08.000000,27,119,63.34099996089935,461,false,10,4,,,,user_left_after_instructions,services__other__other,normal

2022-05-26
15:58:33.000000,17,39,17.64159998893738,1249,false,10,15,,,,solved,services__other__other,normal

2022-05-26
16:02:59.000000,26,89,34.20763637802818,1025,false,11,18,,,,solved,other__not_related,normal