

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук
(повна назва)
Кафедра _____ Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський)

Розробка AI-системи класифікації, маршрутизації та автоматичного
опрацювання звернень клієнтів у месенджерах
(тема)

Виконав:
здобувач _____ четвертого _____ року навчання,
групи _____ ІТШ-22-2

_____ Ая ОТМАН
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми _____ освітньо-професійна
Освітня програма _____ Штучний інтелект
(повна назва освітньої програми)

Керівник _____ ас. Максим ПОЛІТ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ

(підпис)

_____ Лариса ЧАЛА
(власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____

Кафедра _____ Штучного інтелекту _____

Рівень вищої освіти _____ перший (бакалаврський) _____

Спеціальність _____ 122 Комп'ютерні науки _____
(код і повна назва)

Тип програми _____ освітньо-професійна _____

Освітня програма _____ Штучний інтелект _____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

« _____ » _____ 20 ____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві _____ Отман Аї Шаді _____
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка AI-системи класифікації, маршрутизації та автоматичного
опрацювання звернень клієнтів у месенджерах

затверджена наказом університету від 5 травня 2026 р. № 488Ст

2. Термін подання студентом роботи до екзаменаційної комісії 7 липня 2026 р.

3. Вихідні дані до роботи дані клієнтських звернень у месенджерах; мова програмування
Python; фреймворк FastAPI; бібліотеки scikit-learn, PyTorch та Transformers; СУБД PostgreSQL;
брокер повідомлень RabbitMQ; Telegram Bot API; платформа ngrok.

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі та постановка задачі

2) Методи обробки природної мови та класифікації текстів

3) Проєктування та розробка AI-системи

4) Тестування та аналіз результатів

КАЛЕНДАРНИЙ ПЛАН

[illegible]

Дата видачі завдання 8 червня 20 26 р.

Здобувач _____
(підпис)

Керівник роботи _____
(підпис)

ас. Максим ПОЛІЇТ
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 67 с., 12 рис., 10 табл., 1 дод., 20 джерел.

КЛАСИФІКАЦІЯ ТЕКСТІВ, МАРШРУТИЗАЦІЯ ЗВЕРНЕНЬ, МАШИННЕ НАВЧАННЯ, МЕСЕНДЖЕРИ, ОБРОБКА ПРИРОДНОЇ МОВИ, API, BERT, NLP, SVM, TF-IDF.

Об'єкт дослідження – процес автоматизованої класифікації та маршрутизації клієнтських звернень у месенджерах.

Предмет дослідження – методи обробки природної мови та алгоритми машинного навчання для класифікації неформальних текстів у системах клієнтської підтримки.

Мета роботи – проектування та реалізація AI-системи автоматичної класифікації, пріоритезації та маршрутизації звернень клієнтів, що надходять через месенджери.

Методи дослідження – методи обробки природної мови, векторне представлення текстів, багатокласова класифікація, трансформерні архітектури, порівняльний аналіз методів машинного навчання.

Проведено аналіз особливостей неформального тексту в месенджерах та типології клієнтських звернень. Розглянуто методи попередньої обробки тексту, векторизації та класифікації – від TF-IDF і Word2Vec до BERT і DistilBERT. Спроектовано модульну архітектуру системи, що включає пайплайн NLP-обробки, класифікатор на основі DistilBERT, модуль пріоритезації та маршрутизації і систему автоматичного опрацювання типових звернень. Обґрунтовано вибір DistilBERT як основної моделі класифікації та TF-IDF з логістичною регресією як резервного підходу.

ABSTRACT

Bachelor's thesis contains: 67 pp., 12 fig., 10 tabl., 1 ann., 20 references.

API, BERT, MACHINE LEARNING, MESSENGER PLATFORMS, NATURAL LANGUAGE PROCESSING, NLP, REQUEST ROUTING, SVM, TEXT CLASSIFICATION, TF-IDF.

Object of research – the process of automated classification and routing of customer requests received through messaging platforms.

Subject of research – natural language processing methods and machine learning algorithms for classification of informal text in customer support systems.

Purpose of the work – design and implementation of an AI system for automatic classification, prioritization and routing of customer requests received via messengers.

Research methods – natural language processing, text vector representation, multi-class classification, transformer architectures, comparative analysis of machine learning methods.

The characteristics of informal text in messengers and the typology of customer requests were analyzed. Text preprocessing, vectorization and classification methods were reviewed, from TF-IDF and Word2Vec to BERT and DistilBERT. A modular system architecture was designed, comprising an NLP processing pipeline, a DistilBERT-based classifier, a prioritization and routing module, and an automatic handling module for typical requests. The choice of DistilBERT as the primary classification model and TF-IDF with logistic regression as the fallback approach was justified.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз предметної галузі та постановка задачі.....	10
1.1 Огляд месенджерів як каналів комунікації	10
1.2 Типологія звернень клієнтів у месенджерах	11
1.3 Особливості неформального тексту в месенджерах	13
1.4 Аналіз існуючих систем обробки звернень	16
1.5 Постановка задачі та вимоги до системи.....	17
1.5.1 Функціональні вимоги.....	18
1.5.2 Нефункціональні вимоги.....	19
2 Методи обробки природної мови та класифікації текстів	21
2.1 Методи попередньої обробки тексту (NLP preprocessing).....	21
2.2 Методи векторизації тексту	24
2.2.1 TF-IDF та Bag of Words	24
2.2.2 Word2Vec, FastText.....	26
2.2.3 BERT-ембедінги.....	27
2.3 Методи ML для класифікації тексту	29
2.3.1 Класичні моделі (LR, SVM, NB, RF)	29
2.3.2 LSTM та рекурентні мережі.....	31
2.3.3 LSTM та рекурентні мережі.....	33
2.4 Вибір та обґрунтування методів NLP і ML	35
3 Проєктування та розробка ai-системи.....	38
3.1 Формування датасету та розмітка даних	38
3.1.1 Джерела даних та категорії класів	38
3.1.2 Баланс класів та підходи до анотування.....	39
3.2 Система пріоритезації та маршрутизації	41
3.3 Модуль автоматичного опрацювання типових звернень.....	43
3.4 Архітектура системи.....	45

3.4.1 Загальна структура та модульна декомпозиція	45
3.4.2 Шлюз повідомлень та адаптери месенджерів.....	47
3.4.3 Модуль NLP-обробки та класифікації	49
3.4.4 Модуль маршрутизації та зберігання даних	51
4 Тестування та аналіз результатів.....	54
4.1 Метрики оцінювання якості класифікації	54
4.2 Результати тестування класифікатора	55
4.3 Демонстрація роботи системи	57
4.4 Аналіз результатів та рекомендації.....	61
Висновки	63
Перелік джерел посилання	65
Додаток А Відомість кваліфікаційної роботи	67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

ШІ – штучний інтелект;

API – Application Programming Interface – інтерфейс програмування застосунків;

BERT – Bidirectional Encoder Representations from Transformers – двонаправлена модель представлення тексту на основі трансформерів;

BoW – Bag of Words – модель мішка слів;

CNN – Convolutional Neural Network – згорткова нейронна мережа;

FAQ – Frequently Asked Questions – часті запитання;

HTTP – Hypertext Transfer Protocol – протокол передачі гіпертексту;

JSON – JavaScript Object Notation – формат обміну даними;

LSTM – Long Short-Term Memory – мережа з довгою короткочасною пам'яттю;

ML – Machine Learning – машинне навчання;

NER – Named Entity Recognition – розпізнавання іменованих сутностей;

NLP – Natural Language Processing – обробка природної мови;

ORM – Object-Relational Mapping – об'єктно-реляційне відображення;

REST – Representational State Transfer – архітектурний стиль побудови API;

RNN – Recurrent Neural Network – рекурентна нейронна мережа;

SQL – Structured Query Language – мова структурованих запитів;

TF-IDF – Term Frequency – Inverse Document Frequency – частота терміна – обернена частота документа;

TTL – Time To Live – час життя запису;

URL – Uniform Resource Locator – уніфікований локатор ресурсів.

ВСТУП

Сучасний бізнес дедалі більше переносить клієнтську комунікацію у простір месенджерів. Telegram, WhatsApp, Instagram Direct та Facebook Messenger стали важливими каналами взаємодії між компаніями та споживачами, оскільки забезпечують швидкий та зручний обмін повідомленнями.

Разом зі зростанням кількості звернень виникає потреба в їх оперативній обробці. Повідомлення можуть стосуватися оформлення замовлень, уточнення характеристик товарів, статусу доставки, повернення продукції або вирішення проблемних ситуацій. Кожен тип звернення потребує різної реакції та може оброблятися різними підрозділами компанії.

Автоматизація цього процесу є складною через специфіку текстів у месенджерах. Користувачі часто використовують скорочення, розмовну лексику, емодзі та допускають орфографічні помилки. Крім того, повідомлення зазвичай є короткими та неструктурованими, що ускладнює їх аналіз.

Ще однією особливістю є різноманітність формулювань. Один і той самий намір може бути виражений десятками різних способів, тому система повинна визначати зміст повідомлення незалежно від його форми. Саме ця особливість робить задачу автоматичної класифікації звернень актуальною для сучасних компаній.

Існуючі програмні рішення пропонують засоби автоматизації роботи з клієнтськими повідомленнями, проте часто мають обмежені можливості адаптації до специфіки конкретного бізнесу. Тому актуальним є використання методів машинного навчання для побудови систем, здатних автоматично визначати тип звернення та підтримувати процес його подальшої обробки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд месенджерів як каналів комунікації

У сучасному бізнес-середовищі месенджери перетворились на один із ключових інструментів взаємодії між компаніями та їхніми клієнтами. На відміну від традиційних каналів (електронної пошти чи телефонних дзвінків) месенджери забезпечують миттєвий двосторонній зв'язок, що відповідає очікуванням сучасного споживача щодо швидкості та зручності обслуговування.

Серед найпоширеніших платформ, що використовуються у клієнтській комунікації, виділяють такі:

- Telegram: платформа з відкритим Bot API, що дозволяє будувати повноцінні автоматизовані системи обробки повідомлень. Підтримує текст, медіафайли, інлайн-кнопки та вебхуки, що робить його одним із найзручніших інструментів для інтеграції з AI-системами [1];

- WhatsApp: найбільш поширений месенджер у світі з аудиторією понад 2 мільярди користувачів. Через WhatsApp Business API компанії можуть автоматизувати обробку звернень, надсилати шаблонні повідомлення та вести діалоги у структурованому форматі [1];

- Instagram Direct: канал, орієнтований переважно на B2C-сегмент. Звернення через Instagram Direct часто пов'язані з інтересом до товарів або послуг, що рекламуються у стрічці, тому мають виражений комерційний характер;

- Facebook Messenger: платформа з розвиненою екосистемою ботів та інтеграцій. Messenger Platform API дозволяє отримувати та обробляти повідомлення в автоматичному режимі, підтримує persistent menu, quick replies та інші елементи взаємодії.

Незважаючи на відмінності в аудиторії та функціональності, усі перелічені платформи мають спільні характеристики з точки зору обробки

природної мови. По-перше, повідомлення в месенджерах відрізняються короткістю та фрагментарністю – клієнт рідко формулює звернення у вигляді розгорнутого тексту. По-друге, у таких повідомленнях активно використовуються розмовна лексика, скорочення, емодзі та виникають граматичні помилки, що ускладнює застосування стандартних NLP-методів без попередньої обробки [6]. По-третє, один і той самий тип звернення може бути сформульований принципово по-різному залежно від каналу, віку клієнта та контексту.

Саме ці особливості визначають вимоги до системи обробки звернень: вона має бути стійкою до шуму у вхідних даних, здатною розпізнавати намір клієнта (intent) незалежно від форми подачі, а також підтримувати уніфіковану обробку повідомлень з різних платформ через єдиний інтерфейс [12].

Месенджери формують специфічне середовище, в якому класичні підходи до аналізу тексту потребують адаптації, а автоматизована обробка звернень набуває особливої практичної цінності для бізнесу.

1.2 Типологія звернень клієнтів у месенджерах

На практиці клієнтської підтримки через месенджери можна виділити кілька стійких типів звернень, які відрізняються за метою, тональністю та очікуваною реакцією системи. Правильна типологізація є фундаментом для побудови системи класифікації, оскільки кожен тип звернення відповідає окремому класу у задачі багатокласової класифікації тексту [6], [12].

Перший і найпоширеніший тип – замовлення. Такі повідомлення містять явний намір придбати товар або уточнити деталі вже оформленої покупки. Як правило, вони включають назву товару, кількість або адресу доставки і є відносно структурованими порівняно з іншими типами.

Питання про товар або послугу – клієнт запитує характеристики, наявність, ціну або умови використання. Попри лаконічність

формулювань («а є розмір М?», «скільки коштує доставка до Харкова?»), такі повідомлення несуть чіткий комерційний намір і потребують швидкої реакції з боку відділу продажів [1].

Скарга – звернення з негативною тональністю, пов'язане з незадоволенням якістю товару, обслуговування або порушенням термінів. Цей тип вимагає пріоритетної обробки, оскільки несвоєчасна реакція підвищує ризик публічного негативу та втрати клієнта [6]. Скарги є одним із найскладніших типів для автоматичної класифікації через варіативність формулювань та емоційне забарвлення тексту.

Повернення та обмін – запит на повернення коштів або заміну товару. Часто супроводжується негативним або нейтральним емоційним забарвленням і потребує залучення відповідного відділу. Попри схожість зі скаргою, цей тип має чіткішу операційну мету – ініціювати конкретну процедуру.

Статус доставки – один із найчастіших типів звернень в e-commerce. Клієнт хоче дізнатись, де знаходиться його посилка або коли очікувати доставку. Цей тип добре піддається автоматизації за наявності інтеграції з логістичними системами і у більшості випадків не потребує залучення живого оператора [1].

Консультація – запит на пораду або рекомендацію без явного наміру купити прямо зараз. Потребує залучення менеджера або розгорнутої шаблонної відповіді. Лінгвістично такі повідомлення часто містять умовні конструкції: «порадьте що краще», «не знаю що вибрати між X та Y».

Технічна підтримка – звернення щодо проблем з використанням продукту або сервісу. Характеризується специфічною термінологією, детальним описом проблеми та потребою в покроковому вирішенні. Для автоматичної класифікації важливо відрізнити цей тип від скарги, оскільки маршрутизація та очікувана відповідь суттєво відрізняються [12].

Співпраця – запити від потенційних партнерів, постачальників або блогерів. Відрізняються від інших типів більш формальним стилем викладу

та специфічною лексикою («комерційна пропозиція», «умови співпраці», «партнерська програма») і маршрутизуються до керівництва або відповідального менеджера.

Систематизацію всіх типів звернень за ключовими параметрами – метою, тональністю, пріоритетом обробки та напрямком маршрутизації – наведено у таблиці 1.1.

Таблиця 1.1 – Типологія звернень клієнтів у месенджерах

Тип звернення	Типовий намір	Тональність	Пріоритет	Маршрут
Замовлення	Оформити покупку	Нейтральна	Середній	Відділ продажів
Питання про товар	Отримати інформацію	Нейтральна	Низький	Відділ продажів
Скарга	Висловити незадоволення	Негативна	Високий	Служба підтримки
Повернення /обмін	Повернути або замінити	Негативна/нейтральна	Високий	Служба підтримки
Статус доставки	Відстежити замовлення	Нейтральна	Низький	Автовідповідь/логістика
Консультація	Отримати пораду	Нейтральна	Середній	Менеджер
Технічна підтримка	Вирішити проблему	Негативна/нейтральна	Середній	Техпідтримка
Співпраця	Запропонувати партнерство	Формальна	Низький	Керівництво

Варто зазначити, що одне повідомлення може містити ознаки кількох типів одночасно – наприклад, скарга з одночасним запитом на повернення. Це ставить питання про доцільність підтримки мультилейбл-класифікації при проектуванні системи.

1.3 Особливості неформального тексту в месенджерах

Повідомлення, які надходять через месенджери, суттєво відрізняються від текстів, на яких традиційно навчаються NLP-моделі – новинних статей, відгуків на товари або офіційної кореспонденції. Ця відмінність має принципове значення для побудови системи класифікації,

оскільки безпосередньо впливає на вибір методів попередньої обробки та векторизації тексту.

Першою і найбільш очевидною особливістю є скорочення та спрощення синтаксису. Клієнти рідко пишуть повними реченнями – замість розгорнутого формулювання типове повідомлення зводиться до двох-трьох слів або навіть знаків питання. Відсутність підмета, присудка або розділових знаків є нормою, а не винятком. Це ускладнює застосування синтаксичного аналізу як допоміжного інструменту класифікації.

Другою особливістю є активне використання емодзі та спеціальних символів. Емодзі можуть як уточнювати тональність повідомлення, так і замінювати частину тексту, виступаючи невербальним індикатором наміру або емоційного стану клієнта. З точки зору NLP, емодзі є окремим типом токенів, які потребують або видалення, або спеціальної обробки залежно від обраної стратегії. Приклад типового повідомлення з характерним поєднанням скорочень, емодзі та неформального стилю наведено на рисунку 1.1.

Третьою особливістю є орфографічні та граматичні помилки. Вони виникають через швидкість набору на мобільному пристрої, автозаміну, діалектні особливості або недбалість. Помилки у написанні ключових слів можуть призводити до того, що токенізатор не розпізнає слово або присвоює йому некоректний вектор [7].

Четвертою особливістю є змішування мов та транслітерація. В україномовному сегменті клієнти нерідко перемикаються між українською та російською в межах одного повідомлення або використовують транслітерацію латиницею. Це потребує або мовної нормалізації на етапі попередньої обробки, або навчання моделі на даних, що відображають реальний розподіл мов у вхідному потоці [3].

П'ятою особливістю є контекстуальна неоднозначність коротких повідомлень. Фраза «не працює» без додаткового контексту може належати як до типу технічної підтримки, так і до скарги або питання. У реальному

діалозі модель часто отримує лише одне речення без попередньої історії розмови, що підвищує вимоги до якості класифікатора та обумовлює доцільність використання контекстних ембедінгів на основі трансформерних архітектур [11].

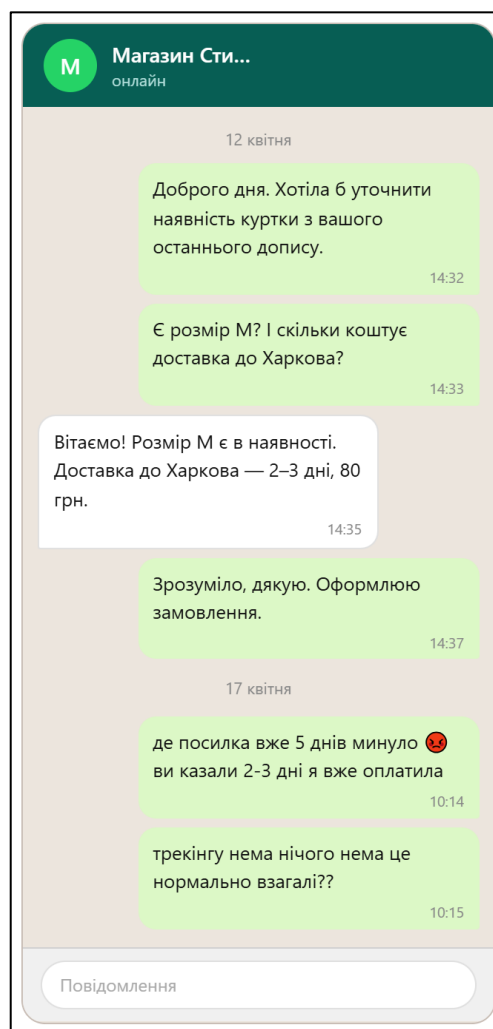


Рисунок 1.1 – Приклад повідомлень клієнта в месенджері з характерними ознаками неформального тексту

Сукупність перелічених особливостей визначає необхідність розробки спеціалізованого пайплайну попередньої обробки тексту, адаптованого до специфіки месенджер-комунікації, а не використання стандартних рішень загального призначення.

1.4 Аналіз існуючих систем обробки звернень

Ринок рішень для автоматизації клієнтської підтримки існує вже більше десяти років, і на ньому сформувалось кілька великих гравців. Перш ніж проєктувати власну систему, варто розібратись, що вже є і чому це не вирішує задачу [1].

Zendesk – одна з найпоширеніших корпоративних платформ підтримки. Підтримує інтеграцію з месенджерами, маршрутизацію тікетів за правилами і базову класифікацію за ключовими словами. Модуль Zendesk Intelligence додає ML-аналіз тональності та пріоритизацію [6]. Проблема одна: система проєктувалась для великих команд із розвиненими SLA-процесами. Для малого і середнього бізнесу це надлишково за функціоналом і дорого за ціною.

Intercom будується навколо концепції conversational support – поєднання живого чату, ботів і бази знань в одному інтерфейсі. AI-модуль Fin на базі великих мовних моделей закриває частину типових запитів автоматично [1]. Але архітектура повністю закрита: навчати модель на власних даних компанії неможливо, а вартість при зростанні навантаження стає суттєвою.

Freshdesk орієнтований на середній бізнес і за функціоналом близький до Zendesk. Модуль Freddy AI класифікує звернення і підказує агентам варіанти відповідей на основі схожих попередніх тікетів [7]. Але інтеграція з месенджерами реалізована через сторонні конектори – це затримка і додаткова точка відмови.

Порівняння платформ за ключовими параметрами наведено у таблиці 1.2. Спільний знаменник у всіх трьох – закрита архітектура і фіксована таксономія класів. Навчити модель на реальних повідомленнях свого бізнесу не вийде, адаптувати категорії під свої процеси – теж [6].

Таблиця 1.2 – Порівняння існуючих платформ обробки клієнтських звернень

Характеристика	Zendesk	Intercom	Freshdesk
Інтеграція з месенджерами	Нативна	Нативна	Через конектори
Навчання на власних даних	Ні	Ні	Ні
Гнучкість таксономії класів	Обмежена	Обмежена	Обмежена
Вартість при масштабуванні	Висока	Дуже висока	Середня
Відкритість архітектури	Закрита	Закрита	Закрита
Підтримка неформального тексту	Часткова	Часткова	Часткова

При цьому вартість API-інтеграції з месенджерами в комерційних платформах зростає пропорційно до обсягу повідомлень, що робить їх економічно не вигідними при серйозному навантаженні [7]. Саме ці обмеження й обумовлюють розробку власної системи з відкритою архітектурою, довільною таксономією та прямою інтеграцією з месенджерами [1].

1.5 Постановка задачі та вимоги до системи

Метою кваліфікаційної роботи є проєктування та реалізація AI-системи автоматичної класифікації, пріоритезації та маршрутизації клієнтських звернень, що надходять через месенджери. Для її досягнення вирішено такі задачі:

- аналіз особливостей неформального тексту в месенджерах та типологізація клієнтських звернень;
- дослідження методів попередньої обробки тексту та підходів до векторизації для коротких неформальних повідомлень;
- порівняльний аналіз класичних ML-моделей та трансформерних архітектур для багатокласової класифікації тексту;
- проєктування модульної архітектури системи з урахуванням функціональних та нефункціональних вимог;
- розробка механізмів пріоритезації, маршрутизації та автоматичного опрацювання типових звернень;

– формування методології складання та розмітки датасету для навчання класифікаційної моделі.

Об’єктом дослідження є процес автоматизованої класифікації та маршрутизації клієнтських звернень у месенджерах. Предметом дослідження є методи обробки природної мови та алгоритми машинного навчання для класифікації неформальних текстів у системах клієнтської підтримки.

1.5.1 Функціональні вимоги

Функціональні вимоги визначають, що система повинна робити – тобто описують її поведінку з точки зору користувача та бізнес-процесів, які вона автоматизує. Формування чітких функціональних вимог є необхідною передумовою для прийняття обґрунтованих архітектурних рішень на етапі проектування.

Система має забезпечувати приймання повідомлень із кількох месенджерів одночасно (Telegram, WhatsApp, Instagram Direct та Facebook Messenger) через уніфікований інтерфейс без прив’язки до конкретної платформи. Кожне вхідне повідомлення має оброблятися незалежно і не блокувати обробку інших.

Щодо класифікації та пріоритезації, система повинна:

- автоматично визначати тип звернення з-поміж визначених класів: замовлення, питання про товар, скарга, повернення, статус доставки, консультація, технічна підтримка, співпраця;
- визначати рівень пріоритету звернення (високий, середній або низький) на основі типу та тональності повідомлення;
- виявляти негативну тональність і автоматично підвищувати пріоритет таких звернень незалежно від їх типу.

Щодо маршрутизації, система повинна:

- автоматично спрямовувати звернення до відповідного відділу або відповідального менеджера на основі результату класифікації;
- підтримувати налаштування правил маршрутизації без необхідності зміни програмного коду;
- фіксувати факт маршрутизації та час передачі звернення у журналі подій.

Щодо автоматичного опрацювання, система повинна:

- надсилати шаблонні відповіді на типові звернення, що не потребують участі оператора;
- перевіряти статус замовлення через інтеграцію із зовнішніми логістичними системами та повертати результат клієнту в автоматичному режимі;
- формувати структуровану заявку на підставі отриманого повідомлення і передавати її у відповідну систему обліку.

Окремою функціональною вимогою є підтримка дообновлення моделі на нових розмічених даних без повного перенавчання з нуля, що дозволяє системі адаптуватись до змін у патернах комунікації з часом.

1.5.2 Нефункціональні вимоги

Нефункціональні вимоги описують не що система робить, а як вона це робить – тобто визначають якісні характеристики, яким має відповідати розроблене рішення. Ігнорування цих вимог на етапі проектування призводить до того, що система може коректно виконувати свої функції в лабораторних умовах, але виявитись непридатною до реального використання.

Ключовою вимогою до продуктивності є час відповіді системи. Класифікація та маршрутизація вхідного повідомлення має відбуватись у межах, що не створюють відчутної затримки для клієнта. Час класифікації одного повідомлення не повинен перевищувати 2 секунд при використанні

легких моделей або 5 секунд при використанні трансформерних архітектур. Пропускна здатність системи має становити не менше 100 повідомлень на хвилину без деградації точності, а час відновлення після збою – не більше 60 секунд.

Для оцінювання ефективності розроблюваної системи встановлюються вимоги до якості класифікації повідомлень. Зокрема, модель повинна забезпечувати загальну точність класифікації не менше 85%, значення показника Macro F1-score на рівні щонайменше 0,80, а також коректно розпізнавати високопріоритетні звернення типу «скарга» з точністю не нижче 90%. Крім того, після повторного навчання моделі її основні метрики повинні залишатися стабільними, а відхилення від базових показників не має перевищувати 3%.

Надійність та доступність системи є критичними параметрами для виробничого середовища. Система має забезпечувати доступність не нижче 99% в робочий час. Усі вхідні повідомлення мають зберігатись у черзі і не втрачатись у разі тимчасової недоступності класифікатора. Результати класифікації та маршрутизації мають логуватись у повному обсязі для подальшого аудиту та дообнавлення моделі на реальних даних.

Щодо масштабованості, архітектура системи має допускати горизонтальне масштабування без зміни програмного коду – збільшення навантаження має вирішуватись додаванням екземплярів сервісу, а не переробкою логіки.

Вимоги до безпеки визначаються як характером оброблюваних даних – повідомлення клієнтів містять персональну інформацію – так і регуляторними вимогами. Усі повідомлення мають передаватись і зберігатись у зашифрованому вигляді. Доступ до адміністративного інтерфейсу та журналів подій має бути розмежований за ролями. Персональні дані клієнтів не мають використовуватись як ознаки при навчанні моделі – для цього на етапі формування датасету застосовується деперсоналізація вхідних текстів.

2 МЕТОДИ ОБРОБКИ ПРИРОДНОЇ МОВИ ТА КЛАСИФІКАЦІЇ ТЕКСТІВ

2.1 Методи попередньої обробки тексту (NLP preprocessing)

Попередня обробка тексту є обов'язковим етапом будь-якої NLP-системи, оскільки якість вхідних даних безпосередньо визначає точність класифікаційної моделі. Особливо це актуально для текстів із месенджерів, які характеризуються підвищеним рівнем шуму, неформальністю та орфографічними помилками. Пайплайн попередньої обробки являє собою послідовність трансформацій, кожна з яких усуває певний тип небажаних артефактів або приводить текст до нормалізованого вигляду, придатного для подальшої векторизації.

Першим кроком є очищення тексту. На цьому етапі з повідомлення видаляються або замінюються елементи, що не несуть семантичного навантаження: URL-посилання, номери телефонів, зайві пробіли, спеціальні символи та HTML-теги, які можуть траплятися у повідомленнях, скопійованих із браузера. Окремим питанням є обробка емодзі – залежно від обраної стратегії вони або видаляються, або конвертуються у текстові описи, що дозволяє зберегти тональну інформацію повідомлення [12].

Другим кроком є токенизація – розбиття тексту на окремі одиниці (токени), якими можуть бути слова, підслова або символи. Для українськомовних текстів із месенджерів токенизація ускладнюється відсутністю пробілів між словами в деяких повідомленнях, злипанням знаків пунктуації зі словами та використанням дефісів у складних словах. Найпоширенішим підходом є пробільна токенизація з додатковим розбиттям за знаками пунктуації, яку реалізує, зокрема, бібліотека NLTK.

Третім кроком є нормалізація регістру – приведення всього тексту до нижнього регістру. Це усуває відмінність між варіантами написання одного

й того ж слова як різними токенами, що суттєво скорочує розмір словника [6].

Четвертим кроком є видалення стоп-слів – службових слів, які зустрічаються у більшості текстів і не несуть розрізняювальної інформації для класифікації. Стандартні списки стоп-слів потребують адаптації для месенджер-комунікації, оскільки деякі слова, що є стоп-словами у книжному тексті, можуть бути значущими у контексті звернень [7].

П'ятим кроком є стемінг або лематизація – приведення слова до його кореневої або словникової форми. Стемінг відсікає закінчення за набором правил, тоді як лематизація повертає словникову форму з урахуванням морфології. Для української мови лематизація є більш точним підходом, оскільки мова характеризується розвиненою флективною системою [6]. Порівняння цих двох підходів наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння стемінгу та лематизації

Характеристика	Стемінг	Лематизація
Принцип роботи	Відсікання закінчень за правилами	Зведення до словникової форми
Точність	Нижча, можливі псевдокорені	Вища, лінгвістично коректна
Швидкість	Висока	Нижча (потребує словника)
Приклад	«доставив» → «достав»	«доставив» → «доставити»
Підходить для	Великих корпусів, швидких рішень	Морфологічно багатих мов (укр.)

Завершальним кроком стандартного пайплайну є обробка орфографічних помилок. Для текстів із месенджерів цей крок має особливе значення – слова «достака», «замовлення», «сьогодні» не будуть розпізнані як відомі одиниці словника жодним стандартним токенизатором, що призводить до втрати семантичної інформації або некоректного векторного представлення токена.

Автоматичне виправлення помилок може бути реалізоване кількома способами. Найпоширенішим є використання алгоритму відстані Левенштейна, який обчислює мінімальну кількість операцій редагування

між двома рядками і дозволяє знайти найближче коректне слово у словнику. Альтернативою є статистичні мовні моделі, які враховують контекст і здатні розрізняти омофони та випадки автозаміни. У практичних реалізаціях застосовуються спеціалізовані бібліотеки корекції тексту, які поєднують обидва підходи [7].

Варто зазначити, що агресивна автокорекція може спотворити значення повідомлення – наприклад, виправити навмисне скорочення або сленгове слово, яке несе змістове навантаження. Тому на практиці виправлення орфографічних помилок часто виконується вибірково або з порогом впевненості, нижче якого слово залишається без змін. Послідовність усіх етапів пайплайну попередньої обробки наведена на рисунку 2.1.



Рисунок 2.1 – Пайплайн попередньої обробки тексту в системі класифікації звернень

Важливо зазначити, що склад та порядок кроків пайплайну не є фіксованим і визначається конкретною архітектурою системи. Зокрема, при

використанні трансформерних моделей на кшталт BERT токенизація та нормалізація виконуються вбудованим токенизатором моделі, а окремі кроки стемінгу або видалення стоп-слів можуть бути надлишковими або навіть шкідливими для якості класифікації.

2.2 Методи векторизації тексту

2.2.1 TF-IDF та Bag of Words

Модель «мішок слів» (Bag of Words, BoW) є найпростішим підходом до векторного представлення тексту. Основна ідея полягає у тому, що документ описується як набір слів без урахування їх порядку – кожне унікальне слово словника відповідає одному виміру вектора, а значення вимірів відображають частоту появи слова у документі [6].

Формально, якщо словник містить n унікальних слів, кожен документ d представляється вектором розмірності n , як наведено у формулі (2.1):

$$d = (f_1, f_2, \dots, f_i), \quad (2.1)$$

де f_i – кількість входжень i -го слова у документі.

Головний недолік BoW полягає у тому, що часті слова, які зустрічаються в більшості документів, отримують високі значення, хоча несуть мало розрізнявальної інформації. Саме для усунення цього недоліку було розроблено метод TF-IDF (Term Frequency – Inverse Document Frequency).

TF-IDF зважає кожне слово за двома критеріями одночасно – частотою слова у конкретному документі (TF) та оберненою частотою документів, у яких це слово зустрічається (IDF). Слова, що трапляються рідко і лише у певних документах, отримують вищу вагу, тоді як слова,

поширені по всьому корпусу, – нижчу. Частота терміна розраховується за формулою (2.2):

$$TF(t, d) = \frac{c(t, d)}{|d|}, \quad (2.2)$$

де $c(t, d)$ – кількість входжень терміна t у документі d ;

$|d|$ – загальна кількість слів у документі.

Обернена частота документів визначається за формулою (2.3):

$$IDF(t) = \log \left(\frac{N}{df(t)} \right), \quad (2.3)$$

де N – загальна кількість документів у корпусі;

$df(t)$ – кількість документів, що містять термін t .

Кінцева вага терміна обчислюється як добуток значень (2.2) та (2.3), що наведено у формулі (2.4):

$$TF-IDF(t, d) = TF(t, d) \times IDF(t). \quad (2.4)$$

Перевагою TF-IDF є простота реалізації, висока швидкість обчислень та інтерпретованість отриманих ваг. Метод добре працює на задачах класифікації з чіткими ключовими словами – наприклад, слово «повернення» з високою ймовірністю вказує на відповідний тип звернення незалежно від контексту [6].

Серед обмежень методу – повне ігнорування порядку слів і семантичних зв'язків між ними. Два речення «товар не сподобався» і «товар сподобався» матимуть майже ідентичні вектори, хоча несуть протилежний зміст. Крім того, TF-IDF не здатний узагальнювати значення між синонімами – «повернення» і «повернути» розглядаються як різні терміни, якщо попередньо не застосовано лематизацію [7].

2.2.2 Word2Vec, FastText

Методи Word2Vec та FastText належать до класу дистрибутивних моделей представлення слів, які на відміну від TF-IDF враховують контекст слова у тексті. Основна гіпотеза, на якій базуються ці підходи, сформульована Джон Руперт Ферт: слова, що зустрічаються в схожих контекстах, мають схоже значення [3].

Word2Vec, запропонований компанією Google у 2013 році, навчає щільні векторні представлення слів (ембедінги) на великих текстових корпусах [3]. Кожне слово відображається у вектор фіксованої розмірності – як правило, від 100 до 300 вимірів – таким чином, що семантично близькі слова опиняються поруч у векторному просторі. Модель реалізується у двох архітектурних варіантах.

Перший – CBOW (Continuous Bag of Words) – передбачає слово за його контекстом. Якщо задано вікно контексту розміром 2, то для речення «клієнт хоче повернути товар» модель намагається передбачити слово «повернути» за словами «хоче» та «товар». Другий варіант – Skip-gram – працює у зворотному напрямку: за центральним словом передбачаються контекстні слова. Skip-gram краще справляється з рідкісними словами, тоді як CBOW є швидшим при навчанні [3].

Ключовою перевагою Word2Vec є здатність до узагальнення. Векторні операції над ембедінгами дають семантично осмислені результати – класичний приклад наведено у формулі (2.5):

$$v(\text{«король»}) - v(\text{«чоловік»}) + v(\text{«жінка»}) \approx v(\text{«королева»}). \quad (2.5)$$

Проте Word2Vec має суттєве обмеження – він будує єдиний вектор для кожного слова незалежно від контексту, а також не здатний обробляти слова, що не зустрічались у навчальній вибірці (out-of-vocabulary

слова). Для месенджер-комунікації це критично, оскільки такі слова як «доставка» або «замовлення» будуть просто ігноруватись моделлю.

Саме цю проблему вирішує FastText, розроблений організацією Facebook AI Research у 2017 році [8]. Ключова відмінність полягає у тому, що FastText представляє кожне слово не як єдиний токен, а як суму векторів його символічних n -грам. Наприклад, слово «замовлення» при $n = 3$ розбивається на підрядки: «зам», «амо», «мов», «овл», «вле», «лен», «енн», «ння» та інші.

Вектор слова обчислюється як сума векторів усіх його n -грам за формулою (2.6):

$$v(w) = \sum_{m \in M(w)} z_m, \quad (2.6)$$

де $M(w)$ – множина n -грам слова w ;

z_m – вектор відповідної n -грами.

Завдяки цьому FastText здатний будувати осмислені вектори навіть для слів із помилками або нових слів, яких не було у навчальній вибірці, оскільки їх символічні n -грами з високою ймовірністю вже зустрічались у корпусі [8]. Для задачі класифікації звернень із месенджерів ця властивість є особливо цінною з огляду неформальності тексту.

2.2.3 BERT-ембедінги

BERT (Bidirectional Encoder Representations from Transformers) – це мовна модель, запропонована компанією Google у 2018 році, яка принципово змінила підхід до отримання векторних представлень тексту [2]. На відміну від Word2Vec та FastText, які будують статичний вектор для кожного слова незалежно від контексту, BERT генерує динамічні контекстні ембедінги – вектор слова змінюється залежно від речення, у якому воно вживається.

Архітектура BERT базується на механізмі самоуваги (self-attention) трансформера, що дозволяє моделі враховувати зв'язки між усіма словами речення одночасно в обох напрямках – як зліва направо, так і справа наліво [11]. Завдяки цьому слово «не прийшло» у реченні «замовлення не прийшло» та у реченні «повідомлення не прийшло» отримуватиме різні векторні представлення, що відображають різний семантичний контекст.

Модель проходить два етапи. На етапі попереднього навчання BERT навчається на великих текстових корпусах за двома задачами: передбачення замаскованих слів у реченні та визначення, чи є два речення послідовними у тексті. На етапі донавчання (fine-tuning) попередньо навчена модель адаптується до конкретної прикладної задачі – у даному випадку класифікації звернень – на відносно невеликому розміченому датасеті.

Вхідна послідовність tokenів подається у модель у вигляді суми трьох типів ембедінгів за формулою (2.7):

$$E(x) = E_{token}(x) + E_{segment}(x) + E_{position}(x), \quad (2.7)$$

де E_{token} – ембедінг токена зі словника;

$E_{segment}$ – ембедінг, що визначає належність токена до першого чи другого речення;

$E_{position}$ – позиційний ембедінг, що кодує порядок токена у послідовності.

Для задач класифікації використовується вектор спеціального токена $[CLS]$, який стоїть на початку кожної вхідної послідовності. Після проходження через усі шари трансформера цей вектор акумулює контекстну інформацію всього речення і подається на вхід класифікаційного шару.

Практичним спрощенням BERT є DistilBERT, який зберігає близько 97% якості оригінальної моделі при вдвічі меншому розмірі та приблизно на 60% вищій швидкості інференсу [5]. Для систем реального

часу, де затримка відповіді є критичним параметром, DistilBERT є більш прагматичним вибором порівняно з повною версією BERT.

2.3 Методи ML для класифікації тексту

2.3.1 Класичні моделі (LR, SVM, NB, RF)

Класичні методи машинного навчання залишаються актуальними для задач класифікації тексту завдяки своїй простоті, швидкості навчання та інтерпретованості результатів. У контексті класифікації звернень вони, як правило, застосовуються у поєднанні з TF-IDF векторизацією і слугують базовими моделями (baseline) для порівняння з більш складними підходами [7].

Logistic Regression (Logistic Regression, LR) – лінійний класифікатор, який моделює ймовірність належності документа до певного класу через сигмоїдну функцію. Для багатокласової задачі використовується узагальнення – softmax-функція, що розраховується за формулою (2.8):

$$P(y = k | x) = \frac{\exp(w_k^T x)}{\sum_j \exp(w_j^T x)}, \quad (2.8)$$

де x – вектор ознак документа;

w_k – вектор ваг для класу k .

Логістична регресія добре працює на лінійно розділюваних даних і є стійкою до перенавчання при правильній регуляризації.

Support Vector Machine (SVM) шукає гіперплощину у просторі ознак, яка максимізує відступ між класами. Для нелінійно розділюваних даних застосовується kernel trick – перетворення простору ознак у вищовимірний, де класи стають лінійно розділюваними. Оптимізаційна задача SVM формулюється як формула 2.9:

$$\min \frac{\|w\|^2}{2} + C \sum_i \xi_i, \quad (2.9)$$

де C – параметр регуляризації;

ξ_i – змінні нежорсткості (slack variables), що допускають порушення межі класифікації.

SVM традиційно демонструє високу точність на текстових задачах завдяки ефективній роботі у високовимірних просторах, характерних для TF-IDF представлень.

Naive Bayes classifier (Naive Bayes, NB) базується на теоремі Байєса з припущенням про умовну незалежність ознак. Клас документа визначається за формулою (2.10):

$$\hat{y} = \arg \max_k P(y = k) \times \prod_i P(x_i | y = k), \quad (2.10)$$

де $P(y = k)$ – апіорна ймовірність класу;

$P(x_i | y = k)$ – ймовірність появи ознаки x_i за умови класу k .

Незважаючи на спрощене припущення про незалежність ознак, NB демонструє конкурентоспроможну точність на задачах класифікації тексту, а також вирізняється надзвичайно високою швидкістю навчання та інференсу.

Random Forest (RF) є ансамблевим методом, що будує множину дерев рішень на випадкових підвибірках навчального набору та випадкових підмножинах ознак. Фінальне рішення приймається голосуванням (формула 2.11):

$$\hat{y} = \arg \max_k \sum_t \mathbb{1}[h_t(x) = k], \quad (2.11)$$

де $h_t(x)$ – передбачення t -го дерева;

T – загальна кількість дерев.

Випадковий ліс стійкий до перенавчання та здатний моделювати нелінійні залежності, проте значно поступається SVM та логістичній регресії за швидкістю на великих розріджених TF-IDF матрицях.

Порівняльні характеристики розглянутих моделей наведено у таблиці 2.2.

Таблиця 2.2 – Порівняння класичних моделей класифікації тексту

Модель	Інтерпретованість	Швидкість навчання	Робота з високовимірними даними
Логістична регресія	Висока	Висока	Добра
SVM	Низька	Середня	Відмінна
Naive Bayes	Середня	Дуже висока	Добра
Random Forest	Середня	Низька	Задовільна

З наведених даних видно, що жодна з моделей не є універсально найкращою – вибір залежить від конкретних вимог системи. Якщо пріоритетом є швидкість і простота розгортання, перевагу варто надати Naive Bayes або логістичній регресії. Якщо ж критично важлива точність класифікації на складних і перекривних класах, SVM з відповідно підібраним ядром демонструє стабільно вищі результати на текстових задачах.

2.3.2 LSTM та рекурентні мережі

Recurrent Neural Network (RNN) були розроблені для обробки послідовних даних – на відміну від класичних моделей, вони враховують порядок слів у тексті. Базова RNN обробляє токени послідовно, на кожному кроці оновлюючи прихований стан за формулою (2.12):

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b), \quad (2.12)$$

де h_t – прихований стан на кроці t ;

x_t – вхідний токен;

W_h та W_x – матриці ваг;

b – вектор зміщення.

Прихований стан акумулює інформацію про всі попередні токени і теоретично дозволяє моделі враховувати довгострокові залежності у тексті.

Однак на практиці базова RNN страждає від проблеми зникаючого градієнта – при навчанні через довгі послідовності градієнти стають настільки малими, що модель фактично перестає навчатись на ранніх токенах. Це означає, що інформація з початку повідомлення може не дійти до кінця при класифікації.

Для вирішення цієї проблеми була запропонована архітектура Long Short-Term Memory (LSTM) [4]. LSTM вводить механізм вентилів, що регулюють потік інформації через мережу. Архітектура містить три типи вентилів та окремий стан пам'яті c_t :

- вентиль забування f_t : визначає, яку частину попереднього стану пам'яті слід відкинути;

- вентиль входу i_t : визначає, яка нова інформація записується у стан пам'яті;

- вентиль виходу o_t : визначає, яка частина стану пам'яті формує прихований стан.

Оновлення стану пам'яті відбувається за формулою (2.13):

$$c_t = f_t \odot c_{t-1} + i_t \odot g(x_t, h_{t-1}), \quad (2.13)$$

де $\odot$ – поелементне множення;

$g(\cdot)$ – функція кандидата нових значень пам'яті.

Прихований стан на виході формується як формула 2.14:

$$h_t = o_t \odot \tanh(c_t). \quad (2.14)$$

Завдяки механізму вентилів LSTM здатна зберігати релевантну інформацію протягом довгих послідовностей і ефективно ігнорувати нерелевантну. Для задачі класифікації звернень це означає, що модель може коректно обробляти довгі повідомлення, де ключовий сигнал – наприклад, слово «повернення» – з’являється наприкінці речення.

На практиці для текстової класифікації часто використовують двонаправлену LSTM (Bidirectional LSTM, BiLSTM), яка обробляє послідовність одночасно у прямому та зворотному напрямках. Це дозволяє моделі враховувати як лівий, так і правий контекст кожного токена, що підвищує якість класифікації на коротких неформальних повідомленнях.

Попри переваги, LSTM поступається трансформерним моделям за точністю на більшості сучасних NLP-задач і вимагає значно більше часу на навчання порівняно з класичними методами. Проте у сценаріях з обмеженими обчислювальними ресурсами або невеликим розміченим датасетом LSTM залишається обґрунтованою альтернативою [4].

2.3.3 LSTM та рекурентні мережі

Трансформерна архітектура, запропонована у роботі Ашиш Васвані та співавторів у 2017 році, кардинально змінила підхід до задач обробки природної мови [11]. На відміну від рекурентних мереж, трансформер обробляє всю послідовність токенів паралельно, а залежності між словами моделюються через механізм уваги (attention), а не через послідовне накопичення прихованого стану.

Ключовим елементом архітектури є механізм scaled dot-product attention, що обчислюється за формулою (2.15):

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (2.15)$$

де Q – матриця запитів (queries);

K – матриця ключів (keys);

V – матриця значень (values);

d_k – розмірність ключів.

Ділення на $\sqrt{d_k}$ стабілізує градієнти при великих значеннях розмірності. Механізм уваги дозволяє кожному токеноу зважено звертатись до будь-якого іншого токена послідовності незалежно від відстані між ними.

BERT (Bidirectional Encoder Representations from Transformers) використовує лише енкодерну частину трансформера і навчається на двох задачах одночасно. Перша – Masked Language Model (MLM) – передбачає випадково замасковані токени у реченні. Друга – Next Sentence Prediction (NSP) – визначає, чи є два речення послідовними у тексті. Завдяки такому попередньому навчанню на великих корпусах BERT формує глибокі контекстні представлення, які потім адаптуються до конкретної задачі через fine-tuning [2].

Для задачі класифікації до виходу токена $[CLS]$ додається лінійний шар, і модель донавчається на розміченому датасеті. Передбачення класу обчислюється як формула 2.16:

$$\hat{y} = \text{softmax}(W \cdot h_{[CLS]} + b), \quad (2.16)$$

де $h_{[CLS]}$ – вихідний вектор токена $[CLS]$;

W та b – параметри класифікаційного шару.

DistilBERT є дистильованою версією BERT, отриманою методом knowledge distillation – навчання меншої моделі на виходах більшої [5]. Студентська модель містить вдвічі менше шарів трансформера, що забезпечує приблизно на 60% вищу швидкість інференсу при збереженні близько 97% якості оригінальної моделі. Для систем обробки звернень у реальному часі, де затримка відповіді є критичним параметром, це робить DistilBERT більш практичним вибором.

Головним обмеженням трансформерних моделей є висока обчислювальна вартість навчання та інференсу, а також потреба у значному обсязі розмічених даних для ефективного fine-tuning. Крім того, стандартний BERT обмежений послідовностями до 512 токенів, що для коротких повідомлень із месенджерів не є суттєвою проблемою, однак може бути обмеженням при обробці розгорнутих звернень із вкладеними документами.

2.4 Вибір та обґрунтування методів NLP і ML

Вибір методів обробки тексту та класифікаційної моделі є ключовим архітектурним рішенням, яке визначає точність системи, вимоги до обчислювальних ресурсів та складність підтримки. Для обґрунтованого вибору необхідно розглянути наявні підходи в контексті конкретних вимог задачі – коротких неформальних текстів із месенджерів, восьми класів із нерівномірним розподілом та вимоги до швидкості інференсу не більше 2–5 секунд на повідомлення.

На етапі векторизації тексту основною альтернативою є вибір між TF-IDF та нейронними ембедінгами. TF-IDF у поєднанні з класичними ML-моделями забезпечує високу швидкість навчання та інференсу, інтерпретованість результатів і низькі вимоги до обчислювальних ресурсів. Проте для коротких повідомлень із месенджерів цей підхід має суттєве обмеження – він не здатний розрізняти семантично близькі, але по-різному сформульовані повідомлення. Нейронні ембедінги на основі BERT або FastText позбавлені цього недоліку, але вимагають значно більше ресурсів [2].

З урахуванням зазначених міркувань для даної системи пропонується двоетапний підхід. На першому етапі навчається та оцінюється набір базових моделей на TF-IDF представленнях – Logistic Regression, Support Vector Machine та Naive Bayes classifier – з метою визначення baseline-

точності та виявлення найбільш проблемних класів. На другому етапі виконується донавчання DistilBERT на розміченому датасеті, результати порівнюються з baseline за метриками точності, повноти та F1-score у розрізі кожного класу. Порівняння підходів за ключовими критеріями наведено у таблиці 2.3.

Таблиця 2.3 – Порівняння підходів до класифікації звернень

Критерій	TF-IDF + LR/SVM	FastText	DistilBERT
Точність на коротких текстах	Середня	Висока	Дуже висока
Швидкість інференсу	Дуже висока	Висока	Середня
Вимоги до розміру датасету	Низькі	Середні	Високі
Стійкість до помилок у тексті	Низька	Висока	Висока
Інтерпретованість	Висока	Низька	Низька
Вимоги до обчислювальних ресурсів	Мінімальні	Помірні	Значні

Обґрунтуванням на користь DistilBERT як основної моделі є кілька факторів. По-перше, повідомлення у месенджерах характеризуються високою варіативністю формулювань при незмінному намірі – саме контекстні ембедінги дозволяють коректно обробляти таку варіативність. По-друге, стійкість до орфографічних помилок є критичною вимогою для даної задачі, і трансформерний токенізатор на основі WordPiece частково вирішує цю проблему на рівні підслів. По-третє, DistilBERT забезпечує прийнятну швидкість інференсу при розгортанні на GPU, що відповідає нефункціональній вимозі щодо часу відповіді системи [5].

TF-IDF у поєднанні з логістичною регресією залишається у системі як резервна модель – вона використовується у випадку недоступності основного класифікатора або при обробці повідомлень, для яких DistilBERT повертає низьку впевненість передбачення. Такий гібридний підхід підвищує загальну надійність системи без суттєвого збільшення складності архітектури.

Вибір функції втрат при навчанні DistilBERT визначається дисбалансом класів у датасеті. Замість стандартної крос-ентропії використовується зважена крос-ентропія, де вага кожного класу обернено

пропорційна його частоті у навчальній вибірці, що визначається за формулою (2.17):

$$w_k = \frac{N}{K \times n_k}, \quad (2.17)$$

де N – загальна кількість прикладів у навчальній вибірці;

K – кількість класів;

n_k – кількість прикладів класу k .

Функція втрат при цьому набуває вигляду (2.18):

$$L = - \sum_k w_k \times y_k \times \log(\hat{p}_k), \quad (2.18)$$

де y_k – істинна мітка класу k ;

$\hat{p}_k$ – передбачена моделлю ймовірність класу k .

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА AI-СИСТЕМИ

3.1 Формування датасету та розмітка даних

3.1.1 Джерела даних та категорії класів

Якість навчальної вибірки є визначальним фактором для точності класифікаційної моделі. Для задачі класифікації звернень клієнтів у месенджерах датасет має відображати реальний розподіл типів повідомлень, характерний для конкретного бізнесу, а не загальні лінгвістичні патерни. Саме тому основним джерелом даних є історичні повідомлення клієнтів, накопичені у системах підтримки або безпосередньо в архівах месенджерів [6].

Формування датасету може відбуватись із кількох джерел. Першим і найціннішим є архів реальних діалогів із клієнтами, експортований із підключених месенджерів. Такі дані відображають автентичний стиль комунікації, характерний для конкретної аудиторії, і містять усі особливості неформального тексту – скорочення, помилки, змішування мов. Другим джерелом є тікети з існуючих систем підтримки – Zendesk, Freshdesk або подібних – якщо компанія використовувала такі інструменти раніше. Третім джерелом є синтетичні дані, згенеровані вручну або за допомогою мовних моделей для поповнення малочисельних класів.

Категорії класів відповідають типології звернень, розглянутій у першому розділі роботи. Кожне повідомлення у датасеті отримує одну з таких міток:

- замовлення: повідомлення з явним наміром оформити покупку або уточнити деталі існуючого замовлення;
- питання про товар: запити щодо характеристик, наявності або ціни;
- скарга: повідомлення з негативною тональністю щодо якості товару або обслуговування;

- повернення або обмін: запити на ініціювання відповідної процедури;
- статус доставки: запити щодо місцезнаходження або термінів доставки замовлення;
- консультація: запити на пораду без явного комерційного наміру;
- технічна підтримка: повідомлення щодо проблем з використанням продукту або сервісу;
- співпраця – запити від потенційних партнерів або постачальників.

Загальний обсяг датасету для навчання моделей рівня BERT має становити не менше 500–1000 прикладів на клас, тоді як для класичних ML-моделей у поєднанні з TF-IDF достатньо 200–300 прикладів на клас [7]. При цьому важливо забезпечити не лише кількісний, але й якісний баланс – кожен клас має містити різноманітні формулювання, а не однотипні повідомлення, що призведе до перенавчання на поверхневих патернах.

Окремою проблемою є мультилейбл-випадки, коли одне повідомлення одночасно належить до кількох класів. На етапі розмітки такі повідомлення або отримують мітку домінантного класу за судженням анотатора, або виділяються в окрему категорію для подальшого аналізу. Вибір стратегії залежить від архітектурного рішення щодо підтримки мультилейбл-класифікації у системі [4].

3.1.2 Баланс класів та підходи до анотування

Реальні датасети звернень клієнтів характеризуються вираженим дисбалансом класів. У типовому e-commerce бізнесі запити про статус доставки та питання про товар складають значну частку усіх повідомлень, тоді як звернення щодо співпраці або технічної підтримки трапляються відносно рідко. Навчання моделі на незбалансованих даних призводить до того, що класифікатор схильний передбачати мажоритарні класи навіть там, де повідомлення фактично належить до рідкісного класу [6].

Для усунення дисбалансу застосовуються кілька підходів. Перший – надвибірка (oversampling) міноритарних класів, при якій приклади рідкісних класів дублюються або доповнюються синтетичними варіантами. Найпоширенішим методом синтетичної надвибірки є SMOTE (Synthetic Minority Oversampling Technique), проте для текстових даних більш ефективним є перефразування існуючих прикладів за допомогою мовних моделей. Другий підхід – підвибірка (undersampling) мажоритарних класів, при якій частина прикладів домінантних класів вилучається з навчальної вибірки. Третій підхід – зважування класів при навчанні, коли функція втрат штрафує помилки на міноритарних класах сильніше, що не вимагає зміни самого датасету [7].

На практиці найкращі результати дає комбінація помірної надвибірки міноритарних класів із зважуванням при навчанні, що дозволяє уникнути як перенавчання на синтетичних прикладах, так і недостатньої уваги до рідкісних класів.

Анотування датасету є трудомістким, але критично важливим етапом. Основним методом є ручна розмітка, при якій кожному повідомленню людиною-анотатором присвоюється мітка класу. Для забезпечення якості розмітки застосовуються такі принципи:

- кожне повідомлення розмічається незалежно двома анотаторами, після чого розбіжності вирішуються третьою стороною;
- анотатори отримують детальний гайдлайн із прикладами граничних випадків для кожного класу;
- для оцінки узгодженості між анотаторами використовується коефіцієнт каппа Коена, прийнятним значенням якого вважається $\kappa \geq 0.7$;
- повідомлення, що не вписуються в жодну категорію, виносяться в окремий клас «інше» і не включаються до навчальної вибірки без додаткового аналізу.

Альтернативою повній ручній розмітці є слабке навчання (weak supervision), при якому початкові мітки формуються за допомогою правил

або наявних метаданих – наприклад, якщо оператор вручну призначив тикет до відділу логістики, повідомлення отримує мітку «статус доставки». Такий підхід дозволяє швидко сформувати великий датасет, проте якість міток є нижчою порівняно з ручною розміткою і потребує подальшої верифікації [6].

Після завершення анотування датасет розбивається на три підвибірки: навчальну, валідаційну та тестову у співвідношенні 70/15/15. Стратифікований поділ забезпечує збереження розподілу класів у кожній підвибірці, що є особливо важливим при дисбалансованих даних.

3.2 Система пріоритезації та маршрутизації

Після того як класифікатор визначив тип звернення та його тональність, система має прийняти рішення щодо двох взаємопов'язаних параметрів – пріоритету обробки та напрямку маршрутизації. Ці два параметри визначаються незалежно, але впливають один на одного: пріоритет може коригувати маршрут, а тип звернення – визначати базовий пріоритет.

Пріоритет звернення визначається за комбінацією двох сигналів. Першим є тип звернення – скарги та запити на повернення отримують високий пріоритет за замовчуванням незалежно від тональності, оскільки затримка реакції на них несе найбільші бізнес-ризики. Замовлення та питання про товар отримують середній пріоритет, консультації та запити про співпрацю – низький.

Другим сигналом є результат аналізу тональності – якщо модель виявляє виражену негативну тональність у повідомленні будь-якого типу, пріоритет автоматично підвищується на один рівень. Таким чином, негативно забарвлений запит про статус доставки, який за типом мав би отримати низький пріоритет, підвищується до середнього.

Логіку визначення пріоритету можна формалізувати наступним чином (формула 3.1):

$$P(c, s) = \min (P_{base}(c) + \delta(s), P_{max}), \quad (3.1)$$

де $P(c, s)$ – фінальний пріоритет;

$P_{base}(c)$ – базовий пріоритет для класу c ;

$\delta(s)$ – поправка за тональністю s , що дорівнює 1 при негативній тональності і 0 в інших випадках;

P_{max} – максимальний рівень пріоритету.

Правила маршрутизації визначають відповідність між типом звернення та відділом або виконавцем, до якого воно спрямовується. Ці правила зберігаються у конфігураційному файлі та можуть змінюватись адміністратором системи без втручання у програмний код. Базова таблиця відповідностей наведена у таблиці 3.1.

Таблиця 3.1 – Правила маршрутизації за типом звернення

Тип звернення	Пріоритет	Напрямок маршрутизації
Замовлення	Середній	Відділ продажів
Питання про товар	Низький	Відділ продажів / автовідповідь
Скарга	Високий	Служба підтримки
Повернення / обмін	Високий	Служба підтримки
Статус доставки	Низький	Автовідповідь / логістика
Консультація	Середній	Менеджер
Технічна підтримка	Середній	Техпідтримка
Співпраця	Низький	Керівництво

Окрім статичних правил маршрутизації, система підтримує динамічну маршрутизацію на основі поточного навантаження. Якщо відповідальний відділ або оператор недоступний – наприклад, через позаробочий час або перевищення порогового значення черги – звернення автоматично переводиться у режим очікування з відправкою клієнту підтверджувального повідомлення про отримання запиту та очікуваний час відповіді. Для

високопріоритетних звернень у такому випадку додатково надсилається сповіщення черговому менеджеру.

Важливим елементом системи маршрутизації є механізм ескалації. Якщо звернення не було опрацьовано протягом визначеного часового порогу – наприклад, 30 хвилин для високого пріоритету і 4 години для середнього – система автоматично підвищує його пріоритет і повторно надсилає сповіщення. Після другої ескалації звернення передається безпосередньо керівнику відповідного підрозділу. Такий механізм гарантує, що жодне звернення не залишиться без відповіді через людський фактор або технічний збій.

Усі події маршрутизації – первинне призначення, зміна пріоритету, ескалація – фіксуються у журналі подій із зазначенням часу та ідентифікатора відповідального. Це забезпечує повну простежуваність кожного звернення та дозволяє виявляти системні проблеми у процесі обробки на основі накопиченої статистики.

3.3 Модуль автоматичного опрацювання типових звернень

Автоматичне опрацювання є одним із ключових елементів системи з точки зору бізнес-цінності – саме воно дозволяє знизити навантаження на операторів і забезпечити миттєву відповідь клієнту у позаробочий час. Модуль активується для тих типів звернень, які маршрутизатор визначив як такі, що не потребують участі людини, або для яких передбачена часткова автоматизація з подальшою передачею оператору.

Основним інструментом автоматичного опрацювання є система шаблонних відповідей. Кожному класу звернень відповідає набір шаблонів, з яких система обирає найбільш відповідний на основі додаткових ознак повідомлення – наявності конкретних ключових слів, згаданого товару або попереднього контексту діалогу. Шаблони підтримують динамічні змінні, які заповнюються даними із зовнішніх систем або з контексту

повідомлення. Наприклад, відповідь на запит про статус доставки формується автоматично і містить номер замовлення, поточний статус та очікувану дату доставки, які підставляються у реальному часі через інтеграцію з логістичною системою.

Перевірка статусу замовлення є найбільш поширеним сценарієм автоматичного опрацювання в e-commerce. При надходженні звернення відповідного типу система витягує номер замовлення з тексту повідомлення за допомогою регулярних виразів або NER-моделі, ідентифікує клієнта за ідентифікатором месенджера у базі даних замовлень, робить запит до логістичної системи через API і підставляє отримані дані у відповідний шаблон. Якщо номер замовлення не вдалося витягти автоматично або клієнт не знайдений у базі, система надсилає уточнювальне повідомлення з проханням надати необхідні дані, після чого чекає на відповідь і повторює спробу.

Модуль також реалізує функцію FAQ – автоматичних відповідей на типові питання про товари або умови роботи. База знань формується адміністратором системи і може поповнюватись динамічно. При надходженні питання система обчислює семантичну схожість між вхідним повідомленням та питаннями у базі знань за формулою (3.2):

$$\text{sim}(q, f_i) = \frac{v_q \cdot v_{f_i}}{\|v_q\| \times \|v_{f_i}\|}, \quad (3.2)$$

де v_q – вектор вхідного питання;

v_{f_i} – вектор i -го питання з бази знань.

Якщо максимальне значення схожості перевищує визначений поріг, система повертає відповідну відповідь з бази знань. Якщо жодне питання не досягає порогу схожості, звернення передається оператору.

Формування заявок є третім сценарієм автоматичного опрацювання. При надходженні звернення типу повернення або обміну система може

автоматично ініціювати процедуру – запитати у клієнта необхідні дані, сформувати структуровану заявку та передати її у відповідну систему обліку. Цей сценарій реалізується у вигляді багатокрокового діалогу, де система послідовно запитує номер замовлення, причину повернення та зручний спосіб компенсації. Зібрані дані формуються у структурований об’єкт і передаються у CRM або систему обліку через API [1].

Для усіх сценаріїв автоматичного опрацювання система фіксує факт відправки відповіді, її зміст та час. Якщо клієнт не відреагував на автоматичну відповідь протягом визначеного часу або явно вказав на незадоволеність нею, звернення автоматично передається оператору з повним контекстом діалогу. Це забезпечує плавний перехід між автоматичним і ручним режимами обробки без втрати інформації та без необхідності для клієнта повторювати своє звернення.

3.4 Архітектура системи

3.4.1 Загальна структура та модульна декомпозиція

Система побудована за модульним принципом, де кожен компонент виконує чітко визначену функцію і взаємодіє з іншими через уніфікований інтерфейс. Такий підхід забезпечує незалежність модулів, спрощує тестування та дозволяє замінювати окремі компоненти без впливу на решту системи.

Архітектурно система розподілена на чотири рівні обробки. Перший – прийом повідомлень: вхідні події надходять від підключених месенджерів через вебхуки і потрапляють до шлюзу повідомлень, який нормалізує їх до єдиного внутрішнього формату та поміщає у чергу обробки на основі RabbitMQ.

Черга гарантує, що жодне повідомлення не буде втрачено при тимчасовому перевантаженні або недоступності наступних модулів. Другий

рівень – обробка тексту: нормалізоване повідомлення передається до модуля NLP-обробки, який реалізує повний пайплайн препроцесингу.

Третій – класифікація: модуль визначає тип звернення з-поміж восьми класів, аналізує тональність та формує рівень пріоритету. Четвертий – маршрутизація та опрацювання: на основі результату класифікації система спрямовує звернення або до модуля автовідповіді, або до відповідного відділу компанії. Склад модулів системи та їх призначення наведено у таблиці 3.2.

Таблиця 3.2 – Модульна декомпозиція системи

Модуль	Директорія	Призначення
Шлюз повідомлень	gateway/	Прийом та нормалізація повідомлень із месенджерів
NLP-обробка	nlp/	Препроцесинг та векторизація тексту
Класифікатор	classifier/	Визначення типу звернення та тональності
Маршрутизатор	router/	Пріоритезація та напрямок обробки
Автовідповідач	responder/	Шаблонні відповіді та FAQ
Сховище даних	storage/	Збереження повідомлень та журналу подій
Адмін-бот	admin_bot/	Керування правилами та моніторинг

Точкою входу є `main.py`, який запускає чотири паралельні сервіси – FastAPI-сервер для прийому вебхуків, RabbitMQ-консьюмер для обробки черги, менеджер ескалації та адміністративний бот. Паралельний запуск реалізований через `asyncio.gather`, що дозволяє сервісам працювати у межах одного процесу без блокування один одного (лістинг 3.1).

Лістинг 3.1 – Запуск паралельних сервісів системи

```

async def main():
    await asyncio.gather(
        run_webhook_server(),
        run_queue_consumer(),
        run_escalation_manager(),
        run_admin_bot()
    )

```

Взаємодія між модулями організована таким чином, щоб унеможливити прямі залежності між компонентами. Шлюз повідомлень не

знає нічого про класифікатор – він лише записує нормалізоване повідомлення у чергу RabbitMQ і повертає HTTP 200 вебхуку месенджера.

Консьюмер черги забирає повідомлення і передає його послідовно через NLP-модуль, класифікатор та маршрутизатор. Кожен з цих модулів отримує на вхід об'єкт повідомлення, доповнює його своїми результатами і передає далі. Маршрутизатор приймає фінальне рішення і або викликає модуль автовідповіді, або записує звернення у чергу відповідного відділу.

Усі проміжні результати та фінальний стан кожного звернення зберігаються у PostgreSQL через асинхронний ORM SQLAlchemy. Журнал подій фіксує кожен етап обробки – час надходження, результат класифікації, призначений пріоритет, напрямок маршрутизації та факт відправки відповіді, що забезпечує повну простежуваність кожного звернення і є основою для дообнавлення моделі на реальних даних.

3.4.2 Шлюз повідомлень та адаптери месенджерів

Шлюз повідомлень є першою точкою контакту між зовнішніми месенджерами та внутрішньою інфраструктурою системи. Його основне завдання – прийняти вхідну подію від будь-якого підключеного месенджера, перевірити її автентичність та привести до єдиного внутрішнього формату незалежно від специфіки платформи.

Для кожного месенджера реалізовано окремий адаптер у директорії gateway/. Адаптер відповідає за три речі: верифікацію підпису вхідного запиту, витягування тексту повідомлення та ідентифікатора відправника з платформи-специфічної структури даних, і формування уніфікованого об'єкта повідомлення.

Telegram передає події через вебхук у форматі JSON з вкладеним об'єктом message, WhatsApp Business API – через власну структуру з полем entry, Instagram та Facebook Messenger – через Graph API з подібною, але не

ідентичною схемою. Без адаптерного шару ця різниця просочувалась би у всі наступні модулі системи (лістинг 3.2).

Лістинг 3.2 – Уніфікований формат внутрішнього повідомлення

```
@dataclass
class IncomingMessage:
    message_id: str
    platform: str
    sender_id: str
    text: str
    received_at: datetime
    raw_payload: dict
```

Верифікація підпису є обов’язковим кроком для кожного адаптера. Telegram підписує запити HMAC-SHA256 токеном бота, WhatsApp – власним механізмом підпису через заголовок X-Hub-Signature-256. Запити з невалідним підписом відхиляються на рівні шлюзу до будь-якої подальшої обробки (лістинг 3.3).

Лістинг 3.3 – Верифікація підпису Telegram-вебхуку

```
def verify_telegram(token: str, body: bytes, secret: str)
-> bool:
    expected = hmac.new(
        secret.encode(), body, hashlib.sha256
    ).hexdigest()
    return hmac.compare_digest(expected, token)
```

Після верифікації та нормалізації повідомлення публікується у чергу RabbitMQ. Шлюз не чекає результату подальшої обробки – він одразу повертає HTTP 200 месенджеру, що є критичним для платформ з коротким таймаутом вебхуку, зокрема WhatsApp, який очікує відповідь протягом не більше 5 секунд. Асинхронна обробка через чергу дозволяє витримати цей таймаут навіть при пікових навантаженнях на класифікатор.

Окремою проблемою є повторна доставка вебхуків. WhatsApp та Facebook Messenger повторно надсилають подію якщо не отримали підтвердження вчасно, що може призвести до дублікатів у черзі та повторної

обробки одного і того ж звернення. Для захисту від цього шлюз перевіряє унікальність `message_id` перед публікацією у чергу – ідентифікатори вже оброблених повідомлень зберігаються у Redis з коротким TTL. Якщо повідомлення з таким `message_id` вже було отримано, шлюз повертає HTTP 200 не публікуючи дублікат у чергу (лістинг 3.4).

Лістинг 3.4 – Перевірка дублікатів через Redis

```
async def is_duplicate(message_id: str, redis: Redis) ->
bool:
    key = f"msg:{message_id}"
    if await redis.exists(key):
        return True
    await redis.setex(key, ttl=86400, value=1)
    return False
```

У разі недоступності RabbitMQ при публікації шлюз не відхиляє запит месенджера, а зберігає повідомлення локально у буфері і повторює спробу публікації з експоненційною затримкою. Це гарантує, що короткочасний збій брокера повідомлень не призведе до втрати вхідних звернень.

3.4.3 Модуль NLP-обробки та класифікації

Модуль NLP-обробки та класифікації є центральним компонентом системи з точки зору інтелектуальної обробки – саме тут текст повідомлення перетворюється на структурований результат: тип звернення, тональність та рівень впевненості передбачення. Взаємодію компонентів модуля наведено на рисунку 3.1.

Модуль отримує уніфікований об'єкт повідомлення з черги і запускає його через пайплайн обробки. Перший крок – препроцесинг тексту: очищення від шуму, нормалізація регістру, корекція орфографічних помилок та токенизація. Реалізація розподілена між трьома класами у директорії `nlp/` – `TextCleaner`, `SpellCorrector` та `Tokenizer`, кожен з яких відповідає за окремий етап і може бути замінений незалежно (лістинг 3.5).

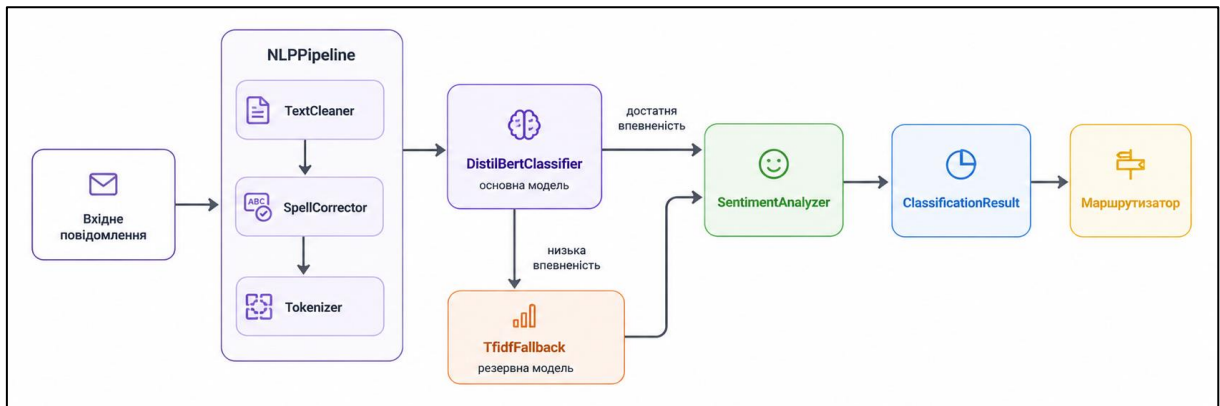


Рисунок 3.1 – Схема взаємодії компонентів модуля NLP-обробки та класифікації

Лістинг 3.5 – Пайплайн NLP-обробки

```

class NLPPipeline:
    def __init__(self):
        self.cleaner = TextCleaner()
        self.corrector = SpellCorrector()
        self.tokenizer = Tokenizer()
    async def process(self, text: str) -> ProcessedText:
        cleaned = self.cleaner.clean(text)
        corrected = self.corrector.correct(cleaned)
        tokens = self.tokenizer.tokenize(corrected)
        return ProcessedText(
            original=text,
            cleaned=cleaned,
            corrected=corrected,
            tokens=tokens)

```

Після препроцесингу оброблений текст передається у модуль класифікації. Основною моделлю є DistilBERT, реалізований у класі DistilBertClassifier. При ініціалізації модуль завантажує донавчену модель та токенизатор з локального сховища. Інференс виконується асинхронно, щоб не блокувати обробку інших повідомлень у черзі (лістинг 3.6).

Лістинг 3.6 – Класифікація повідомлення основною моделлю

```

async def classify(self, text: str) ->
ClassificationResult:
    inputs = self.tokenizer(
        text, return_tensors="pt",

```

Продовження лістингу 3.6

```

        truncation=True, max_length=128)
    with torch.no_grad():
        logits = self.model(**inputs).logits
        probs = torch.softmax(logits, dim=-1)
        confidence, label_id = probs.max(dim=-1)
    return ClassificationResult(
        label=self.labels[label_id],
        confidence=confidence.item())

```

Якщо DistilBERT повертає впевненість нижче визначеного порогу або є недоступним, система автоматично перемикається на резервну модель – TF-IDF у поєднанні з логістичною регресією, реалізовану у класі `TfidfFallback`. Паралельно з класифікацією типу звернення виконується аналіз тональності через `SentimentAnalyzer`. Модуль використовує лексиконний підхід – словник позитивних та негативних слів українською та російською мовами – і повертає одне з трьох значень: позитивна, негативна або нейтральна тональність.

Результати класифікації та аналізу тональності об'єднуються в єдиний об'єкт `ClassificationResult` і передаються маршрутизатору для подальшої обробки звернення.

3.4.4 Модуль маршрутизації та зберігання даних

Модуль маршрутизації отримує об'єкт `ClassificationResult` і на його основі приймає два незалежних рішення – визначає пріоритет звернення та напрямок його подальшої обробки. Логіка пріоритезації і правила маршрутизації реалізовані в окремих класах `PriorityEngine` та `RoutingEngine` у директорії `router/`.

`PriorityEngine` обчислює фінальний пріоритет за формулою: базовий пріоритет визначається типом звернення, після чого коригується залежно від тональності. Реалізація зводиться до двох словників – таблиці базових

пріоритетів за класом та таблиці поправок за тональністю – і однієї операції `min`, що обмежує результат максимальним рівнем (лістинг 3.7).

Лістинг 3.7 – Обчислення пріоритету звернення

```
def compute_priority(self, label: str, sentiment: str) ->
int:
    base = self.base_priorities.get(label,
Priority.MEDIUM)
    delta = self.sentiment_delta.get(sentiment, 0)
    return min(base + delta, Priority.HIGH)
```

`RoutingEngine` зіставляє тип звернення з відділом або виконавцем згідно з правилами маршрутизації, завантаженими з конфігураційного файлу. Правила зберігаються у форматі YAML і можуть змінюватись адміністратором без перезапуску системи – модуль перерахує конфігурацію при кожному старті та підтримує гаряче перезавантаження через адмін-бот. Логіку визначення напрямку маршрутизації наведено у лістингу 3.8.

Лістинг 3.8 – Визначення напрямку маршрутизації

```
async def route(self, result: ClassificationResult) ->
RoutingDecision:
    rule = self.rules.get(result.label)
    if not rule:
        return RoutingDecision(target="default_queue")
    available = await self.check_availability(rule.target)
    if not available:
        return RoutingDecision(
            target="waiting_queue",
            notify_manager=result.priority ==
Priority.HIGH
        )
    return RoutingDecision(target=rule.target)
```

Якщо відповідальний відділ недоступний, звернення потрапляє у чергу очікування. Для високопріоритетних звернень додатково надсилається сповіщення черговому менеджеру через адмін-бот. Механізм ескалації реалізований у окремому фоновому сервісі `EscalationManager`,

який з визначеним інтервалом перевіряє чергу і підвищує пріоритет прострочених звернень.

Усі події обробки зберігаються у PostgreSQL через асинхронний ORM SQLAlchemy. Кожне звернення зберігається двічі – повний об’єкт повідомлення з усіма метаданими у таблиці messages, і окремий запис у журналі подій event_log для кожного етапу обробки. Реалізацію збереження події маршрутизації наведено у лістингу 3.9.

Лістинг 3.9 – Збереження результату маршрутизації

```
async def save_routing_event(
    self, message_id: str,
    decision: RoutingDecision,
    session: AsyncSession
) -> None:
    event = EventLog(
        message_id=message_id,
        event_type="routed",
        target=decision.target,
        priority=decision.priority,
        timestamp=datetime.utcnow()
    )
    session.add(event)
    await session.commit()
```

Окремий запис у event_log для кожного етапу дозволяє відновити повну хронологію обробки будь-якого звернення – від надходження до відправки відповіді клієнту.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Метрики оцінювання якості класифікації

Для оцінювання якості класифікаційної моделі використовується набір метрик, що дозволяють отримати повну картину її поведінки на тестовій вибірці. Базовою метрикою є ассигасу – частка правильно класифікованих прикладів від загальної кількості. Попри простоту інтерпретації, ассигасу є недостатньою метрикою при дисбалансі класів: модель, що завжди передбачає мажоритарний клас, може демонструвати високу ассигасу при низькій практичній цінності.

Для детальнішої оцінки використовуються *precision*, *recall* та *F1-score* у розрізі кожного класу. *Precision* показує, яка частка передбачень певного класу є коректною, і обчислюється за формулою (4.1):

$$Precision(k) = \frac{TP_k}{TP_k + FP_k}, \quad (4.1)$$

де TP_k – кількість правильно передбачених прикладів класу k ;

FP_k – кількість прикладів інших класів, помилково віднесених до класу k .

Recall показує, яка частка реальних прикладів класу була коректно розпізнана моделлю у формулі 4.2:

$$Recall(k) = \frac{TP_k}{TP_k + FN_k}, \quad (4.2)$$

де FN_k – кількість прикладів класу k , помилково віднесених до інших класів.

F1-score є гармонічним середнім між *precision* та *recall* і обчислюється за формулою 4.3:

$$F1(k) = 2 \times \frac{Precision(k) \times Recall(k)}{Precision(k) + Recall(k)}. \quad (4.3)$$

Для агрегування метрик по всіх класах використовується *macro-averaging* – обчислення середнього арифметичного метрики по кожному класу без урахування його частоти у вибірці. Такий підхід є більш інформативним при дисбалансі класів, оскільки однаково враховує як мажоритарні, так і міноритарні класи.

Окремо оцінюється матриця помилок (*confusion matrix*), яка показує розподіл передбачень моделі по всіх парах класів. Аналіз матриці дозволяє виявити систематичні помилки – наприклад, часте змішування класів «скарга» та «повернення» – і прийняти обґрунтовані рішення щодо донавчання моделі або корекції правил маршрутизації для граничних випадків.

4.2 Результати тестування класифікатора

Тестування проводилось на валідаційній вибірці з 509 прикладів, сформованій шляхом стратифікованого розбиття датасету у співвідношенні 80/20. Окремо оцінювались обидві моделі системи – TF-IDF з логістичною регресією та донавчений DistilBERT.

TF-IDF fallback показав загальну *accuracy* 0.94 при *macro F1-score* 0.84. Більшість класів демонструють стабільні показники в діапазоні 0.93–0.97, однак клас *product_question* отримав *F1-score* лише 0.22 через критично малий обсяг навчальних прикладів – 20 зразків проти 420 у решти класів. Детальні метрики TF-IDF fallback наведено у таблиці 4.1. DistilBERT показав вищі результати по більшості класів – загальна *accuracy* 0.95 при *macro F1-score* 0.89. Клас *order_status* досяг *precision* 1.00, що означає відсутність хибнопозитивних передбачень.

Таблиця 4.1 – Метрики TF-IDF fallback по класах

Клас	Precision	Recall	F1-score	Support
complaint	0.94	0.92	0.93	85
delivery_problem	0.95	0.94	0.95	84
general	0.95	0.92	0.93	84
order_status	0.99	0.95	0.97	84
payment_issue	0.95	0.95	0.95	84
product_question	0.14	0.50	0.22	4
return_exchange	0.96	0.95	0.96	84
macro avg	0.84	0.88	0.84	509
weighted avg	0.95	0.94	0.94	509

Клас `payment_issue` при precision 1.00 показав recall 0.89 – частина платіжних звернень помилково класифікується як суміжні класи. Клас `product_question` залишається проблемним з F1-score 0.50, що є прямим наслідком дисбалансу датасету. Детальні метрики DistilBERT наведено у таблиці 4.2.

Таблиця 4.2 – Метрики DistilBERT по класах

Клас	Precision	Recall	F1-score	Support
complaint	0.95	0.94	0.95	85
delivery_problem	0.94	0.98	0.96	84
general	0.94	0.98	0.96	84
order_status	1.00	0.96	0.98	84
payment_issue	1.00	0.89	0.94	84
product_question	0.38	0.75	0.50	4
return_exchange	0.92	0.95	0.94	84
macro avg	0.88	0.92	0.89	509
weighted avg	0.95	0.95	0.95	509

Порівняння двох моделей за ключовими агрегованими метриками наведено у таблиці 4.3.

Таблиця 4.3 – Порівняння моделей за агрегованими метриками

Метрика	TF-IDF + LR	DistilBERT
Accuracy	0.94	0.95
Macro F1	0.84	0.89
Weighted F1	0.94	0.95

DistilBERT перевершує TF-IDF за macro F1 на 0.05, що є значущим покращенням з урахуванням дисбалансу класів. Низькі показники класу `product_question` в обох моделях вказують на необхідність розширення датасету для цього класу – наявних 20 прикладів недостатньо для надійного навчання. Для решти шести класів обидві моделі відповідають нефункціональним вимогам системи щодо мінімального F1-score 0.80.

4.3 Демонстрація роботи системи

Для перевірки роботи системи в реальних умовах було проведено серію тестових звернень через Telegram-канал. Система запускала локально з підключенням через ngrok-тунель, що забезпечило доступність вебхуку для Telegram API. Усі компоненти – FastAPI-сервер, RabbitMQ-черга, класифікатор та адмін-бот – працювали паралельно в рамках одного процесу.

Перше звернення містило скаргу на якість товару: «Добрий день, я вже третій раз пишу і ніхто не відповідає, товар прийшов зіпсований, дуже незадоволена». Класифікатор визначив клас `complaint` з негативною тональністю, що автоматично підвищило пріоритет до рівня 1. Для цього класу автовідповідь вимкнена – система передала звернення до відділу якості та надіслала клієнту підтвердження, що наведено на рисунку 4.1.

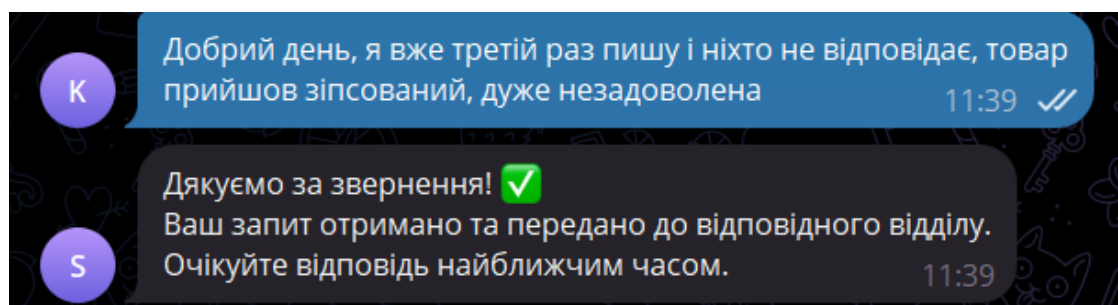


Рисунок 4.1 – Обробка скарги: підтвердження прийому та передача до відділу якості

Наступним надійшов запит про статус доставки: «Привіт, підкажіть де моя посилка? Замовлення #4521». Тут спрацював сценарій повної автоматизації – система визначила клас `order_status`, витягла номер замовлення, отримала статус з логістичної системи і відповіла клієнту без жодного залучення оператора, що наведено на рисунку 4.2.

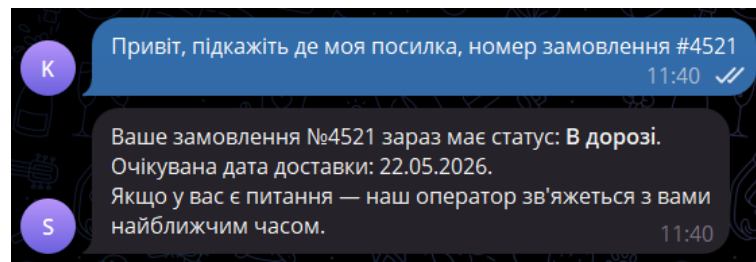


Рисунок 4.2 – Автоматична відповідь на запит статусу замовлення #4521

Питання про товар «Добрий день, підкажіть ця куртка є в чорному кольорі?» було класифіковано як `product_question`. База знань не містила відповіді на це конкретне запитання, тому система повідомила клієнта про передачу до оператора відділу продажів, що наведено на рисунку 4.3.

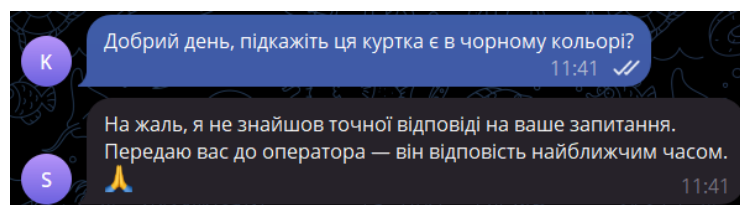


Рисунок 4.3 – Передача питання про товар до оператора відділу продажів

Звернення «Хочу повернути кросівки, вони мені малі виявились» отримало клас `return_exchange` з пріоритетом 2. Система ініціювала процедуру повернення та підтвердила прийом клієнту, що наведено на рисунку 4.4.

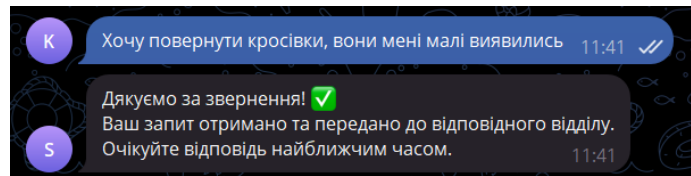


Рисунок 4.4 – Прийом запиту на повернення та передача до служби підтримки

«Моя посилка застрягла на сортувальному центрі вже 5 днів, що відбувається?» – класифіковано як `delivery_problem`. Негативна тональність підвищила пріоритет до 1, звернення маршрутизовано до відділу логістики, що наведено на рисунку 4.5.

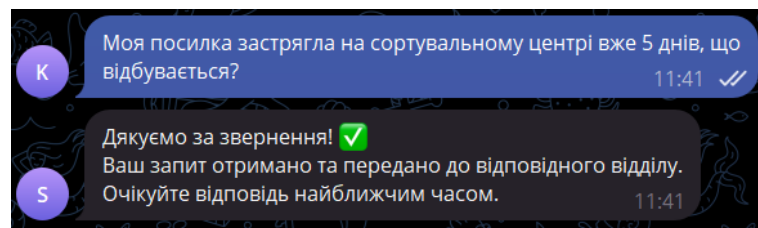


Рисунок 4.5 – Маршрутизація проблеми з доставкою до відділу логістики з пріоритетом 1

Результати обробки звернень відображались в адмін-боті в реальному часі. Команда `/stats` показала підсумкову статистику: всього 10 звернень, 7 з яких – 70% – оброблено автоматично, 3 передано операторам, що наведено на рисунку 4.6.

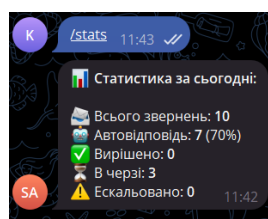


Рисунок 4.6 – Статистика обробки звернень в адмін-боті

Команда `/queue` відобразила поточний стан черги: `logistics` – 2 звернення, `support` – 1, найстаріше звернення класу `delivery_problem` з пріоритетом 1 очікує вже 3 хвилини, що наведено на рисунку 4.7.

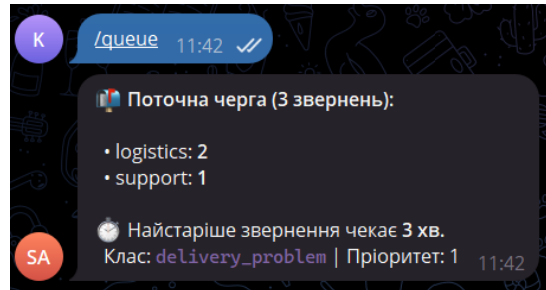


Рисунок 4.7 – Поточна черга звернень по відділах

Команда `/escalations` показала одне активне ескалаційне звернення класу `return_exchange` з пріоритетом 1 – з кнопками для позначення як вирішеного або передачі вище, що наведено на рисунку 4.8.

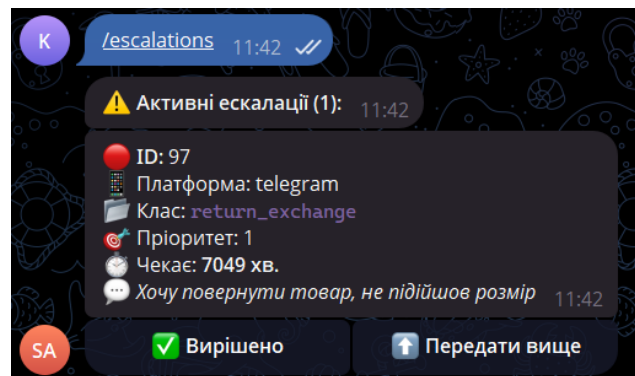


Рисунок 4.8 – Інтерфейс управління ескалаціями в адмін-боті

Окрім моніторингу, адмін-бот надає доступ до повного переліку шаблонів автовідповідей через вбудоване меню з можливістю їх перегляду та редагування в реальному часі без перезавантаження системи, що наведено на рисунку 4.9.

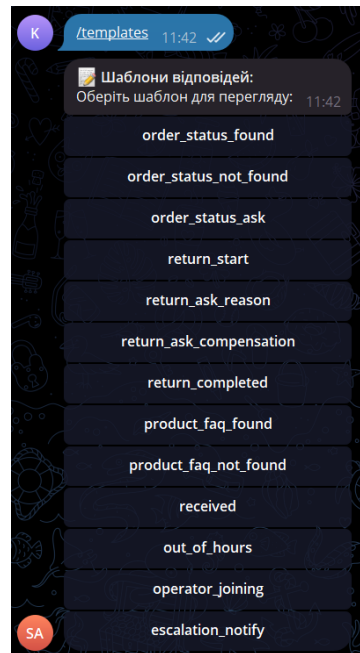


Рисунок 4.9 – Перелік шаблонів автовідповідей в адмін-боті

Управління шаблонами реалізоване через інлайн-кнопки безпосередньо в інтерфейсі Telegram, що не вимагає від адміністратора жодних технічних знань для внесення змін у логіку автовідповідей.

4.4 Аналіз результатів та рекомендації

За результатами тестування на валідаційній вибірці з 509 прикладів система показала accuracy 0.95 при використанні DistilBERT. Для шести з семи класів F1-score перевищує 0.80 – встановлену нефункціональну вимогу.

Єдиний проблемний клас – `product_question` з F1-score 0.50. Причина проста: 20 прикладів у датасеті проти 420 у решти класів. Жодна модель не навчиться надійно розпізнавати клас при такому дисбалансі. Розширення до 400-500 прикладів є першочерговим кроком.

DistilBERT перевершує TF-IDF за macro F1 на 0.05 – з 0.84 до 0.89. На шести збалансованих класах різниця помірна, але на `product_question` вона

суттєва: 0.50 проти 0.22. Контекстні ембедінги краще справляються з малими вибірками завдяки перенесенню знань з попереднього навчання.

Гібридний підхід з TF-IDF як резервною моделлю виправдав себе на практиці. При недоступності DistilBERT або низькій впевненості передбачення система перемикається автоматично без втрати функціональності. Час інференсу на CPU склав 2-3 секунди на повідомлення, що вкладається у встановлений ліміт.

Демонстрація підтвердила коректну маршрутизацію по всіх п'яти тестових сценаріях – 70% звернень оброблено автоматично. Механізм корекції пріоритету за тональністю спрацював коректно: `delivery_problem` з негативним забарвленням отримав пріоритет 1 і був направлений до відділу логістики.

З напрямків подальшого розвитку найбільш актуальними є три. Розширення датасету для класу `product_question` – без цього якість по цьому класу залишатиметься незадовільною незалежно від архітектури моделі. Впровадження мультилейбл-класифікації для обробки змішаних звернень, що одночасно несуть ознаки кількох класів. І додавання підтримки контексту діалогу – зараз система класифікує кожне повідомлення ізольовано, що знижує точність на коротких неформальних фразах без достатнього контексту.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було вирішено задачу автоматизації класифікації, пріоритезації та маршрутизації клієнтських звернень у месенджерах на основі методів обробки природної мови та машинного навчання.

В ході виконання роботи проведено аналіз предметної галузі, який показав, що месенджери сформували специфічне середовище клієнтської комунікації з характерними особливостями – короткістю повідомлень, неформальністю, орфографічними помилками та змішуванням мов. Існуючі комерційні платформи – Zendesk, Intercom, Freshdesk – не дозволяють навчати класифікаційні моделі на власних даних компанії та мають фіксовану таксономію класів, що обумовило доцільність розробки власного рішення.

Розглянуто методи попередньої обробки тексту, векторизації та класифікації. Порівняльний аналіз підходів показав, що для задачі класифікації коротких неформальних текстів з месенджерів трансформерні моделі мають суттєву перевагу над класичними методами завдяки контекстним ембедінгам та стійкості до орфографічних помилок на рівні підслів. Обґрунтовано вибір DistilBERT як основної моделі та TF-IDF з логістичною регресією як резервної.

Розроблено програмну систему з модульною архітектурою, яка включає шлюз повідомлень з адаптерами для Telegram, WhatsApp, Instagram та Facebook, NLP-пайплайн, двоетапний класифікатор, модуль пріоритезації та маршрутизації, систему автоматичних відповідей та адміністративний бот для моніторингу.

За результатами тестування на валідаційній вибірці з 509 прикладів система досягла ассураcy 0.95 та macro F1-score 0.89 для моделі DistilBERT. Для шести з семи класів F1-score перевищує встановлену вимогу 0.80.

Єдиним проблемним класом є `product_question`, що пояснюється критичним дисбалансом датасету – 20 прикладів проти 420 у решті класів.

Демонстрація роботи системи в реальних умовах підтвердила коректну класифікацію та маршрутизацію по всіх тестових сценаріях. Автоматично оброблено 70% звернень тестової сесії, механізм корекції пріоритету за тональністю спрацював коректно для всіх класів.

Практична цінність роботи полягає у можливості розгортання системи для будь-якого бізнесу, що використовує месенджери як канал клієнтської підтримки, з адаптацією таксономії класів та правил маршрутизації під конкретні бізнес-процеси без зміни програмного коду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jurafsky D., Martin J. H. Speech and language processing. 3rd ed. Stanford, 2023. URL: <https://web.stanford.edu/~jurafsky/slp3/> (date of access: 13.05.2026).
2. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT 2019*. Minneapolis, 2019. P. 4171–4186. URL: <https://doi.org/10.18653/v1/N19-1423> (date of access: 13.05.2026).
3. Distributed representations of words and phrases and their compositionality / T. Mikolov et al. *Advances in Neural Information Processing Systems 26 (NeurIPS 2013)*. 2013. P. 3111–3119. URL: <https://doi.org/10.48550/arXiv.1310.4546> (date of access: 13.05.2026).
4. Hochreiter S., Schmidhuber J. Long short-term memory. *Neural Computation*. 1997. Vol. 9, № 8. P. 1735–1780. URL: <https://doi.org/10.1162/neco.1997.9.8.1735> (date of access: 13.05.2026).
5. Sanh V., Debut L., Chaumond J., Wolf T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint*. 2019. URL: <https://doi.org/10.48550/arXiv.1910.01108> (date of access: 13.05.2026).
6. Manning C. D., Schütze H. Foundations of statistical natural language processing. Cambridge : MIT Press, 1999. 680 p.
7. Scikit-learn: Machine Learning in Python / F. Pedregosa et al. *Journal of Machine Learning Research*. 2011. Vol. 12. P. 2825–2830. URL: <https://doi.org/10.48550/arXiv.1201.0490> (date of access: 13.05.2026).
8. Joulin A., Grave E., Bojanowski P., Mikolov T. Bag of tricks for efficient text classification. *Proceedings of EACL 2017*. Valencia, 2017. P. 427–431. URL: <https://doi.org/10.18653/v1/E17-2068> (date of access: 13.05.2026).
9. Kim Y. Convolutional neural networks for sentence classification. *Proceedings of EMNLP 2014*. Doha, 2014. P. 1746–1751. URL: <https://doi.org/10.3115/v1/D14-1181> (date of access: 13.05.2026).

10. Zhang Y., Wallace B. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *Proceedings of IJCNLP 2017*. Taipei, 2017. P. 253–263. URL: <https://doi.org/10.18653/v1/I17-1026> (date of access: 13.05.2026).
11. Attention Is All You Need / A. Vaswani et al. *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. 2017. URL: <https://doi.org/10.48550/arXiv.1706.03762> (date of access: 13.05.2026).
12. Bird S., Klein E., Loper E. Natural language processing with Python. Sebastopol : O'Reilly Med.
13. Python 3.14 documentation. *Python documentation*. URL: <https://docs.python.org/3/> (date of access: 25.05.2026).
14. FastAPI - FastAPI. *FastAPI - FastAPI*. URL: <https://fastapi.tiangolo.com/> (date of access: 25.05.2026).
15. SQLAlchemy Documentation – SQLAlchemy 2.0 Documentation. *SQLAlchemy Documentation – SQLAlchemy 2.0 Documentation*. URL: <https://docs.sqlalchemy.org/en/20/> (date of access: 25.05.2026).
16. RabbitMQ Documentation | RabbitMQ. *RabbitMQ: One broker to queue them all | RabbitMQ*. URL: <https://www.rabbitmq.com/docs> (date of access: 25.05.2026).
17. Telegram Bot API. *Telegram APIs*. URL: <https://core.telegram.org/bots/api> (date of access: 25.05.2026).
18. Transformers: State-of-the-Art Natural Language Processing / T. Wolf et al. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online. Stroudsburg, PA, USA, 2020. URL: <https://doi.org/10.18653/v1/2020.emnlp-demos.6> (date of access: 25.05.2026).
19. PyTorch Contributors. *PyTorch documentation*. *PyTorch*. URL: <https://docs.pytorch.org/docs/2.12/index.html> (date of access: 25.05.2026).
20. PostgreSQL 15.18 Documentation. *PostgreSQL Documentation*. URL: <https://www.postgresql.org/docs/15/> (date of access: 25.05.2026).