**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**БУДІВНИЦТВА І АРХІТЕКТУРИ**

Автоматизації і інформаційних технологій

(факультет)

Інформаційних технологій

(назва кафедри)

# ПОЯСНЮВАЛЬНА ЗАПИСКА

ДО КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТР

на тему:
«Автоматизована система класифікації та маршрутизації запитів клієнтів на основі обробки природної мови»

Колосовський Владислав Володимирович
(прізвище, ім’я та по батькові студента повністю)

Київ 2024 р.

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**БУДІВНИЦТВА І АРХІТЕКТУРИ**

Автоматизації і інформаційних технологій
(факультет)
Інформаційних технологій
(назва кафедра)

ЗАТВЕРДЖУЮ
Завідувач кафедри
д. т. н., доцент Гончаренко Т. А.
„___” _____2024 року

# ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТР

Автоматизована система класифікації та маршрутизації запитів клієнтів на основі обробки природної мови
(назва)

Виконав:
Колосовський
Владислав Володимирович
(прізвище, ім’я та по батькові повністю)
122. Комп’ютерні науки
(спеціальність)
Комп’ютерні науки
(освітня програма)
Групи КН-20-1
Керівники Долгополов С. Ю. (1), Гончаренко Т.А. (2)
(прізвище та ініціали)
ас. (1), д.т.н., доцент (2)
(вчене звання, науковий ступінь)
*Ідентичність підтверджую*

Київ 2024 р.

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**БУДІВНИЦТВА І АРХІТЕКТУРИ**

Факультет: Автоматизації і інформаційних технологій

Випускова кафедра: Інформаційних технологій

Освітній рівень: «магістр»

Спеціальність: 122 «Комп'ютерні науки»

Освітня програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри

д. т. н., доцент Гончаренко Т. А.

„___" _____2024 року

**З А В Д А Н Н Я**

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТР

Колосовський Владислав Володимирович

(прізвище, ім'я та по батькові студента)

1. Тема роботи
Автоматизована система класифікації та маршрутизації запитів клієнтів на основі обробки природної мови
затверджена наказом ректора КНУБА № 2613/2 від 08.10.2024
2. Керівник роботи Долгополов С. Ю, ас.; Гончаренко Т. А., д. т. н., доцент
( прізвище, ім'я та по батькові, науковий ступінь, вчене звання)
3. Строк подання студентом роботи до захисту _____
4. Зміст пояснювальної записки за розділами:
Р. 1. Теоретичні основи обробки природної мови та класифікації клієнтських запитів.
Р. 2. Методологічні засади розробки автоматизованої системи класифікації запитів на основі обробки природної мови.
Р. 3. Програмно-технічне рішення автоматизованої системи класифікації та маршрутизації запитів на основі обробки природної мови.
Р. 4. Впровадження та оцінка економічної ефективності.
5. Графічний матеріал за розділами:
Р. 1. Архітектури моделей та блок-схеми до теоретичної складової дослідження.
Р. 2. Загальна архітектура системи, деталізовані модулі та методи системи.
Р. 3. Стек технологій, ілюстрації відповідно до розробленого програмного продукту, метрики точності/якості моделі.

Р. 4. Плани розвитку відповідно до плану розвитку.

6. Календарний план виконання роботи:

| Види робіт та їх зміст | Дата виконання |
|---|---|
| Розділ 1 | |
| Розділ 2 | |
| Розділ 3 | |
| Розділ 4 | |
| Остаточне оформлення роботи | |
| Направлення роботи для перевірки на плагіат | |
| Попередній захист роботи на випусковій кафедрі | |
| Направлення роботи на рецензування | |

7. Консультанти розділів кваліфікаційної випускної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Перевірив | |
|---|---|---|---|
| | | дата | підпис |
| Розділ 1 | Гончаренко Т. А. | 15.10.2024 | |
| Розділ 2 | Гончаренко Т. А. | 25.10.2024 | |
| Розділ 3 | Долгополов С.Ю. | 10.11.2024 | |
| Розділ 4 | Долгополов С.Ю. | 23.11.2024 | |

8. Дата видачі завдання 08.10.2024

Зав. кафедри _____ _____
(підпис) (прізвище та ініціали)
Керівник (1) _____ _____
(підпис) (прізвище та ініціали)
Керівник (2) _____ _____
(підпис) (прізвище та ініціали)
Здобувач _____ _____
(підпис) (прізвище та ініціали)

# РЕЗЮМЕ (SUMMARY )

| | | | |
|---|---|---|---|
| РЕЗЮМЕ (SUMMARY) до кваліфікаційної випускної роботи Здобувача: | Колосовський Владислав Володимирович<br><br>Kolosovskyi Vladyslav Volodymyrovych | | |
| ЗВО | Київський національний університет будівництва і архітектури | | |
| Тема (українською та англійською) | Автоматизована система класифікації та маршрутизації запитів клієнтів на основі обробки природної мови<br>Automated System for Client Query Classification and Routing Based on Natural Language Processing | | |
| Освітній ступінь | Магістр | | |
| Факультет | Автоматизації і інформаційних технологій | | |
| Випускаюча кафедра | Інформаційних технологій | | |
| Спеціальність | 122 «Комп'ютерні науки» | | |
| Освітня програма | Комп'ютерні науки | | |
| Керівники | Долгополов С. Ю. та Гончаренко Т. А. | | |
| Обсяг роботи: | пояснювальна записка, стор. | розділів | слайди презентації |
| | 114 | 4 | 41 |
| Ключові слова:<br><br>Keywords: | Обробка природної мови, класифікація текстів, машинне навчання, нейронні мережі, маршрутизація<br><br>Natural language processing, text classification, machine learning, neural networks, request routing | | |

Робота присвячена розробці автоматизованої системи класифікації та маршрутизації запитів клієнтів на основі обробки природної мови. Розроблена система використовує методи машинного навчання та нейронні мережі для автоматичної категоризації запитів та їх оптимального розподілу між операторами, що значно підвищує ефективність обслуговування клієнтів.

The work is dedicated to the development of an automated system for customer request classification and routing based on natural language processing. The developed system uses machine learning methods and neural networks for automatic request categorization and their optimal distribution among operators, which significantly improves customer service efficiency.

Здобувач: _____ / Владислав КОЛОСОВСЬКИЙ /
Керівник (1): _____/ Сергій ДОЛГОПОЛОВ /
Керівник (2): _____/ Тетяна ГОНЧАРЕНКО /
"___" _____ 2024 р.

**ЗМІСТ**

# ВСТУП

У сучасному світі, де цифрова трансформація бізнесу стає необхідністю, а не перевагою, ефективна обробка та управління клієнтськими запитами є критично важливим фактором успіху організації. Зростаючий обсяг комунікацій через різні канали зв'язку створює значне навантаження на служби підтримки клієнтів, що призводить до збільшення часу очікування відповіді та потенційного зниження якості обслуговування.

Традиційні методи ручної обробки запитів стають все менш ефективними, особливо в умовах постійного зростання кількості звернень та вимог до швидкості реакції, що створює потребу в розробці інтелектуальних систем, здатних автоматично аналізувати, класифікувати та спрямовувати запити до відповідних спеціалістів, забезпечуючи оптимальний розподіл ресурсів та підвищення якості обслуговування клієнтів.

**Актуальність даного дослідження** зумовлена необхідністю підвищення ефективності обробки зростаючого потоку клієнтських звернень, оптимізації використання людських ресурсів та забезпечення високої якості обслуговування. Використання методів обробки природної мови та машинного навчання дозволяє створити інтелектуальну систему, здатну автоматично визначати категорію запиту та обирати оптимального обробника, що значно скорочує час реакції та підвищує задоволеність клієнтів.

**Аналіз останніх досліджень та наукових праць.** У галузі обробки природної мови та класифікації текстів значний внесок зробили як вітчизняні, так і зарубіжні науковці. Дослідження T. Brown та його колег з OpenAI демонструють ефективність використання великих мовних моделей для розуміння та класифікації тексту. Значний прогрес у розвитку трансформерних архітектур досягнуто завдяки роботам A. Vaswani та співавторів. Вагомий внесок у розвиток методів векторизації тексту зробили Y. Liu та його колеги з розробкою RoBERTa. Дослідження M. Lewis та співавторів у сфері послідовностей до послідовностей (sequence-to-sequence) з використанням

BART архітектури відкрили нові можливості для обробки текстових даних. Роботи Z. Yang та його колег з XLNet значно покращили розуміння контексту в задачах класифікації тексту.

**Метою кваліфікаційної роботи** є розробка та впровадження автоматизованої системи класифікації та маршрутизації запитів клієнтів на основі методів обробки природної мови, що забезпечує ефективну обробку клієнтських звернень та оптимальний розподіл ресурсів.

Виходячи з мети визначимо наступні завдання:

1) Дослідити сучасні підходи до обробки природної мови та проаналізувати їх ефективність;
2) Охарактеризувати методи та алгоритми класифікації текстових даних;
3) Проаналізувати існуючі системи маршрутизації клієнтських запитів;
4) Визначити архітектуру системи класифікації та маршрутизації запитів;
5) Розробити методи попередньої обробки та векторизації текстових даних;
6) Реалізувати алгоритм класифікації та маршрутизації запитів;
7) Обґрунтувати вибір технологій та інструментів розробки;
8) Розробити компоненти системи та здійснити їх інтеграцію;
9) Провести тестування та оцінку ефективності системи;
10) Спланувати процес впровадження системи в експлуатацію;
11) Оцінити економічну ефективність впровадження системи;
12) Визначити перспективи розвитку та масштабування системи.

**Об’єктом дослідження** є автоматизована система обробки, класифікації та маршрутизації текстових запитів на основі обробки природної мови.

**Предметом дослідження** є методи та алгоритми класифікації текстових даних на основі обробки природної мови та механізми маршрутизації запитів клієнтів.

**Методи дослідження.** Для досягнення мети роботи використовуються методи машинного навчання для класифікації текстів, методи векторизації та попередньої обробки текстових даних, методи глибокого навчання, зокрема

нейронні мережі трансформерної архітектури, методи статистичного аналізу для оцінки ефективності системи, методи системного аналізу для проектування архітектури системи, методи програмної інженерії для реалізації компонентів системи.

**Наукова новизна роботи** полягає в розробці гібридного підходу до класифікації текстів, що поєднує методи TF-IDF та BERT-ембедінги для забезпечення підвищеної точності класифікації, створенні адаптивного алгоритму маршрутизації з урахуванням множини факторів при розподілі запитів, а також розробці методики оцінки ефективності класифікації з урахуванням специфіки клієнтських запитів. Крім того, було вдосконалено методи попередньої обробки текстових даних, що дозволило значно підвищити якість класифікації та точність розподілу запитів між операторами.

**Практичне значення роботи** проявляється у створенні повноцінної автоматизованої системи обробки клієнтських запитів, що значно скорочує час їх обробки та оптимізує використання людських ресурсів служби підтримки. Реалізовані алгоритми характеризуються високим рівнем адаптивності та можуть бути використані в різних предметних областях та організаціях різного масштабу. Система також забезпечує комплексний збір та аналіз статистичних даних для подальшої оптимізації процесів обслуговування клієнтів, а розроблені компоненти легко інтегруються з існуючими системами управління взаємовідносинами з клієнтами, що робить її практично цінною для широкого спектру бізнес-застосувань.

**Структура та обсяг кваліфікаційної роботи.** Робота складається з 114 сторінки, які включають: вступ, чотири розділи з висновками до кожного з них, загальні висновки, список використаних джерел, що налічує 36 найменувань; містить 12 таблиць та 62 рисунки.

# РОЗДІЛ 1.
# ТЕОРЕТИЧНІ ОСНОВИ ОБРОБКИ ПРИРОДНОЇ МОВИ ТА КЛАСИФІКАЦІЇ КЛІЄНТСЬКИХ ЗАПИТІВ

## 1.1. Сучасні підходи до обробки природної мови

Обробка природної мови (Natural Language Processing, NLP) являє собою міждисциплінарну галузь, що поєднує в собі методи комп'ютерних наук, лінгвістики та штучного інтелекту для забезпечення взаємодії між комп'ютерними системами та природною мовою людини. Дослідницька група з OpenAI на чолі з T. Brown у співпраці з науковцями з різних установ, включаючи Stanford University та University of California, здійснили революційний прорив у розумінні можливостей великомасштабних мовних моделей (Large Language Models, LLM) [1], продемонструвавши, що такі моделі здатні до few-shot learning, тобто навчання на основі кількох прикладів, що відкриває нові перспективи у розвитку систем обробки природної мови.

Візуальне представлення архітектури сучасних мовних моделей можна побачити на рис. 1.1, де зображено базову структуру трансформера з механізмом encoder-decoder, що став основою для багатьох сучасних рішень у сфері NLP. Енкодер використовується для обробки вхідної послідовності, а декодер для генерації вихідної послідовності.

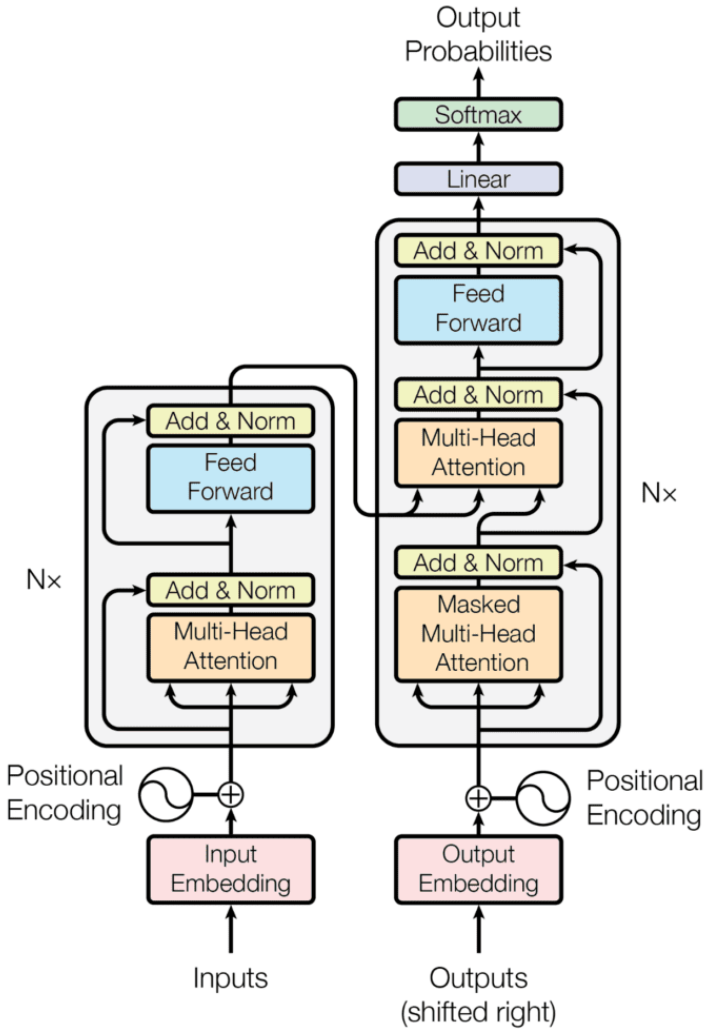


Рисунок 1.1. Базова архітектура трансформера

Колектив дослідників з Google Research – C. Raffel, N. Shazeer, разом зі своїми колегами з Google Brain, запропонували уніфіковану структуру T5 (Text-to-Text Transfer Transformer), яка радикально змінила підхід до вирішення різноманітних завдань NLP [2]. Основна ідея їх підходу полягає у представленні будь-якого NLP-завдання як перетворення тексту в текст, що математично можна описати наступною формулою:

$$P(y|x) = \prod_i P(y_i|y_{<i}, x), \quad (1.1)$$

Ця формула відображає ймовірність генерації вихідної послідовності $y$ на основі вхідної послідовності $x$, де:

де $P(y|x)$ – ймовірність генерації послідовності $y$ на основі вхідної послідовності $x$;

$y_i$ – поточний токен, що генерується;

$y_{<i}$ – попередні згенеровані токени;

$\prod_i$ – добуток ймовірностей для кожного токена.

Подібний підхід дозволяє уніфікувати різноманітні завдання NLP, від машинного перекладу до відповідей на запитання.

Колектив дослідників з Facebook AI Research під керівництвом Y. Liu представив світу RoBERTa – оптимізовану версію BERT, яка демонструє вражаючі результати у широкому спектрі NLP-завдань [3]. На рис. 1.2 можна побачити детальну схему механізму багатоголової уваги, який є ключовим елементом цієї архітектури. Механізм Multi-head attention передбачає паралельну обробку різних аспектів вхідної інформації через окремі голови уваги, їх подальшу конкатенацію та лінійне перетворення. Кожна голова уваги працює з різними проекціями вхідних даних, що дозволяє моделі фокусуватися на різних аспектах вхідної послідовності.

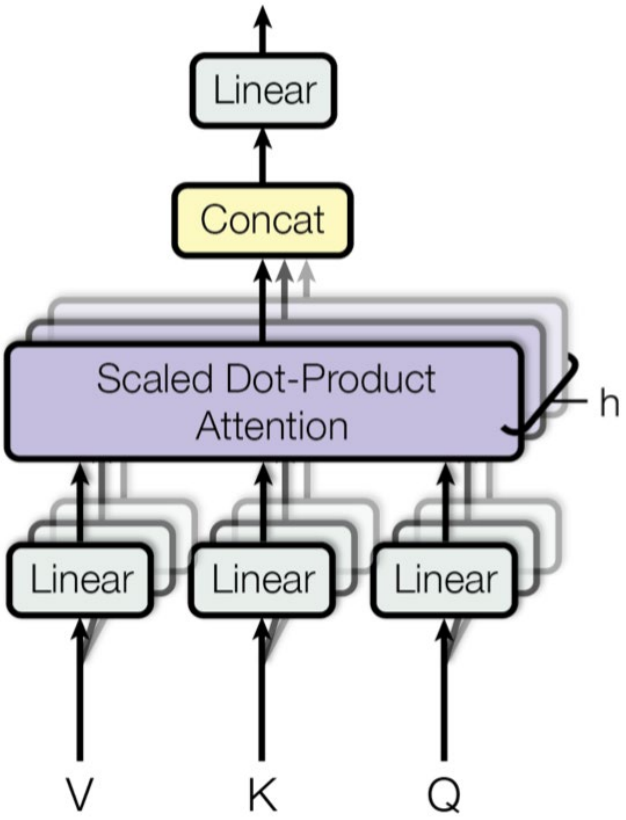


Рисунок 1.2. Механізм multi-head attention

Математично механізм багатоголової уваги можна описати наступною системою рівнянь:

$$\text{MultiHead}(\text{Q}, \text{K}, \text{V}) = Concat(head_1, \ldots, head_h) \cdot W^O, \quad (1.2)$$

$$head_i = Attention(QW_i^g, KW_i^k, VW_i^v),$$

де, $Q$, $K$, $V$ представляють матриці запитів, ключів та значень відповідно;

$W_i^g$, $W_i^k$, $W_i^v$ є матрицями проекцій для кожної голови;

$W^O$ – матриця вихідного лінійного перетворення;

*Concat* означає операцію конкатенації результатів різних голів уваги.

Значний внесок у розвиток сучасних архітектур NLP зробила група науковців з Carnegie Mellon University та Google AI Brain Team на чолі з Z. Yang, які представили XLNet – узагальнений авторегресивний метод попереднього навчання [4], що вдосконалив існуючі підходи до моделювання послідовностей, представивши нову метрику оцінки надійності:

$$\text{R} = \frac{\sum(P_i \times C_i)}{N}, \quad (1.3)$$

де, $P_i$ відображає ймовірність правильної відповіді для $i$-го прикладу;

$C_i$ представляє рівень впевненості моделі в своєму передбаченні;

$N$ є загальною кількістю тестових прикладів;

$R$ представляє загальний показник надійності моделі.

Варто відзначити роботу колаборації дослідників з Facebook AI Research під керівництвом M. Lewis, які представили архітектуру BART [5]. На рис. 1.3 можна побачити детальну схему цієї архітектури, що включає механізм деноізингу автоенкодера та авторегресивного декодера. Схема демонструє процес обробки пошкодженого тексту через енкодер, та відновлення оригінального тексту через декодер, що робить модель особливо ефективною для завдань генерації та розуміння тексту.

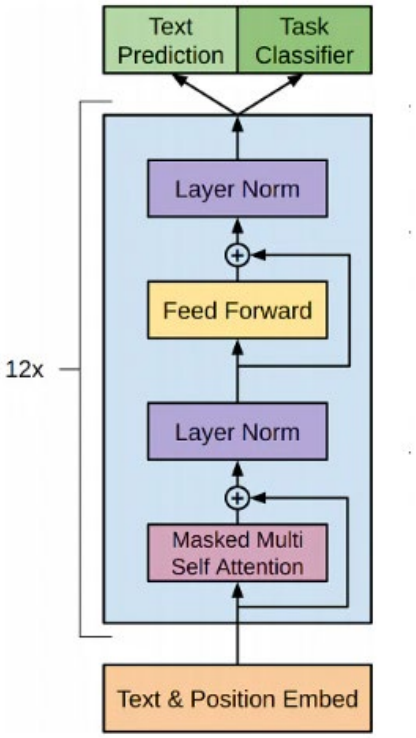


Рисунок 1.3. Архітектура моделі BART

Токенізація та лематизація є фундаментальними процесами попередньої обробки тексту в системах NLP, що забезпечують перетворення неструктурованого тексту в формат, придатний для подальшого аналізу. Дослідницький колектив з Стенфордського університету у складі: C. D. Manning, K. Clark, J. Hewitt, U. Khandelwal та O. Levy у своїй фундаментальній праці визначає токенізацію як процес розбиття тексту на елементарні одиниці (токени), які можуть бути словами, підсловами або символами, залежно від обраного підходу [6]. На рис. 1.4. представлено загальну схему процесу токенізації тексту.

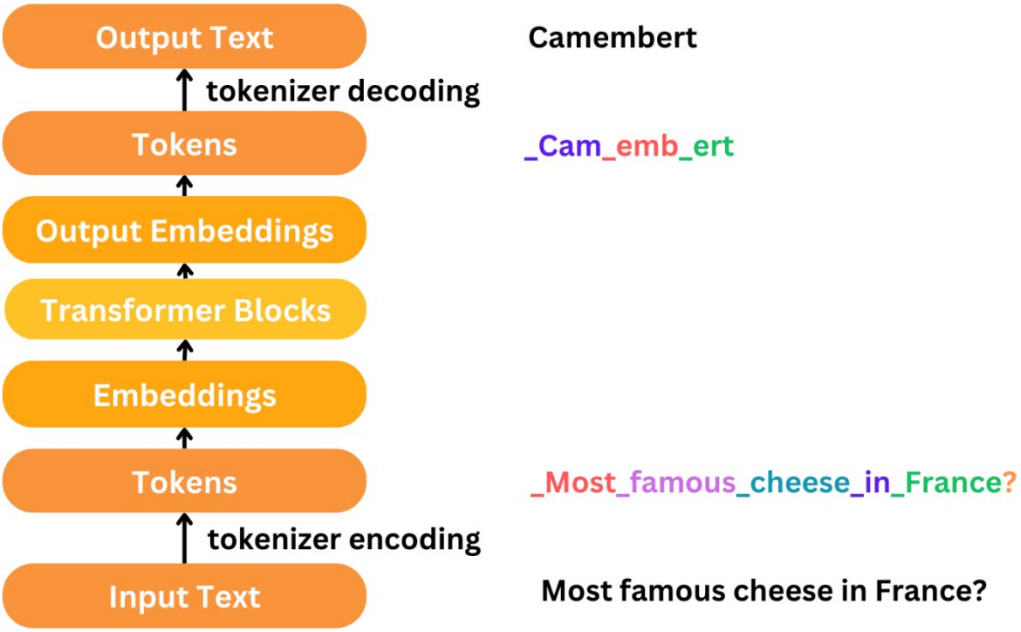


Рисунок 1.4. Схема процесу токенізації тексту

На рис. 1.5. представлено типи токенізації тексту, а саме: словесну токенізацію (word-based), підсловну токенізацію (subword-based) та символьну токенізацію (character-based). Зображення демонструє, як один і той самий текст розбивається на різні типи токенів залежно від обраного методу, а також показує переваги та обмеження кожного підходу.

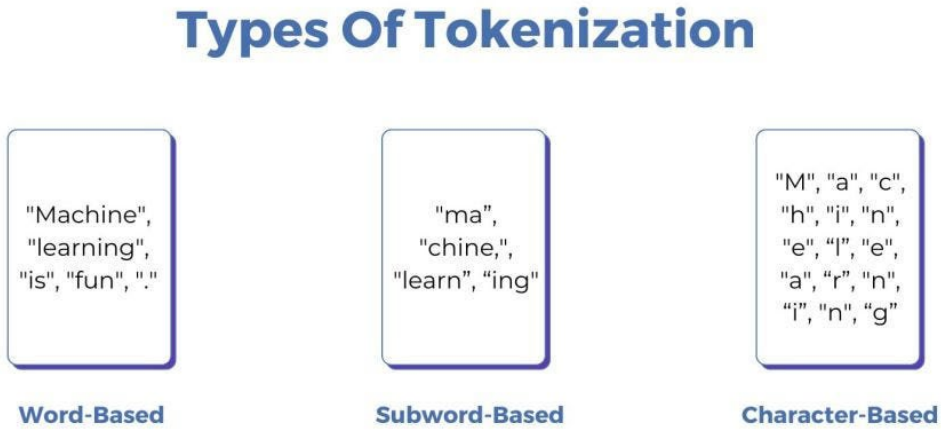


Рисунок 1.5. Схема процесу токенізації тексту

Важливим етапом після токенізації є лематизація – процес приведення слів до їх базової форми (леми). Колектив дослідників з Google Brain, Google Research та University of Toronto, зокрема A. Vaswani, N. Shazerr, N. Parmar, J. Uszkoreit та інші запропонував ефективний підхід до лематизації з використанням механізму трансформерів [7], що описується наступною системою рівнянь:

$$\mathrm{L}(w) = Encoder(w) \oplus \mathrm{Decoder(c)}, \quad (1.4)$$

де, $\mathrm{L}(w)$ – лема для слова w;

$Encoder(w)$ – векторне представлення слова;

$\mathrm{Decoder(c)}$ – декодер контекстної інформації;

$\oplus$ – операція конкатенації векторів.

На рис. 1.6 представлено багатошарову архітектуру нейронної мережі для лематизації, що включає вхідний шар для обробки символів слова, шари LSTM для обробки контексту, та вихідний шар для генерації леми. Особливу увагу приділено механізму уваги, який допомагає мережі фокусуватися на релевантних частинах слова.

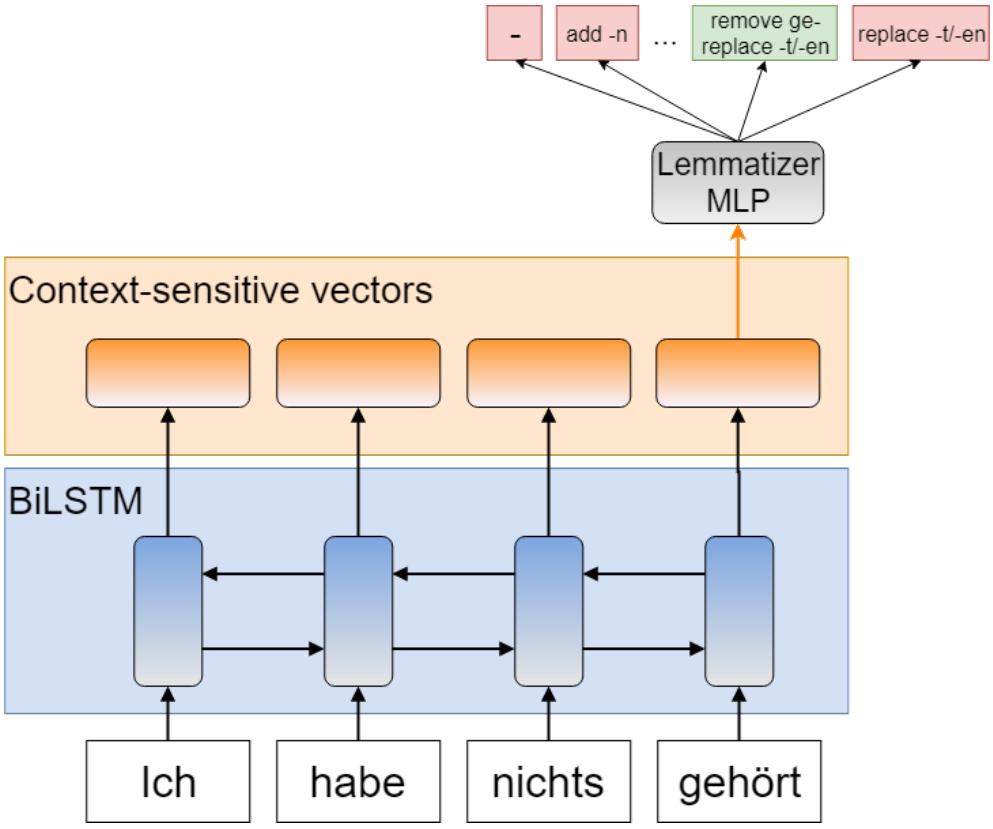


Рисунок 1.6. Архітектура нейронної мережі для лематизації

Науковий колектив з OpenAI: A. Radford, J. Wu, R. Child, D. Luan, D. Amodei та I. Sutskever представив комплексний аналіз ефективності різних підходів до токенізації та лематизації в контексті великих мовних моделей [8], базуючись на метриці якості обробки природної мови:

$$\mathrm{Q} = \frac{(\alpha \times P + \beta \times R)}{(\alpha \times \beta)}, \tag{1.5}$$

де, $Q$ – загальна якість обробки;

$P$ – точність (precision) відновлення базової форми слова;

$R$ – повнота (recall) охоплення морфологічних варіацій;

$\alpha, \beta$ – вагові коефіцієнти для балансування метрик.

Значний внесок у розвиток методів токенізації зробила команда дослідників з Hugging Face на чолі з T. Wolf, спрямовуючи свої зусилля на вдосконалення алгоритмів розбиття тексту на токени — базові елементи, які слугують основою для роботи моделей природної мовної обробки [9]. Завдяки розробкам цієї команди, токенізатори стали більш ефективними, універсальними та оптимізованими для роботи з багатомовними моделями. Вони впровадили нові підходи до сегментації тексту, такі як використання Byte Pair Encoding (BPE), WordPiece, та SentencePiece, що дозволяють забезпечувати точність та компактність представлення текстових даних. Дані інструменти широко використовуються в сучасних мовних моделях, таких як GPT, BERT, і T5.

Семантичний та синтаксичний аналіз є ключовими компонентами в розумінні природної мови, що забезпечують глибинне розуміння як структури, так і значення тексту. Науковий колектив з Stanford University, очолюваний C. Manning та його колегами, визначає синтаксичний аналіз як процес виявлення граматичної структури речення відповідно до певної формальної граматики, тоді як семантичний аналіз фокусується на розумінні значення тексту на різних рівнях – від окремих слів до цілих речень та контексту [6].

На рис. 1.7. представлено приклад синтаксичного дерева розбору речення, де продемонстровано ієрархічну структуру складових речення: від окремих слів

через фрази до повного речення. На схемі також відображено граматичні зв'язки між елементами та їх функціональні ролі в реченні.

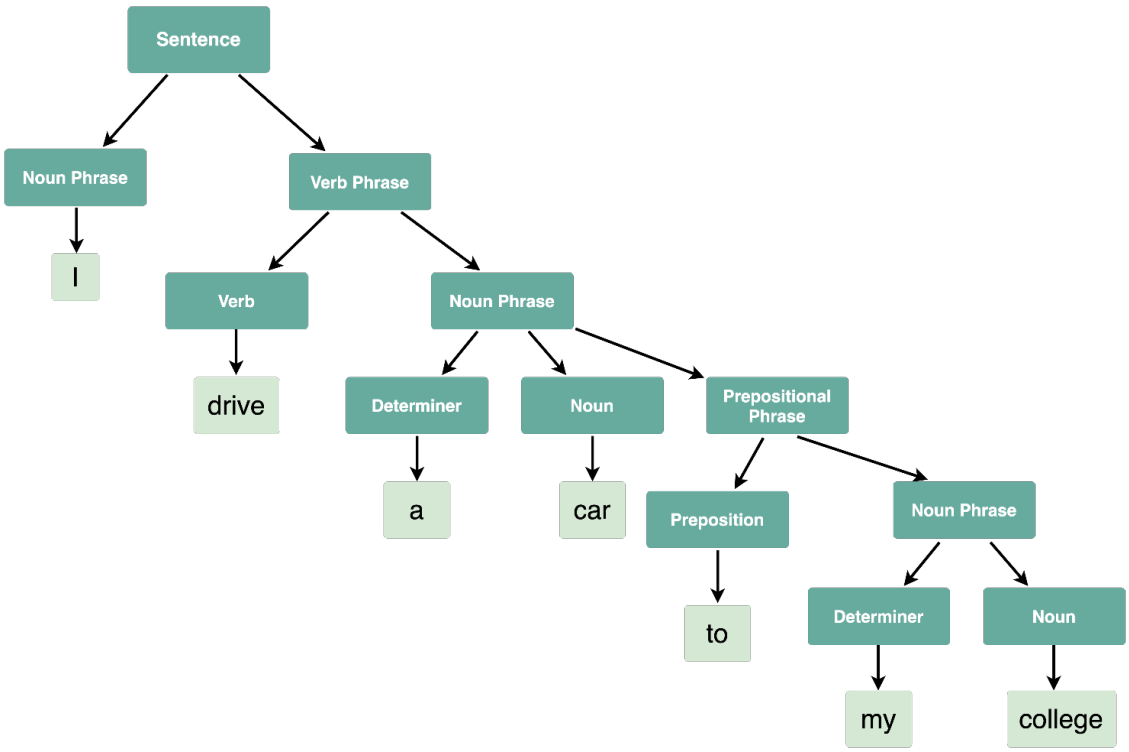


Рисунок 1.7. Приклад синтаксичного дерева

У контексті розвитку методів синтаксичного аналізу особливу увагу привертає робота дослідницької групи з Google AI Language у складі: J. Devlin, M.-W. Chang, L. Lee та K. Touranova. Вчені запропонували революційний підхід до розуміння синтаксичних структур через механізм самоуваги в архітектурі BERT [10]. Цей підхід дозволяє моделі автоматично виявляти синтаксичні залежності між словами без явного програмування граматичних правил.

Семантичний аналіз, за визначенням колективу дослідників з Facebook AI Research на чолі з A. Conneau та K. Khandelwal, можна розділити на кілька рівнів розуміння [11]:

- Лексична семантика – значення окремих слів та їх взаємозв'язки;
- Композиційна семантика – значення окремих слів комбінуються у більші смислові одиниці;
- Прагматична семантика – контекст, що впливає на інтерпретацію значення;

- Дискурсивна семантика – зв’язок значень між реченнями та більшими частинами тексту.

Дослідник Y. Goldberg з Bar Ilan University підкреслив важливість інтеграції синтаксичного та семантичного аналізу для повноцінного розуміння тексту. Його експерименти продемонстрували, як нейронні мережі неявно вивчають як синтаксичні, так і семантичні особливості мови під час попереднього навчання на великих обсягах текстів [12].

На рис. 1.8. представлено концептуальну схему інтеграції синтаксичного та семантичного аналізу в сучасних системах обробки природної мови. Схема демонструє, як різні рівні аналізу взаємодіють між собою та доповнюють один одного для досягнення повного розуміння тексту.

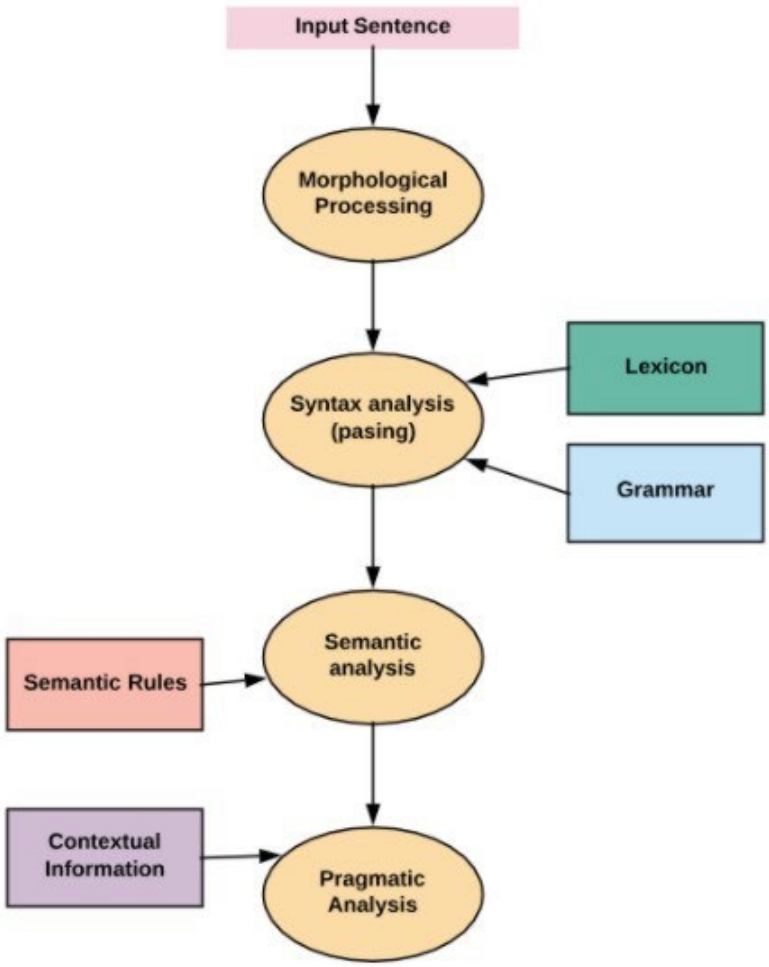


Рисунок 1.8. Концептуальна схема інтеграції синтаксичного та семантичного аналізу в системі обробки природної мови

Значний внесок у розвиток методів семантичного аналізу зробила група дослідників з Language Technologies Institute, Allen Institute for Artificial Intelligence, University of Washington та XNOR.AI.: J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi та N. Smith. Вони розробили підхід до оцінки семантичної узгодженості тексту, який враховує не лише локальні зв'язки між словами, але й глобальний контекст документа [13]. Дослідники виділяють три ключові аспекти семантичного аналізу:

- Послідовність викладу думок;
- Тематична когерентність;
- Логічна несуперечливість.

Архітектура трансформерів стала революційним проривом у сфері обробки природної мови, радикально змінивши підхід до розуміння та генерації тексту. Дослідницький колектив з DeepMind, очолюваний J. Hoffmann та його колегами, представив детальний аналіз масштабування моделей трансформерів, який демонструє, що збільшення розміру моделі та кількості параметрів призводить до емерджентних властивостей – появи нових здібностей, які не були явно запрограмовані [14].

Група науковців з Baidu Inc під керівництвом Y. Sun розробила інноваційний підхід до попереднього навчання трансформерів, що отримав назву ERNIE [15]. Дане дослідження фокусується на вдосконаленні розуміння моделлю контекстуальних зв'язків та семантичних відношень у тексті. Особливу увагу приділено багаторівневій архітектурі уваги, яка дозволяє моделі ефективно обробляти інформацію на різних рівнях абстракції.

На рис. 1.9 відображено багаторівневу архітектуру моделі ERNIE з детальним представленням механізмів уваги на різних рівнях обробки тексту. Схема демонструє, як модель інтегрує знання з різних джерел та рівнів абстракції для покращення розуміння тексту.

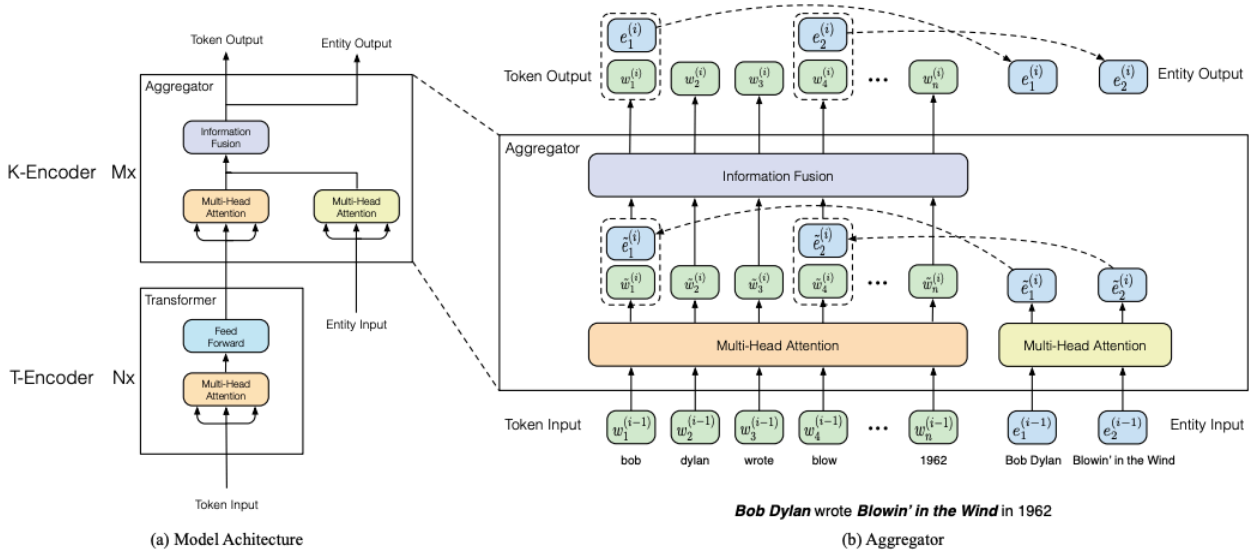


Рисунок 1.8. Архітектура моделі ERNIE

Дослідницький колектив з EleutherAI – S. Black, G. Leo, P. Wang, C. Leahy та S. Biderman представили комплексний аналіз різних архітектур трансформерів, виділяючи три ключові напрямки їх розвитку [16]:

- Оптимізація обчислювальної ефективності;
- Покращення здатності до узагальнення;
- Підвищення інтерпретованості моделей.

Значний внесок у розуміння внутрішніх механізмів роботи трансформерів зробила міжнародна група дослідників з Harvard University: Y. Chen, F. Viégas та M. Wattenberg [17]. Дослідження авторів присвячене аналізу механізмів самоуваги в великих мовних моделях, розкриває, як різні шари трансформера спеціалізуються на обробці різних аспектів мови. Вчені провели масштабні експерименти з візуалізації та аналізу процесів прийняття рішень у різних шарах трансформера, що дозволило виявити спеціалізацію окремих шарів: нижні шари зазвичай фокусуються на синтаксичних особливостях, середні – на семантичних зв'язках, а верхні – на більш абстрактних концепціях та загальному контексті.

Науковий колектив з U.C. Berkeley та Google Research: N. Kitaev, L. Kaiser та A. Levskaya представив новий підхід до створення ефективних

трансформерів з використанням техніки розрідженої уваги, демонструючи, що можливо значно зменшити обчислювальну складність моделі без істотної втрати якості [18].

В той же час, дослідницький колектив з OpenAI, Anthropic AI та Zipline під керівництвом M. Chen зосередив увагу на проблемі надійності та робастності моделей NLP. Вчені виділяють наступні критичні обмеження сучасних систем [19]:

- Вразливість до змін у вхідних даних;
- Нестабільність результатів при незначних модифікаціях запитів;
- Складність верифікації достовірності згенерованих відповідей;
- Обмежена здатність до логічного міркування.

### 1.2. Методи та алгоритми класифікації текстових даних

Класифікація тексту залишається однією з фундаментальних задач у галузі обробки природної мови, де традиційні методи машинного навчання продовжують демонструвати свою ефективність. Дослідницька група з Facebook AI Research – T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch та A. Joulin провела комплексний аналіз традиційних методів класифікації тексту, серед яких: Naive Bayes, Support Vector Machines (SVM), Decision Trees та їх ансамблі, підкреслюючи їх переваги у контексті інтерпретованості та обчислювальної ефективності [20].

Дослідницька група з Carnegie Mellon University та Google Brain на чолі з Z. Dai детально проаналізувала ефективність методу Naive Bayes для класифікації текстів. Особливістю їх дослідження стало вивчення впливу різних припущень про незалежність ознак на якість класифікації [21]. Вони представили модифіковану версію алгоритму, що враховує часткові залежності між словами:

$$\mathrm{P}(C|X) = \frac{P(C)\cdot P(X|C)}{P(X)}, \tag{1.6}$$

де, $P(C|X)$ – ймовірність класу $C$ при спостереженні ознак $X$;

$P(C)$ – апріорна ймовірність класу;

$P(X|C)$ – правдоподібність спостереження ознак $X$ у класі $C$;

$P(X)$ – загальна ймовірність спостереження ознак.

K. Cho та його колеги D. Bahdanau, F. Bougares, H. Schwenk та Y. Bengio з Université de Montréal й Jacobs University представили інноваційний підхід до оптимізації SVM для задач текстової класифікації, фокусуючись на проблемі вибору оптимальної ядерної функції для різних типів текстових даних [22]. Вони запропонували адаптивний метод підбору параметрів ядра:

$$\mathrm{K}(x, y) = e^{-\gamma||x-y||^2}, \tag{1.7}$$

де, $K(x, y)$ – ядерна функція;

$x$, $y$ – вектори ознак;

$\gamma$ – параметр масштабу;

$||x - y||$ – евклідова відстань між векторами.

Група дослідників з Université de Montréal та Canadian Institute for Advanced Research (CIFAR): Y. Bengio, A. Courville та P. Vincent розробила модифіковану версію дерев рішень для текстової класифікації [23]. Розроблений підхід базується на динамічному формуванні правил розбиття з урахуванням семантичної близькості слів:

$$\text{Information Gain} = H(S) - \sum \frac{|S_v|}{S} \cdot H(S_v), \tag{1.8}$$

де, $H(S)$ – ентропія множини $S$;

$S_v$ – підмножина після розбиття;

$|S|$ – розмір множини.

Нейромережеві підходи до класифікації тексту здійснили революційний прорив у галузі обробки природної мови. Дослідницька група з Google Research під керівництвом A. Dosovitskiy представила комплексний аналіз різних архітектур нейронних мереж для задач класифікації тексту, демонструючи їх переваги над традиційними методами машинного навчання [24].

Науковці P. Shaw, J. Uszkoreit та A. Vaswani з Google й Google Brain зосередили увагу на рекурентних нейронних мережах (RNN) та їх варіаціях для

текстової класифікації [25]. Вони представили модифіковану архітектуру LSTM (Long Short-Term Memory), що може бути описана наступними рівняннями:

$$\mathrm{f}_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \quad (1.9)$$

$$\mathrm{i}_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i),$$

$$\mathrm{c}_t = f_t * c_{t-1} + i_t * \tan(W_c \cdot [h_{t-1}, x_t] + b_c),$$

$$\mathrm{o}_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o),$$

$$\mathrm{h}_t = o_t * \tan(c_t),$$

де, $f_t$ – вектор забування;

$\mathrm{i}_t$ – вхідний вектор;

$\mathrm{c}_t$ – стан комірки пам’яті;

$\mathrm{o}_t$ – вихідний вектор;

$\mathrm{h}_t$ – прихований стан;

$\sigma$ – сигмоїдна функція активації.

На рис. 1.9 показано детальну архітектуру LSTM-комірки з візуалізацією всіх внутрішніх компонентів та потоків даних. Схема демонструє механізми забування, оновлення та виведення інформації.

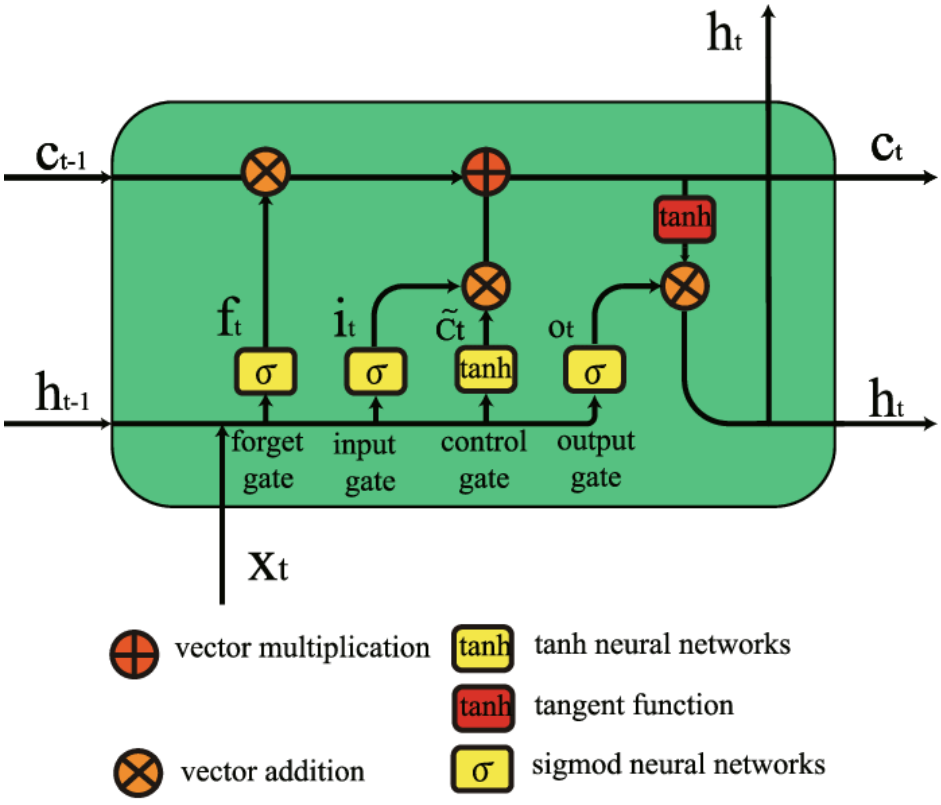


Рисунок 1.9. Архітектура LSTM-комірки

Команда дослідників з Google – N. Shazeer, Z. Lan, Y. Cheng, N. Ding та L. Hou представила інноваційний підхід до використання згорткових нейронних мереж (CNN) для класифікації тексту [26]. Запропонована науковцями архітектура використовує багатошарові згортки для виявлення локальних патернів у тексті:

$$\mathrm{h} = ReLU(Conv1D(x, W) + b), \tag{1.9}$$

де, $x$ – вхідна послідовність;

$W$ – матриця ваг згорткового шару;

$b$ – вектор зміщення;

$ReLU$ – функція активації;

На рис. 1.10. представлено архітектуру згорткової мережі для обробки тексту, де показано процес згортки та виділення ознак на різних рівнях абстракції.

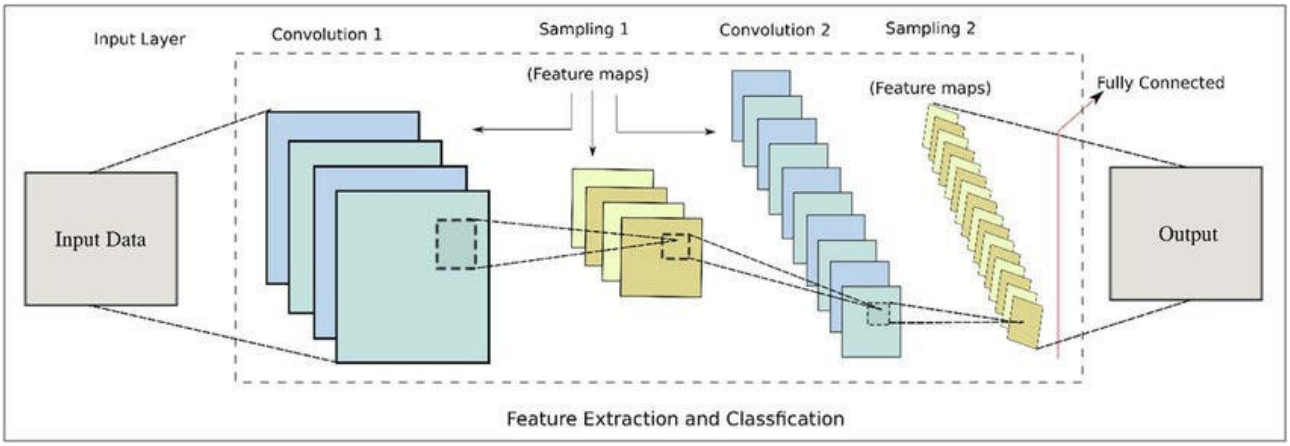


Рисунок 1.10. Класифікація тексту за допомогою згорткової нейромережі

Дослідницький колектив з Google та University of Toronto – I. Sutskever, J. Martens, G. Dahl та G. Hinton провів глибокий аналіз впливу різних функцій активації та методів регуляризації на якість класифікації у нейронних мережах. Основна увага була зосереджена на оптимізації функції втрат, яка враховує як похибку прогнозування, так і регуляризацію вагів для запобігання перенавчанню [27].

Формула функції втрат має вигляд:

$$L = -\sum y_i \log(\hat{y}_i) + \lambda \sum ||W||^2, \tag{1.10}$$

де, $L$ – загальна функція втрат, яка використовується для оцінки ефективності моделі;

$y_i$ – істинні мітки класів, що відповідають кожному зразку даних;

$\hat{y}_i$ – передбачені ймовірності, які генерує модель для кожного класу i;

$\lambda$ – коефіцієнт регуляризації, що визначає ступінь впливу регуляризаційного члена;

$W$ – ваги нейронної мережі, які підлягають регуляризації.

У своїй роботі команда показала, що правильний вибір функції активації, наприклад ReLU або Swish, значно покращує здатність моделі до узагальнення. Крім того, регуляризація вагів за допомогою L2-норми, яка враховується у другому члені функції втрат, допомагає зменшити складність моделі, запобігаючи перенавчанню та покращуючи її стабільність.

Дослідники Guo Yang, Yan Jiayu, Xu Dongdong, Guo Zelin & Huan Hai [28], розробили новий підхід до класифікації довгих текстових послідовностей. їх робота демонструє, як спеціалізовані архітектури можуть ефективно обробляти документи значної довжини, зберігаючи контекстуальні зв’язки та важливі семантичні особливості тексту.

На рис. 1.11 представлено архітектуру обробки довгих текстових послідовностей, де показано механізми збереження контексту та виділення ключових особливостей тексту на різних рівнях абстракції.

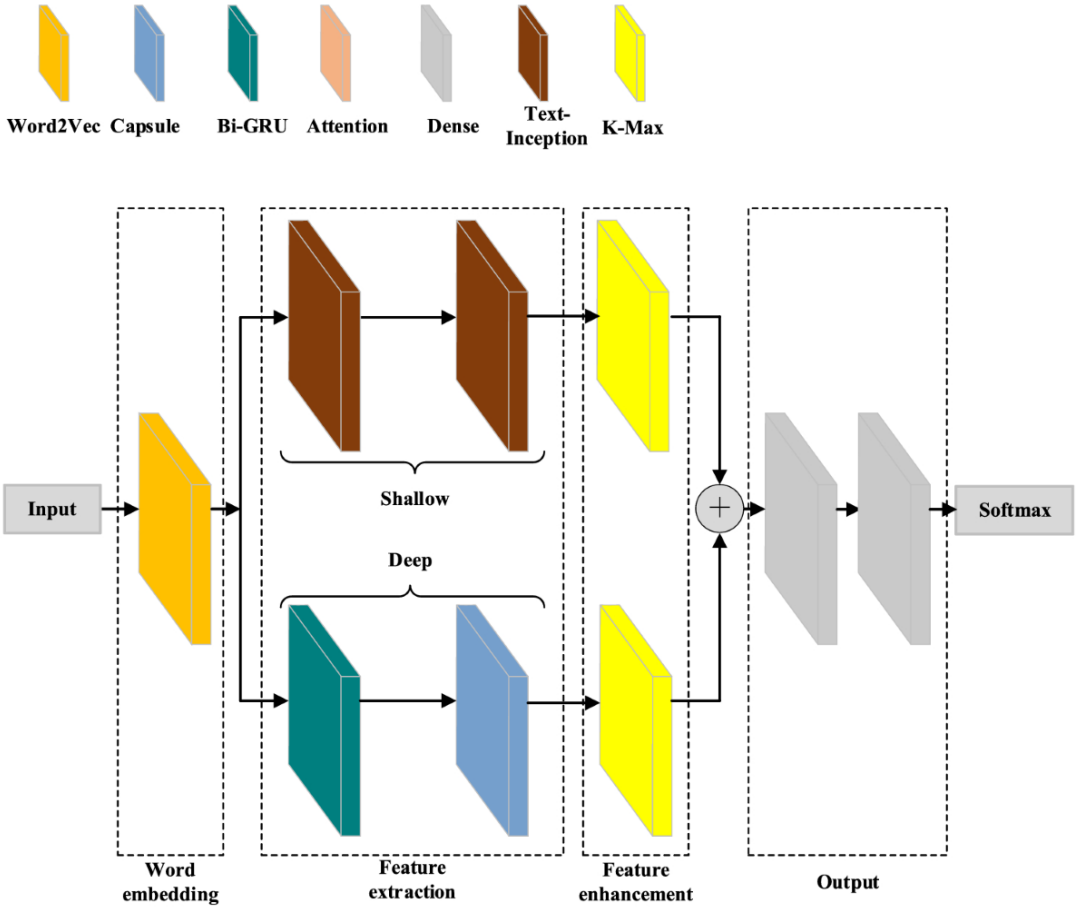


Рисунок 1.11. Архітектура обробки довгих текстових послідовностей

Оцінка якості класифікації є критичним аспектом у розробці та впровадженні систем обробки природної мови. Дослідницька група з U.C. Berkeley та Google Research – S. Levine, A. Kumar, G. Tucker та J. Fu представила комплексний аналіз різних метрик оцінки, підкреслюючи важливість вибору правильних критеріїв для різних типів задач класифікації [29].

На рис. 1.12 представлено візуалізацію основних метрик оцінки якості класифікації та їх взаємозв'язків. Схема демонструє, як різні метрики доповнюють одна одну та дають повну картину продуктивності моделі

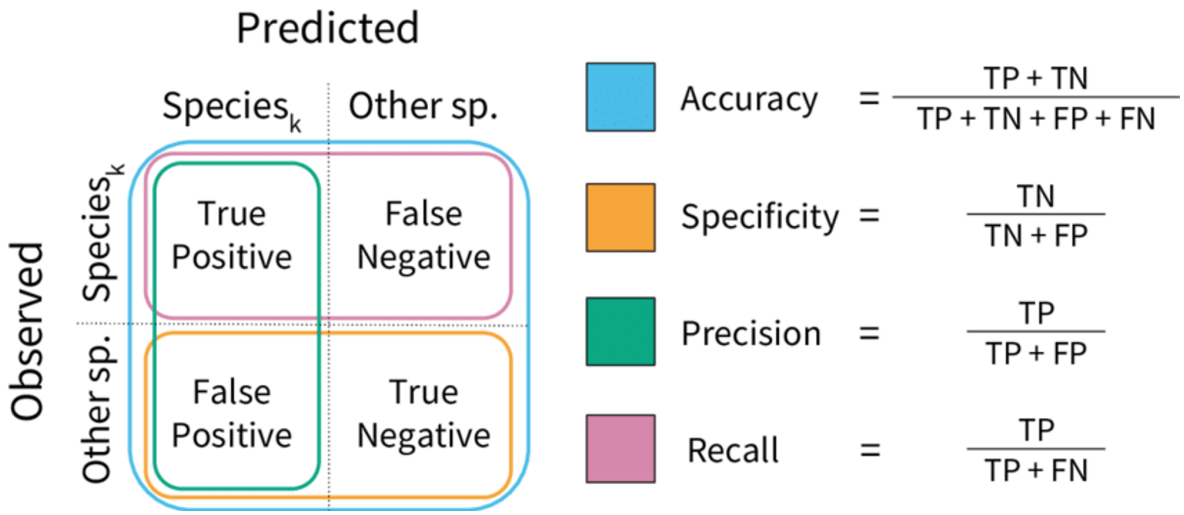


Рисунок 1.12. Основні метрики оцінки якості класифікації

Базові метрики включають:

1) Прецизійність

$$\text{Precision} = \frac{\text{TP}}{TP+FP}, \tag{1.11}$$

2) Повноту

$$Recall = \frac{\text{TP}}{TP+FN}, \tag{1.12}$$

3) F1-міру

$$\text{F1} - Score = \frac{2 \times (\text{Precision} \times \text{Recall})}{Precision + Recall}, \tag{1.13}$$

4) Точність:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \tag{1.14}$$

де, *TP* – істинно позитивні результати;

*TN* – істинно негативні результати;

*FP* – хибно позитивні результати;

*FN* – хибно негативні результати.

Науковий колектив з Stanford University та Facebook AI Research – C. Potts, Z. Wu, A. Giger та D. Kiela розробив методологію оцінки якості класифікації для незбалансованих наборів даних [30]. Вони запропонували використання зваженої F1-міри та площі під ROC-кривою (AUC-ROC) для більш об’єктивної оцінки продуктивності моделей.

На рис. 1.13 показано порівняння ROC-кривих для різних випадків класифікації, де демонструється залежність між чутливістю та специфічністю класифікатора при різних порогових значеннях.

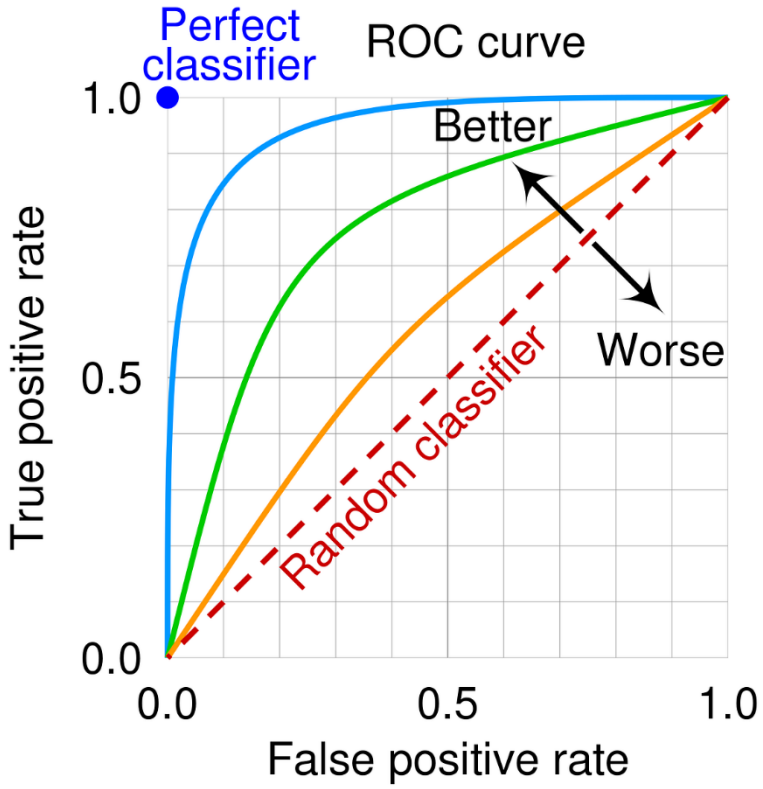


Рисунок 1.13. Порівняння ROC-кривих

Команда дослідників з Stanford University під керівництвом R. Socher розробила комплексний підхід до оцінки робастності класифікаторів [31]:

- Тестування на зашумлених даних;
- Оцінку стійкості до adversarial attacks;
- Аналіз поведінки при граничних випадках;
- Вимірювання стабільності результатів.

### 1.3. Аналіз існуючих систем маршрутизації клієнтських запитів

Zendesk AI пропонує комплексне рішення для автоматизації обслуговування клієнтів, інтегруючи інструменти штучного інтелекту для підвищення ефективності взаємодії з користувачами [32]. Основні компоненти:

- Answer Bot – автоматично відповідає на поширені запитання, що дозволяє знизити навантаження на команду підтримки.
- Маршрутизація на основі машинного навчання – запити клієнтів розподіляються до відповідних агентів на основі їх специфікацій та історії взаємодії.
- Аналітика взаємодії з клієнтами – забезпечує глибокий аналіз поведінки користувачів, їх запитів і ключових метрик ефективності.
- Інтеграція з різними каналами комунікації – платформа підтримує роботу з email, чатами, соціальними мережами та іншими каналами.

Intercom Resolution Bot є спеціалізованим рішенням для обробки клієнтських запитів, що спрямоване на забезпечення персоналізованого та ефективного обслуговування [33]. Основні характеристики:

- Проактивна взаємодія з клієнтами – бот ініціює комунікацію, передбачаючи потреби користувача.
- Персоналізовані відповіді – адаптується до контексту клієнта, використовуючи дані про його історію взаємодії.
- Багатомовна підтримка – забезпечує обслуговування клієнтів на різних мовах.
- Інтеграція з CRM-системами – дозволяє використовувати інформацію з бази даних для підвищення точності відповідей.

Freddy AI від Freshdesk забезпечує автоматизацію процесів підтримки клієнтів за допомогою інтелектуальних рішень [34]. Основні особливості:

- Автоматична категоризація та пріоритизація запитів – система автоматично визначає тип і рівень пріоритетності кожного запиту.
- Інтелектуальна маршрутизація – направляє запити до відповідних агентів на основі їх експертизи.
- Передбачення намірів клієнта – використовує моделі машинного навчання для аналізу контексту повідомлень.

- Аналіз настроїв у повідомленнях – визначає емоційний тон клієнтських запитів для адаптації відповіді.

HubSpot Service Hub використовує можливості обробки природної мови (NLP) для автоматизації ключових аспектів клієнтської підтримки [35]. Основні функції:

- Автоматична класифікація вхідних запитів – система швидко і точно визначає тип запиту.
- Створення бази знань – генерує статті та довідкову інформацію на основі історичних даних.
- Оптимізація розподілу навантаження – розподіляє запити між агентами з урахуванням їх завантаженості.
- Аналіз ефективності служби підтримки – надає звіти про продуктивність та основні показники команди.

Salesforce Einstein забезпечує автоматизацію обслуговування клієнтів через інноваційні AI-інструменти [36]. Основні можливості:

- Предиктивна маршрутизація запитів – система прогнозує, який агент зможе найкраще вирішити запит.
- Автоматичне тегування та категоризація – спрощує класифікацію великої кількості запитів.
- Рекомендації щодо наступних дій – допомагає агентам швидко реагувати на запити клієнтів.
- Аналітика клієнтського досвіду – надає візуалізовані дані про задоволеність і поведінку клієнтів.

Для комплексного аналізу існуючих рішень доцільно розглянути їх характеристики за кількома ключовими критеріями (Функціональні можливості систем – табл. 1.1.; специфічні характеристики обробки запитів – табл. 1.2.).

Таблиця 1.1. Порівняння функціональних можливостей систем

| **Критерій** | **Zendesk AI** | **Intercom Resolution Bot** | **Freddy AI** | **HubSpot Service Hub** | **Salesforce Einstein** |
|---|---|---|---|---|---|
| Автоматичні відповіді | Answer Bot з базою знань | Проактивні відповіді | Категоризовані відповіді | Відповіді на основі бази знань | Предиктивні відповіді |
| Маршрутизація | На основі ML | На основі контексту | Інтелектуальна на основі експертизи | За навантаженням | Предиктивна |
| Аналітика | Базова аналітика взаємодій | Інтегрована з CRM | Аналіз настроїв | Комплексна звітність | Розширена аналітика досвіду |
| Мультиканальність | Висока | Середня | Висока | Середня | Висока |
| Інтеграції | Широкі можливості | Фокус на CRM | Помірні | Екосистема HubSpot | Екосистема Salesforce |

Таблиця 1.2. Порівняння специфічних характеристик обробки запитів

| **Характеристика** | **Zendesk AI** | **Intercom Resolution Bot** | **Freddy AI** | **HubSpot Service Hub** | **Salesforce Einstein** |
|---|---|---|---|---|---|
| Точність класифікації | Висока | Середня | Висока | Висока | Дуже висока |
| Швидкість обробки | Швидка | Дуже швидка | Швидка | Середня | Швидка |
| Обробка складних запитів | Обмежена | Обмежена | Помірна | Помірна | Розширена |
| Самонавчання системи | Так | Обмежено | Так | Так | Так |
| Багатомовність | 40+ мов | 30+ мов | 35+ мов | 25+ мов | 50+ мов |

На основі проведеного порівняльного аналізу можна зробити висновок, що кожна система має свої переваги та обмеження, які слід враховувати при виборі рішення залежно від конкретних потреб. Zendesk AI та Salesforce Einstein пропонують найбільш комплексні рішення для великих підприємств та є гарними орієнтирами для виконання даного дослідження.

**Висновки до розділу 1**

Отже, дослідження теоретичних основ обробки природної мови та класифікації клієнтських запитів виявило кілька ключових аспектів. По-перше, сучасні підходи до обробки природної мови базуються на архітектурі трансформерів, яка забезпечує ефективну обробку текстової інформації через механізми самоуваги та багаторівневого аналізу. Важливими етапами обробки тексту є токенізація та лематизація, які забезпечують перетворення неструктурованого тексту в формат, придатний для аналізу.

Методи класифікації текстових даних еволюціонували від традиційних підходів (Naive Bayes, SVM, Decision Trees) до складних нейромережевих архітектур, включаючи RNN, LSTM та CNN. Особлива увага приділяється оцінці якості класифікації через метрики precision, recall, F1-score та ROC-AUC, що дозволяє об’єктивно оцінювати ефективність моделей.

Аналіз існуючих систем маршрутизації клієнтських запитів (Zendesk AI, Intercom Resolution Bot, Freddy AI, HubSpot Service Hub, Salesforce Einstein) показав різноманітність підходів до автоматизації обслуговування клієнтів. Найбільш комплексні рішення пропонують Zendesk AI та Salesforce Einstein, які використовують передові технології NLP для забезпечення ефективної класифікації та маршрутизації запитів, автоматичних відповідей та аналітики клієнтського досвіду і є гарними орієнтирами для реалізації нашої системи.

# РОЗДІЛ 2.
# МЕТОДОЛОГІЧНІ ЗАСАДИ РОЗРОБКИ АВТОМАТИЗОВАНОЇ СИСТЕМИ КЛАСИФІКАЦІЇ ЗАПИТІВ НА ОСНОВІ ОБРОБКИ ПРИРОДНОЇ МОВИ

## 2.1. Архітектура системи класифікації та маршрутизації запитів на основі обробки природної мови

Автоматизована система класифікації та маршрутизації запитів клієнтів представляє собою комплексне програмне рішення, що базується на модульній архітектурі з чітким розподілом відповідальності між компонентами. В основі архітектурного рішення лежить принцип мікросервісної організації, що забезпечує високу гнучкість та масштабованість системи, дозволяючи ефективно обробляти різноманітні типи клієнтських запитів незалежно від джерела їх надходження.

Модуль прийому запитів виступає єдиною точкою входу для всіх клієнтських звернень та реалізує уніфікований інтерфейс обробки вхідних даних. Архітектурне рішення цього модуля передбачає можливість одночасної обробки звернень, що надходять через різні канали комунікації, такі як електронна пошта, месенджери та соціальні мережі. Важливою особливістю даного модуля є реалізація механізму первинної валідації даних, що дозволяє на ранньому етапі відфільтровувати некоректні або потенційно небезпечні запити. Кожному вхідному зверненню присвоюється унікальний ідентифікатор, що забезпечує можливість відстеження його статусу протягом всього життєвого циклу обробки. Рис. 2.1 демонструє загальну архітектуру системи, що розробляється.

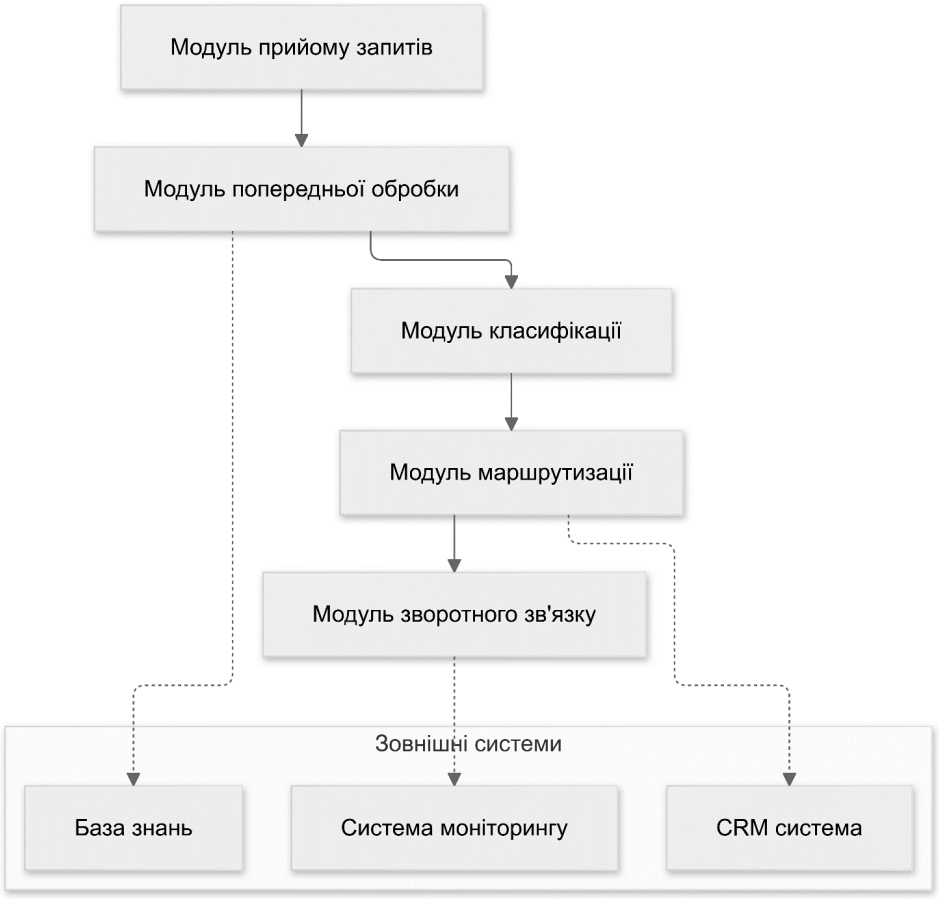


Рисунок 2.1. Загальна архітектура системи, що розробляється

Модуль попередньої обробки є критично важливим компонентом системи, що забезпечує якісну підготовку текстових даних для подальшої класифікації. В рамках цього модуля реалізовано складний процес трансформації неструктурованого тексту в формат, придатний для машинного аналізу. Процес включає послідовність операцій з нормалізації тексту, під час якої відбувається приведення всіх символів до єдиного регістру та видалення спеціальних символів, що не несуть смислового навантаження. Наступним етапом є глибока очистка тексту від шумів та неінформативних елементів, включаючи видалення стоп-слів та виправлення типових орфографічних помилок. Рис. 2.2 демонструє процес попередньої обробки тексту.


Рисунок 2.2. Процес попередньої обробки тексту

Модуль класифікації представляє собою ядро системи, що реалізує складні алгоритми машинного навчання для аналізу та категоризації клієнтських запитів. Даний модуль використовує попередньо навчену модель, яка здатна визначати не лише основну категорію запиту, але й оцінювати рівень впевненості у класифікації, що є критично важливим для прийняття подальших рішень щодо маршрутизації. Процес класифікації також включає виділення ключових сутностей з тексту запиту, що дозволяє формувати більш точне розуміння контексту звернення та його пріоритетності. Рис. 2.3 демонструє процес обробки запиту.

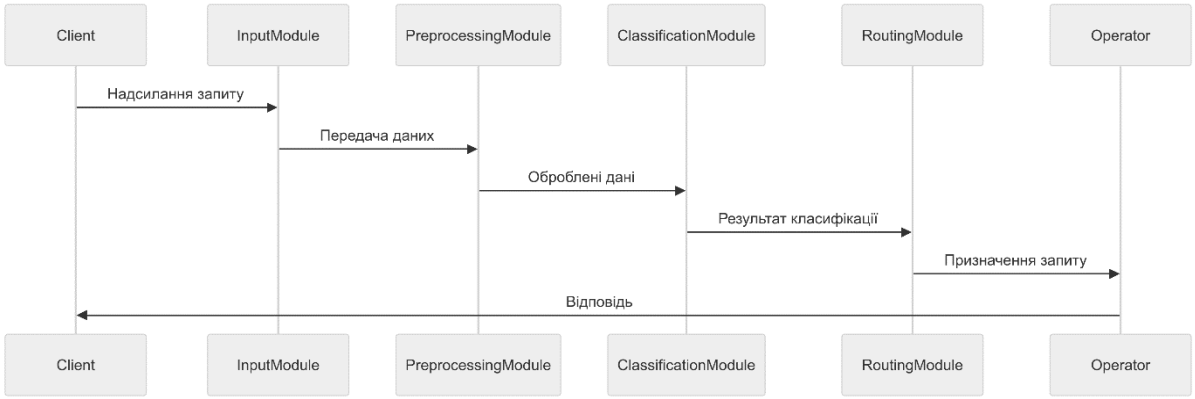


Рисунок 2.3. Процес обробки запиту

Модуль маршрутизації реалізує інтелектуальний механізм розподілу запитів між операторами або автоматичними системами обробки. В основі роботи модуля лежить складний алгоритм, що враховує множину факторів: результати класифікації, поточне навантаження на операторів, їх компетенції та спеціалізацію, історію взаємодії з конкретним клієнтом, а також пріоритетність запиту. Особлива увага приділяється оптимізації розподілу навантаження, що досягається шляхом використання адаптивних алгоритмів балансування. Рис. 2.4 демонструє схему бази даних системи.

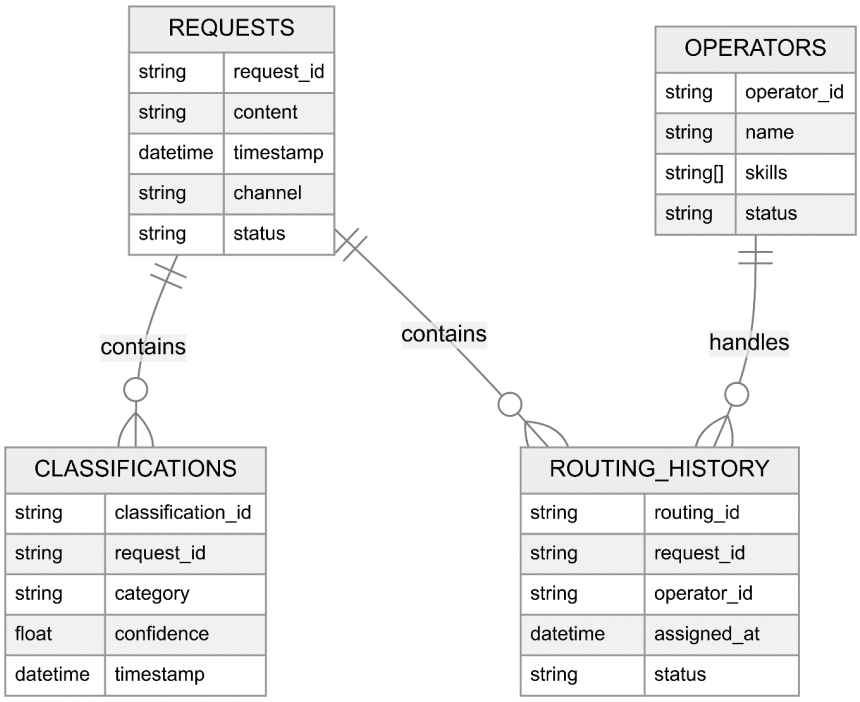


Рисунок 2.4. Схема бази даних системи

Модуль зворотного зв’язку забезпечує комплексний аналіз ефективності роботи системи через збір та обробку різноманітних метрик. Цей модуль реалізує механізми оцінки якості обробки запитів, включаючи аналіз часу відповіді, точності класифікації та рівня задоволеності клієнтів. Особливістю даного модуля є здатність агрегувати дані з різних джерел та формувати комплексні аналітичні звіти, які використовуються як для оперативного моніторингу роботи системи, так і для її довгострокового вдосконалення.

Взаємодія між усіма модулями системи реалізована через стандартизовані API-інтерфейси, що забезпечує високу гнучкість та можливість незалежного оновлення окремих компонентів. Система спроектована з урахуванням вимог до інформаційної безпеки та захисту персональних даних, реалізуючи багаторівневу систему автентифікації та авторизації для всіх операцій з даними. Архітектурне рішення передбачає можливість горизонтального масштабування системи для забезпечення обробки зростаючого обсягу запитів без втрати продуктивності.

Система обробки вхідних даних представляє собою комплексний механізм, який забезпечує уніфіковану обробку різнотипних клієнтських запитів, що надходять через множину каналів комунікації. Архітектурна реалізація даного компонента передбачає наявність спеціалізованих адаптерів для кожного типу вхідного каналу, що забезпечує максимальну гнучкість та розширюваність системи при додаванні нових джерел надходження запитів. Рис. 2.5 демонструє архітектуру обробки вхідних даних

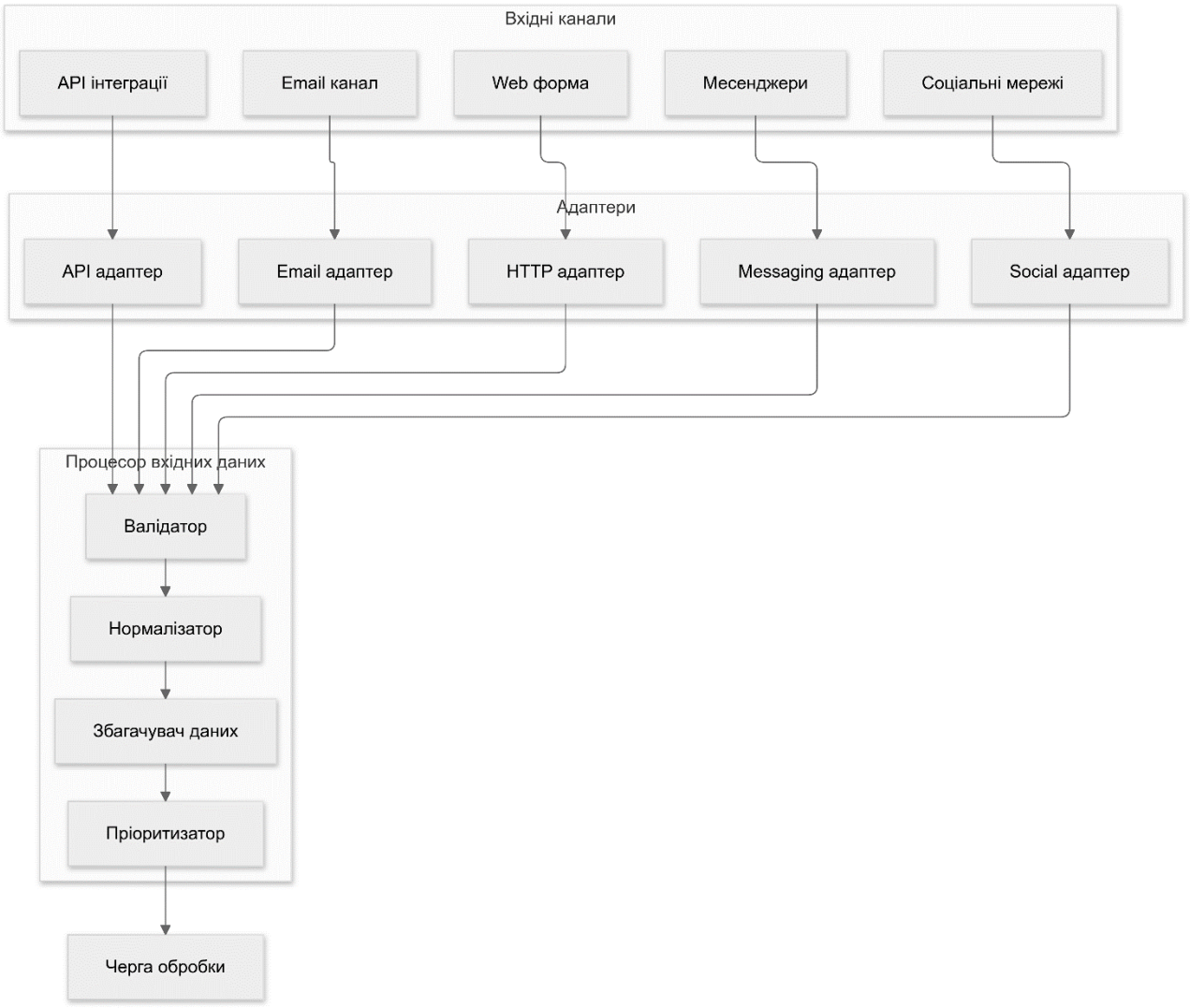


Рисунок 2.5. Архітектура обробки вхідних даних

Процес обробки вхідних даних реалізований у вигляді послідовності етапів, кожен з яких відповідає за специфічний аспект підготовки даних для подальшої класифікації. Розглянемо детальніше послідовність обробки вхідних даних на прикладі конкретного запиту. Рис. 2.6 демонструє послідовність обробки запиту.

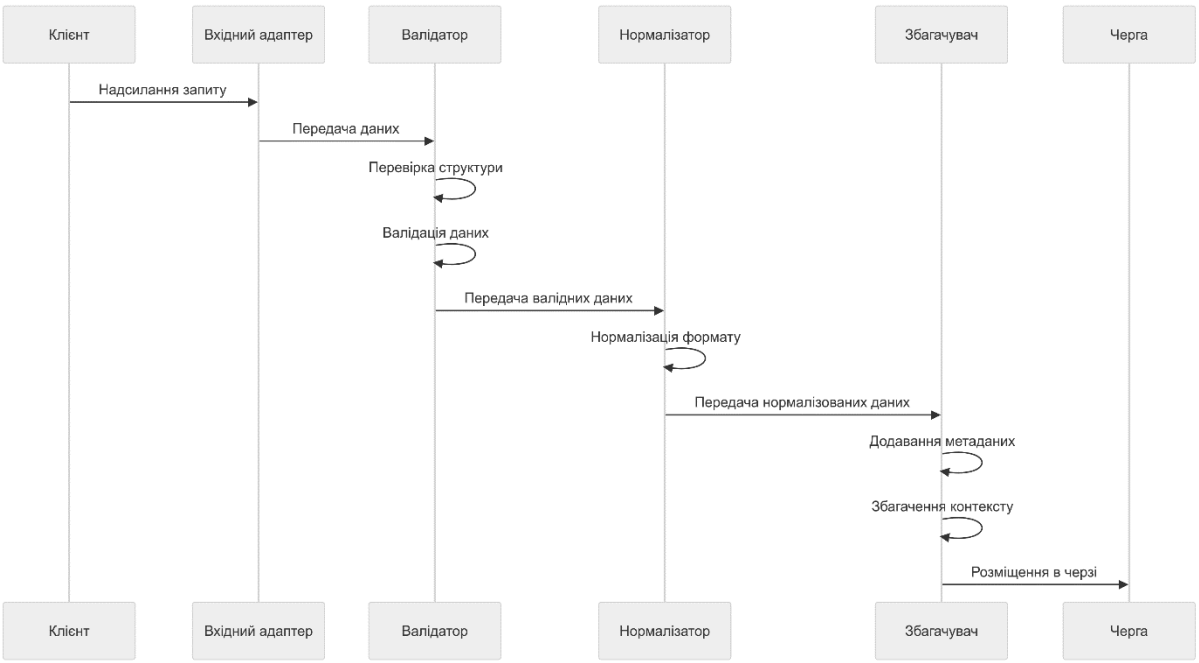


Рисунок 2.6. Послідовність обробки запиту

Для забезпечення надійності та відмовостійкості системи реалізовано механізм буферизації вхідних даних, який базується на розподіленій черзі повідомлень. Архітектура черги забезпечує гарантовану доставку повідомлень та їх обробку навіть у випадку тимчасової недоступності окремих компонентів системи. Рис. 2.7 демонструє архітектуру черги повідомлень.

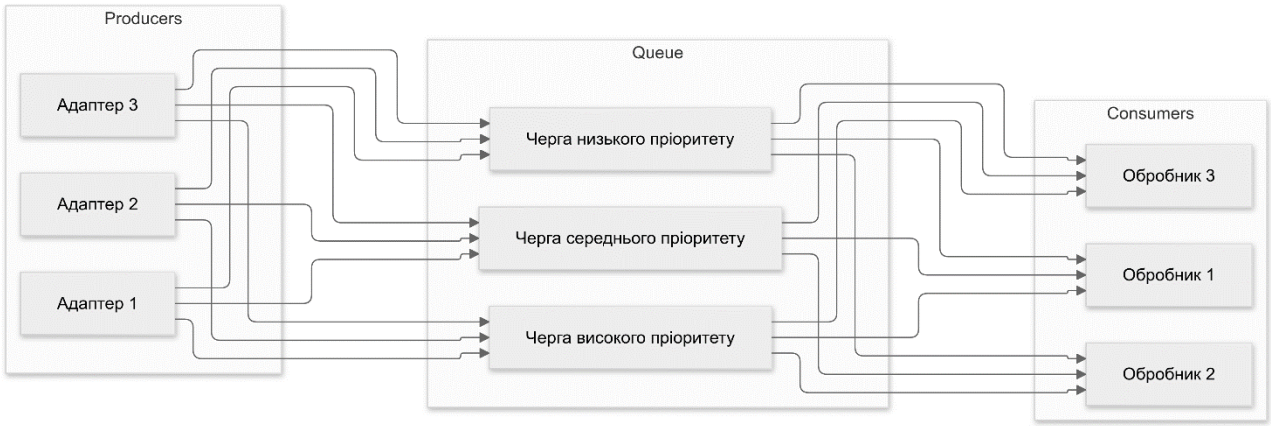


Рисунок 2.7. Архітектура черги повідомлень

Особлива увага в системі приділяється механізму валідації вхідних даних, який забезпечує перевірку структурної цілісності запитів та їх відповідність встановленим критеріям якості. Валідатор реалізує багаторівневу систему

перевірок, що включає синтаксичний аналіз, перевірку обов'язкових полів, валідацію форматів даних та перевірку бізнес-правил. Рис. 2.8 демонструє процес валідації даних.


Рисунок 2.8. Процес валідації даних

Для забезпечення ефективної обробки різнотипних запитів система реалізує механізм динамічного масштабування ресурсів, що дозволяє адаптуватися до змінного навантаження та забезпечувати стабільну роботу при пікових навантаженнях. Архітектурне рішення передбачає можливість горизонтального масштабування як на рівні адаптерів, так і на рівні процесорів обробки даних.

Модуль класифікації запитів є центральним компонентом системи, що реалізує інтелектуальний аналіз та категоризацію клієнтських звернень на основі їх змісту та контексту. Архітектурна реалізація даного модуля базується на багаторівневому підході до аналізу даних, що дозволяє досягти високої точності класифікації при збереженні ефективності обробки запитів. Рис. 2.9 демонструє архітектуру модуля класифікації.

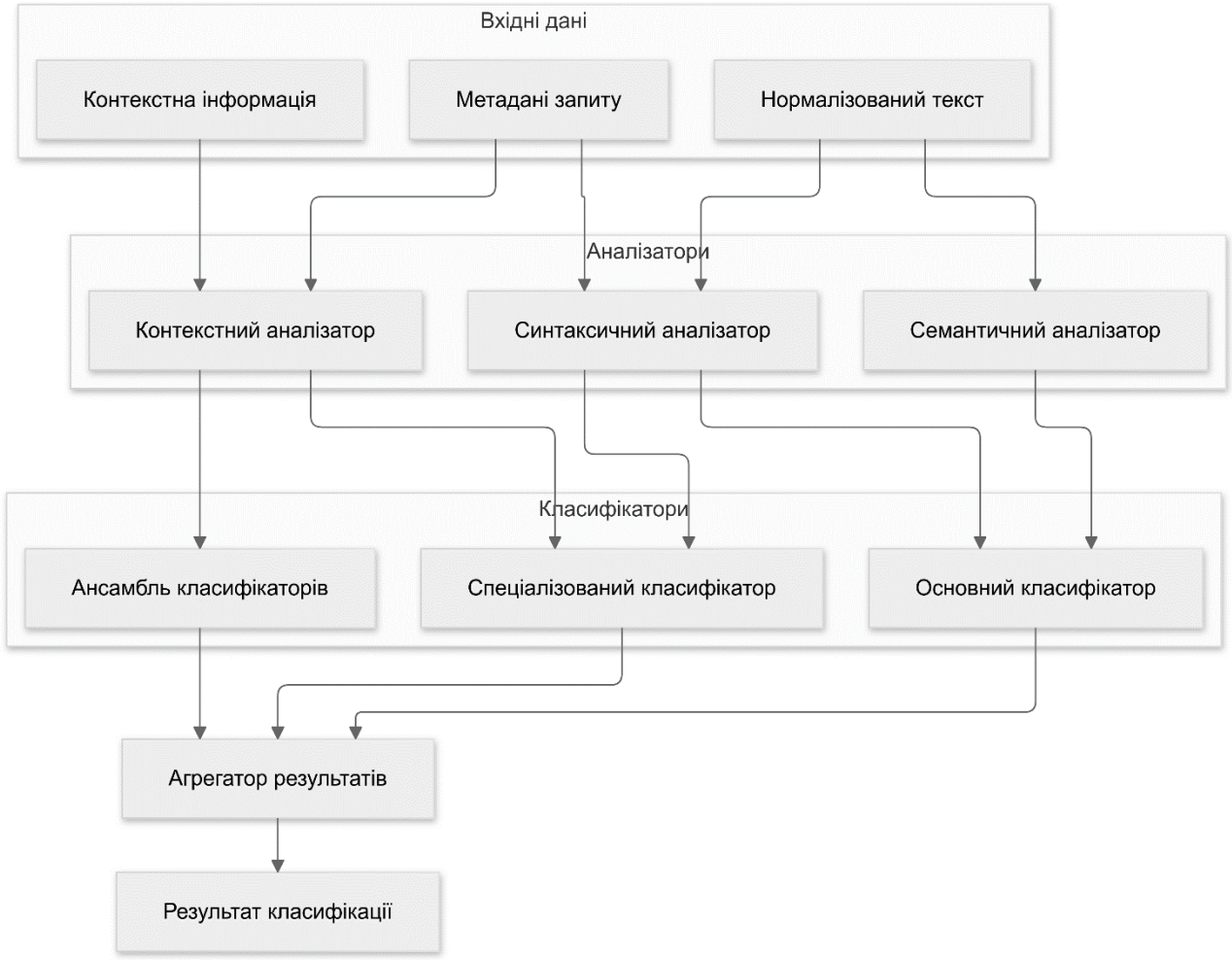


Рисунок 2.9. Архітектура модуля класифікації

Процес класифікації реалізований як послідовність етапів обробки, де кожен етап збагачує розуміння системою змісту запиту. Розглянемо детальну послідовність обробки запиту в рамках модуля класифікації (рис. 2.10).

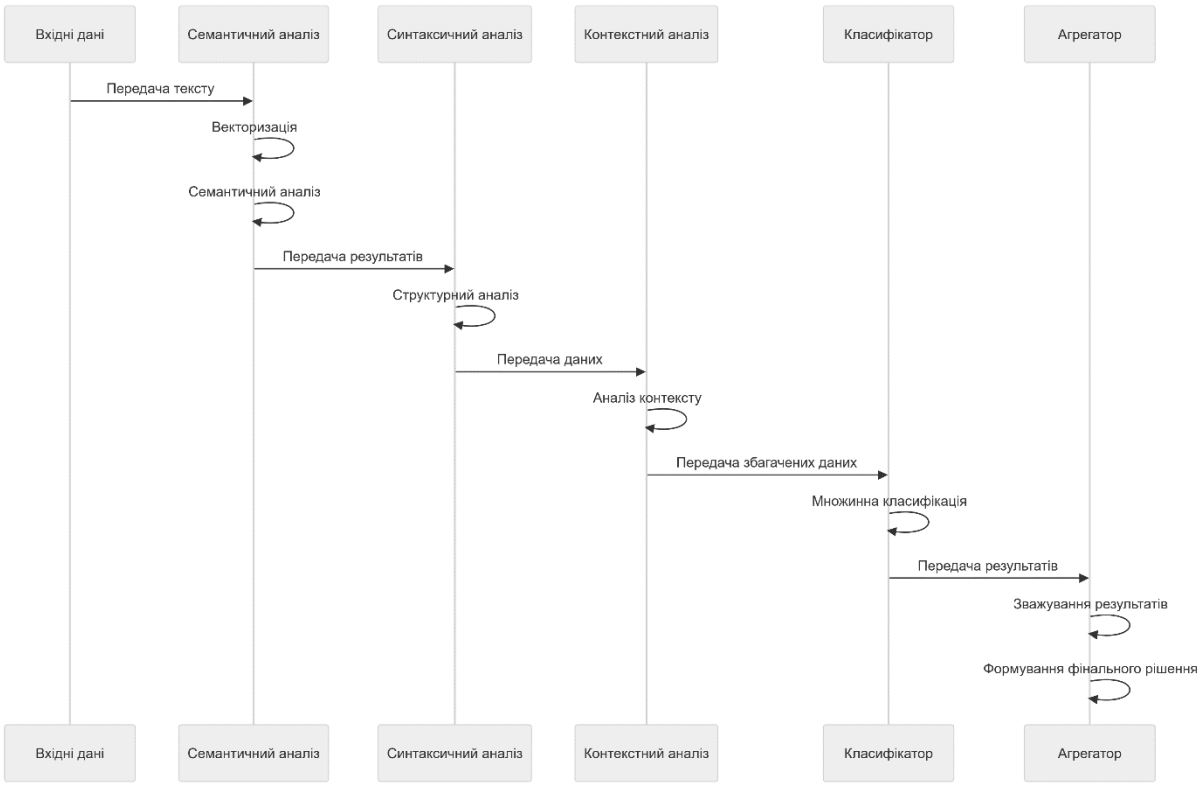


Рисунок 2.10. Послідовність класифікації

Для забезпечення надійності та точності класифікації система використовує ансамбль моделей машинного навчання, кожна з яких спеціалізується на певному аспекті аналізу тексту (рис. 2.11):

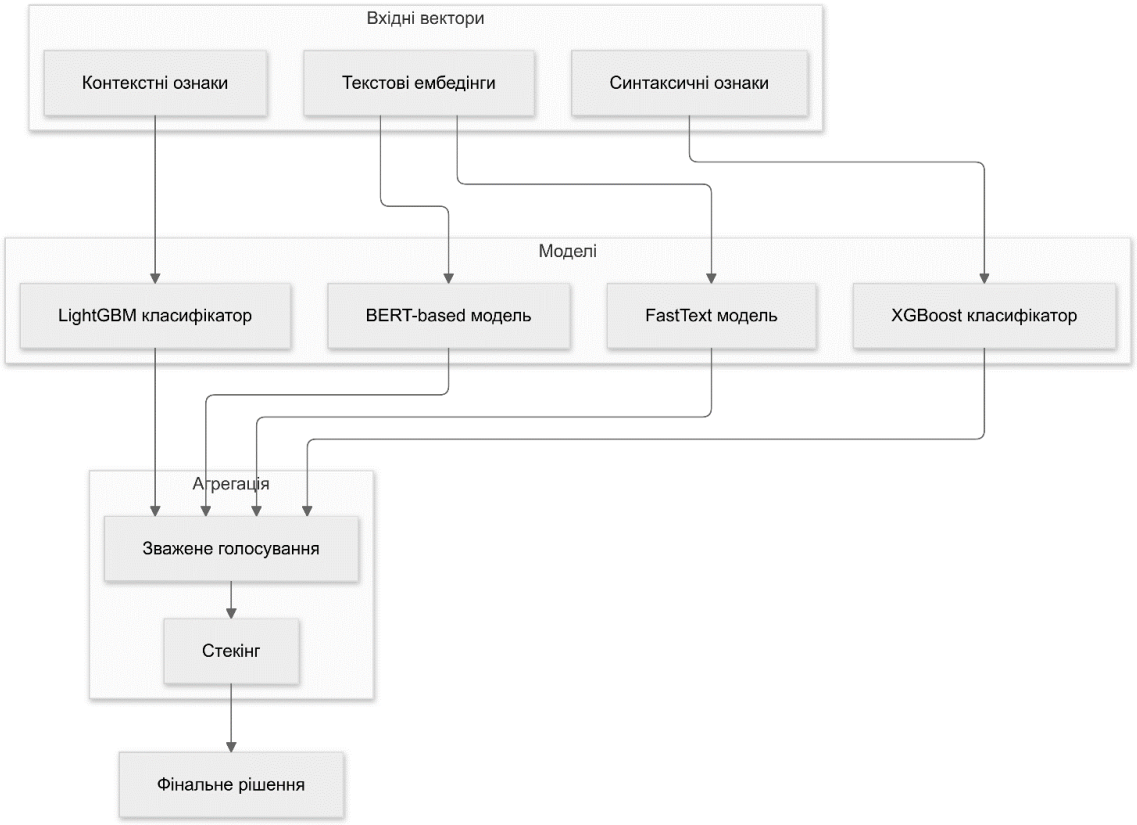


Рисунок 2.11. Ансамбль моделей класифікації

Система забезпечує постійне вдосконалення якості класифікації через механізм зворотного зв’язку та автоматичного донавчання моделей (2.12):

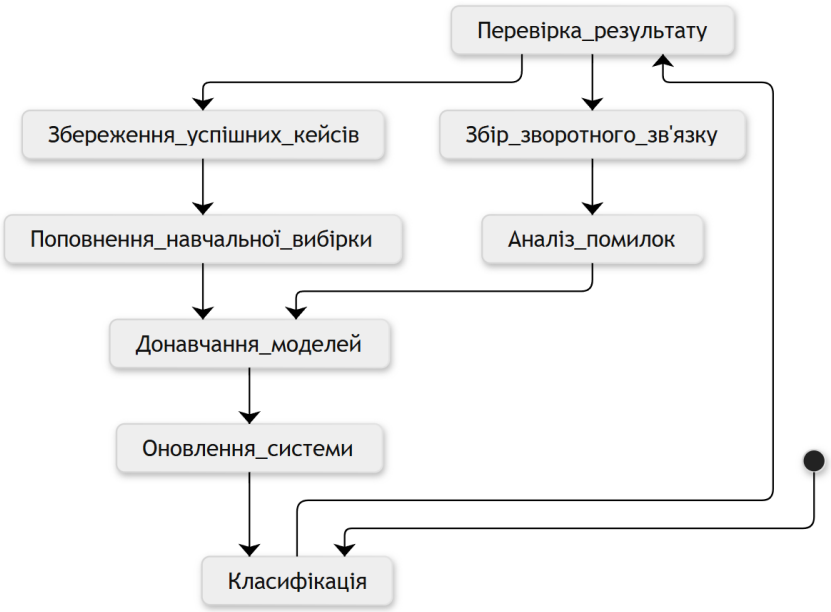


Рисунок 2.12. Цикл зворотного зв’язку

Особлива увага приділяється обробці нестандартних та складних випадків, для яких реалізовано механізм ескалації та залучення експертної оцінки. Система також включає механізми виявлення та обробки невизначеностей у класифікації, що дозволяє приймати зважені рішення навіть у складних випадках.

### 2.2. Методи попередньої обробки та векторизації текстових даних

Процес очистки та нормалізації текстових даних є фундаментальним етапом попередньої обробки, що забезпечує підвищення якості подальшої класифікації запитів. Система реалізує комплексний підхід до обробки текстових даних, який враховує специфіку природної мови та особливості користувацьких повідомлень у різних каналах комунікації. Рис. 2.13 демонструє процес очистки та нормалізації тексту.


Рисунок 2.13. Процес очистки та нормалізації тексту

Система реалізує багаторівневий процес обробки текстових даних, що включає послідовність спеціалізованих фільтрів та перетворень (рис. 2.14):

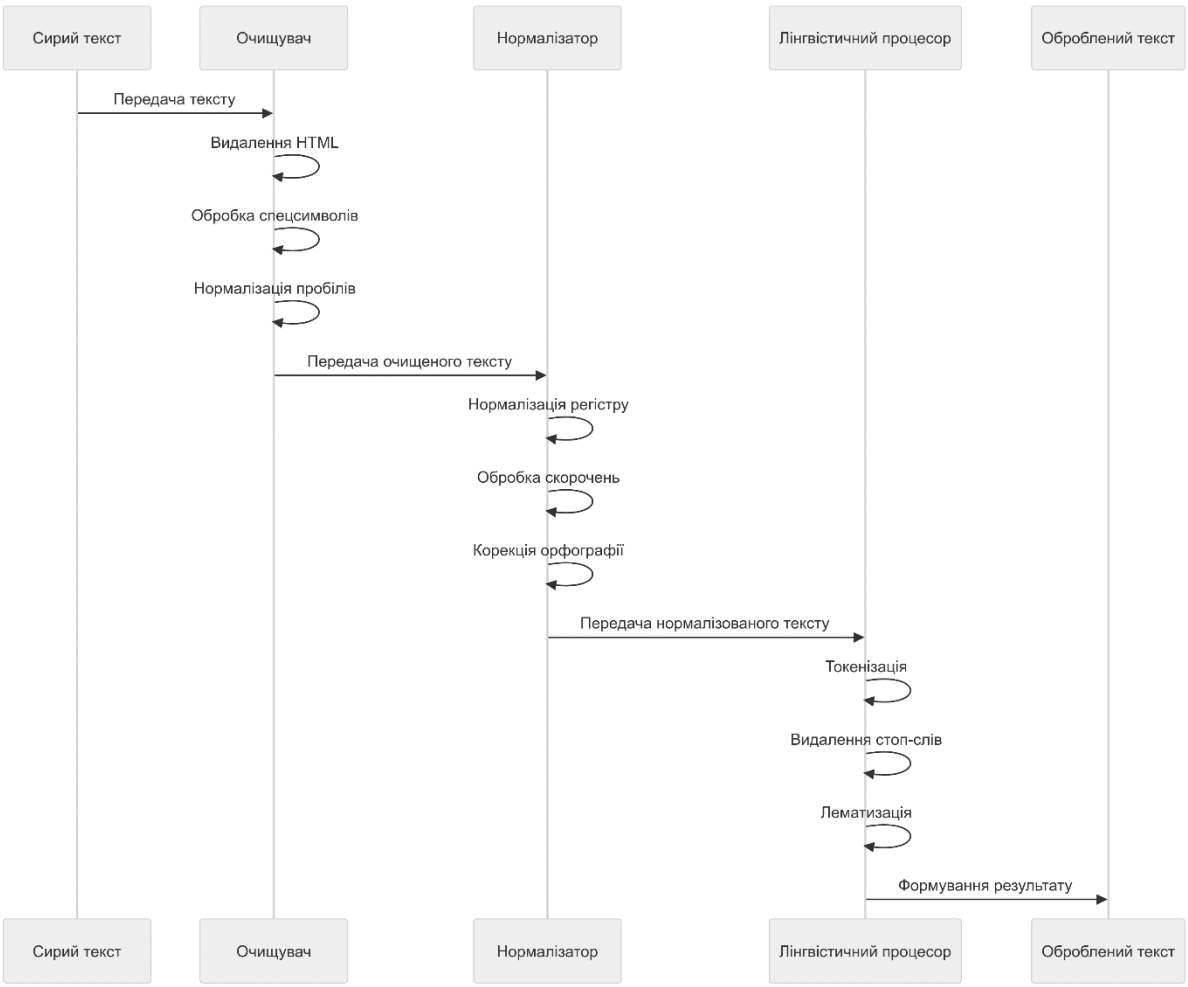


Рисунок 2.14. Послідовність обробки тексту

Система також включає механізми адаптивної обробки тексту, що дозволяють враховувати специфіку різних типів повідомлень (рис. 2.15):

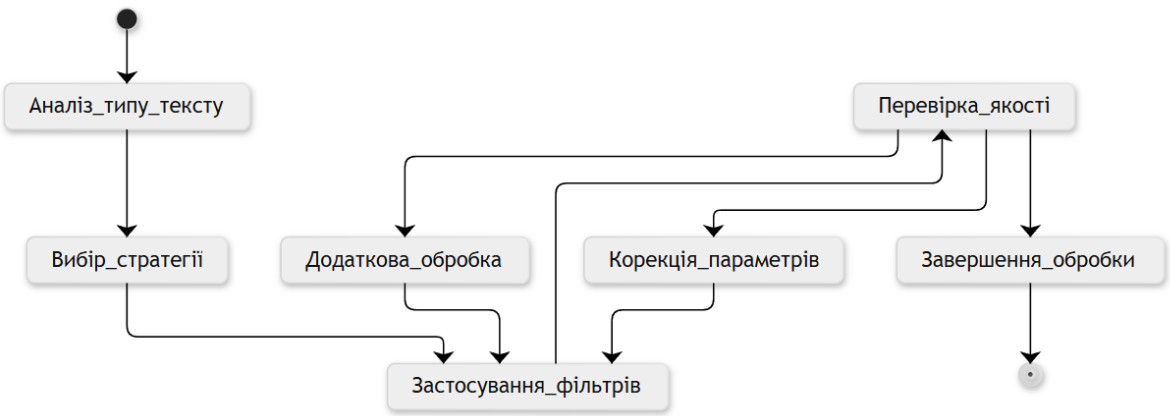


Рисунок 2.15. Адаптивна обробка тексту

Реалізований підхід до очистки та нормалізації тексту забезпечує високу якість підготовки даних для подальшої класифікації, при цьому зберігаючи семантичну значущість текстового повідомлення та враховуючи контекстуальні особливості клієнтських запитів.

Система реалізує комплексний підхід до векторизації текстових даних, використовуючи сучасні методи представлення слів у багатовимірному просторі. В основі реалізації лежить гібридний підхід, що поєднує різні техніки векторизації для досягнення оптимального балансу між якістю представлення семантики тексту та обчислювальною ефективністю. Рис. 2.17 демонструє процес очистки та нормалізації тексту.

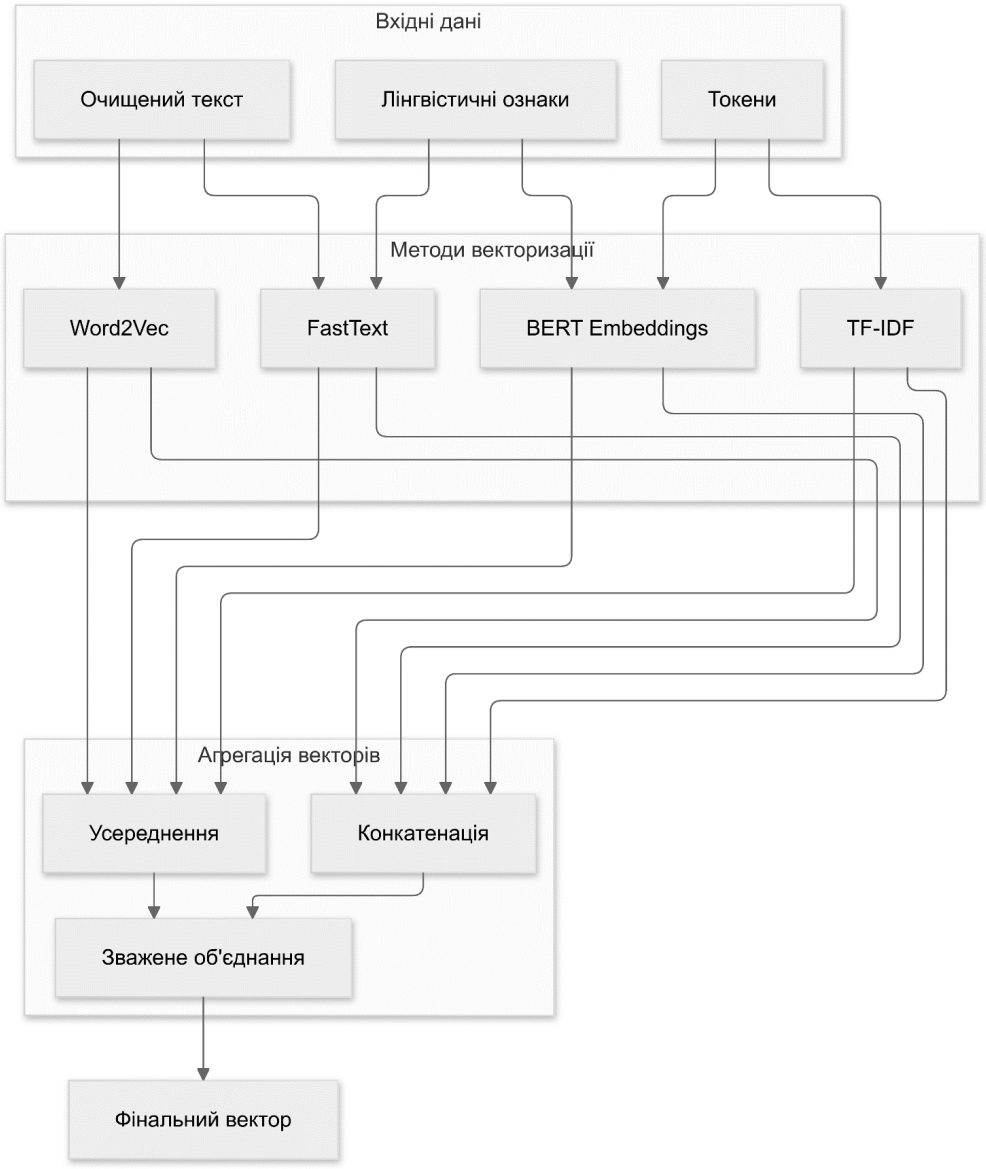


Рисунок 2.17. Архітектура векторного представлення

Процес формування векторного представлення реалізований як багатоетапна процедура, що враховує різні аспекти текстової інформації (рис. 2.18):

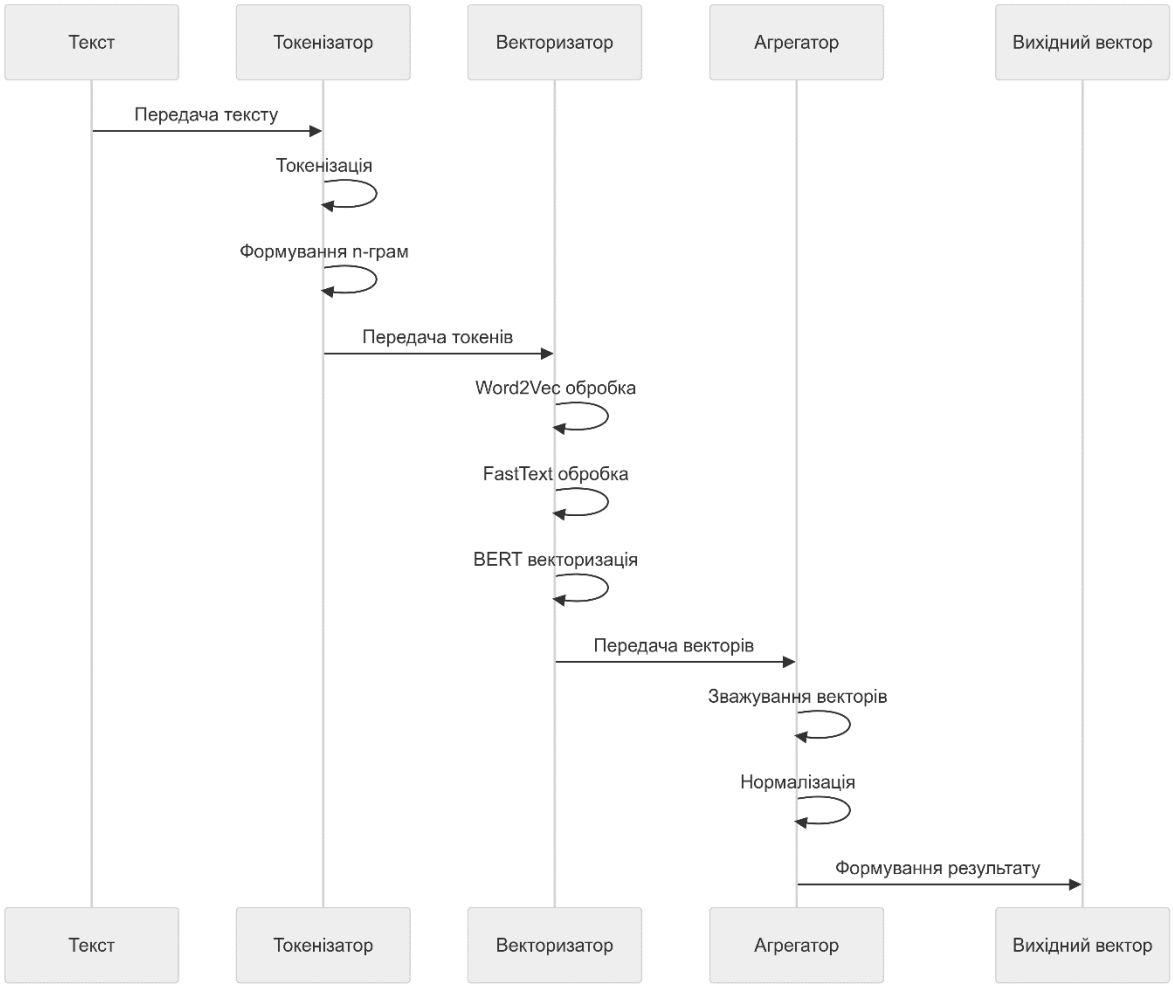


Рисунок 2.18. Процес форматування векторного представлення

Система підтримує динамічне оновлення векторних представлень на основі нових даних (рис. 2.19):


Рисунок 2.19. Процес оновлення векторних представлень

Для забезпечення оптимальної продуктивності реалізовано механізм кешування та оптимізації векторних представлень (рис. 2.20):

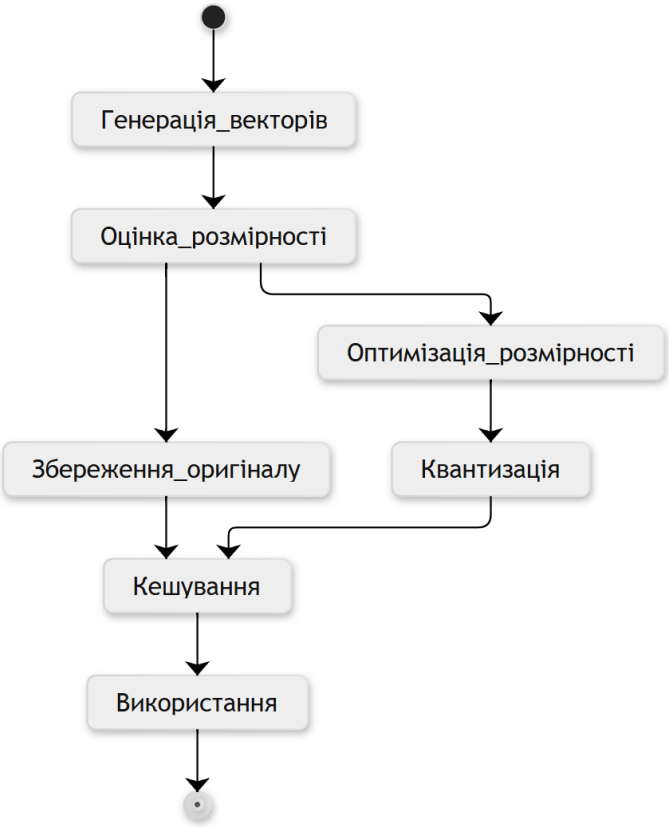


Рисунок 2.20. Система оптимізації векторів

Реалізована система векторного представлення слів забезпечує ефективну трансформацію текстових даних у формат, придатний для подальшої обробки алгоритмами машинного навчання, зберігаючи при цьому семантичні та синтаксичні особливості вхідного тексту.

### 2.3. Розробка алгоритму класифікації та маршрутизації запитів на основі обробки природної мови

Система реалізує багаторівневий алгоритм класифікації та маршрутизації запитів, що забезпечує точне визначення категорії запиту та його оптимальну маршрутизацію до відповідного обробника. Архітектура алгоритму побудована з урахуванням можливості паралельної обробки та динамічного масштабування системи. Рис. 2.21 демонструє архітектуру алгоритму класифікації.

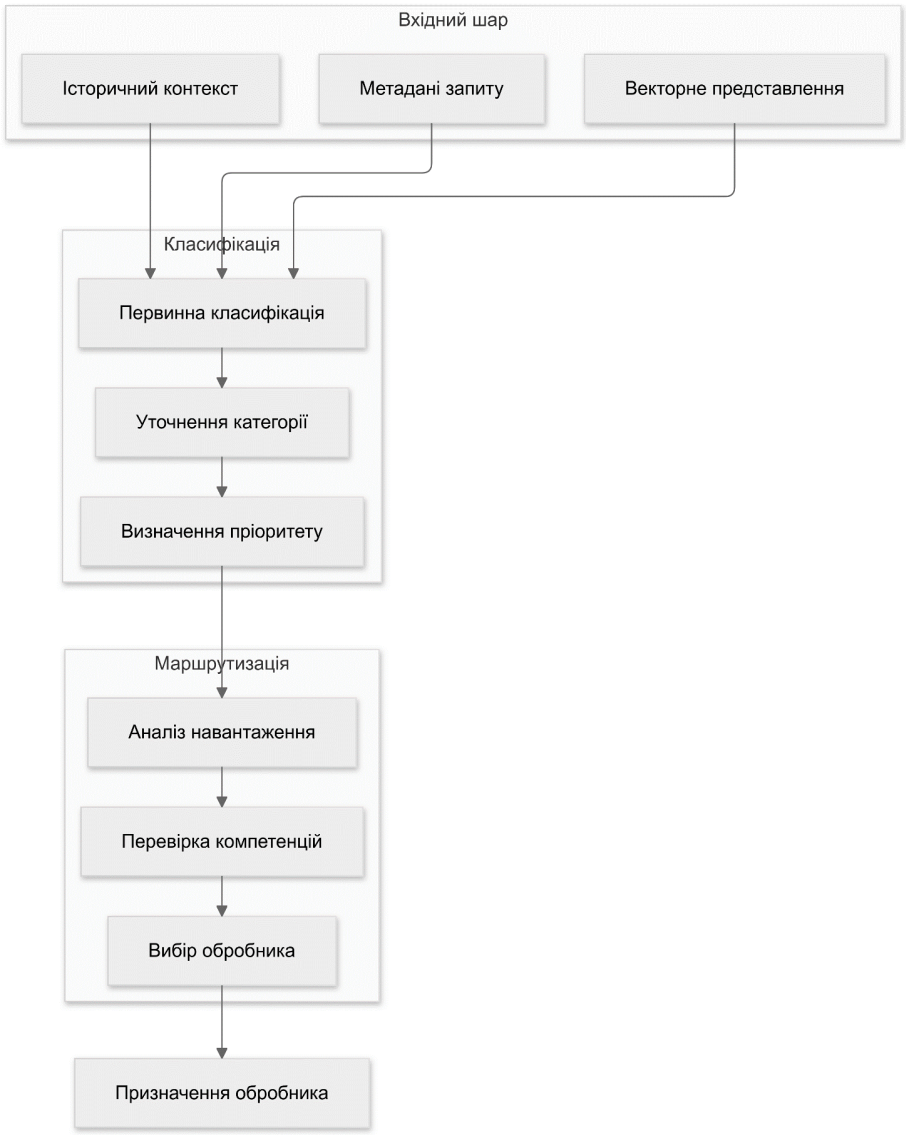


Рисунок 2.21. Архітектура алгоритму класифікації

Процес класифікації та маршрутизації реалізується як послідовність взаємопов’язаних етапів (рис. 2.22):

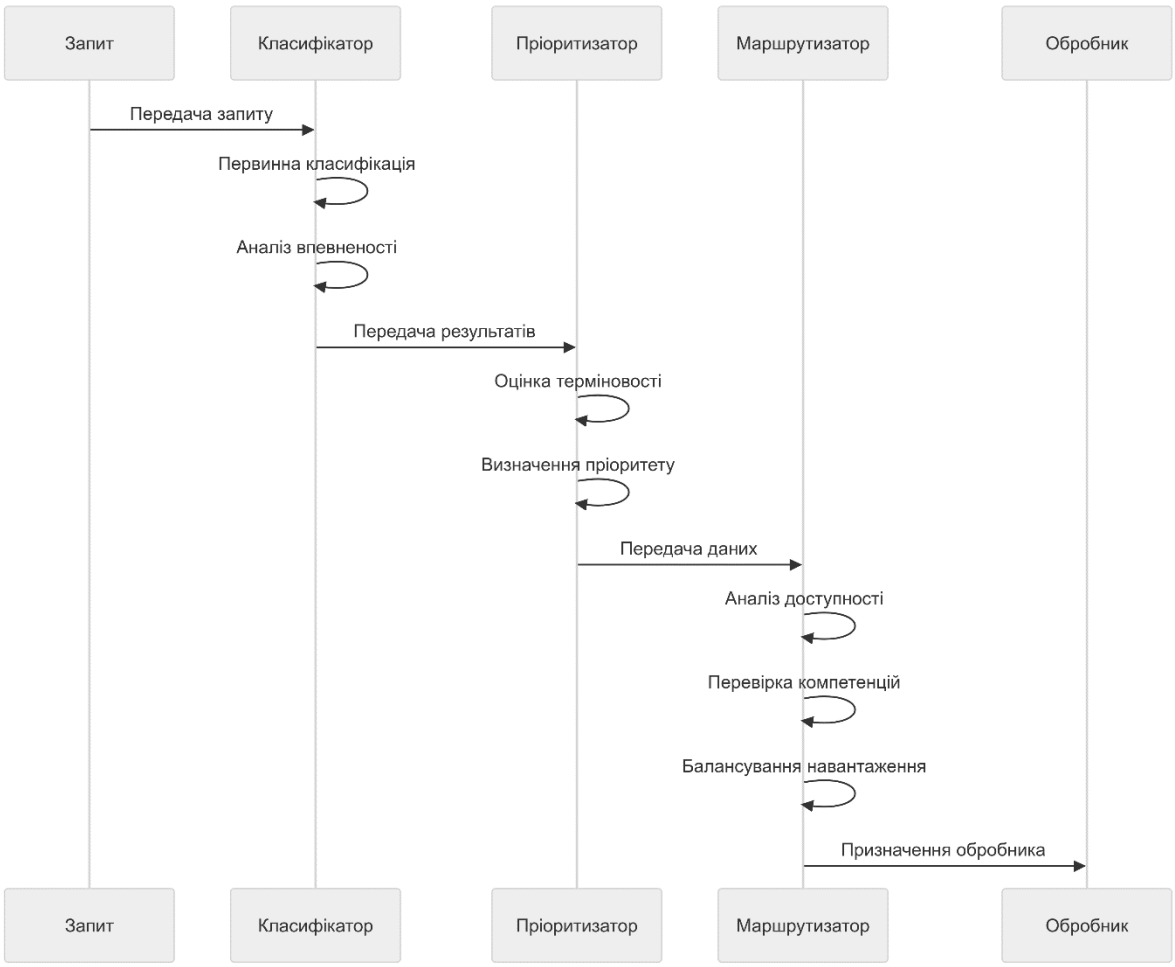


Рисунок 2.22. Потік обробки запиту

Система включає механізми обробки невизначених ситуацій та конфліктів (рис. 2.23):

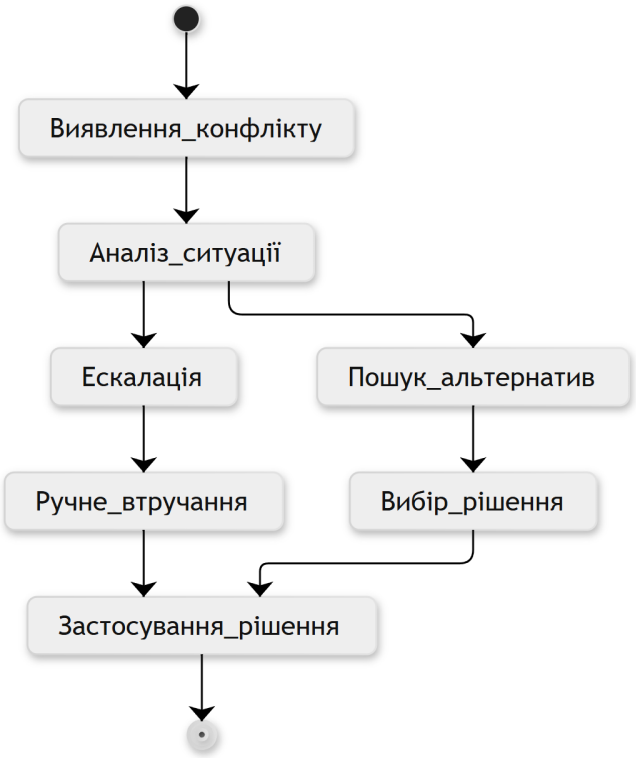


Рисунок 2.23. Механізм вирішення конфліктів

Для оптимізації процесу маршрутизації реалізовано систему динамічного розподілу навантаження (рис. 2.24):

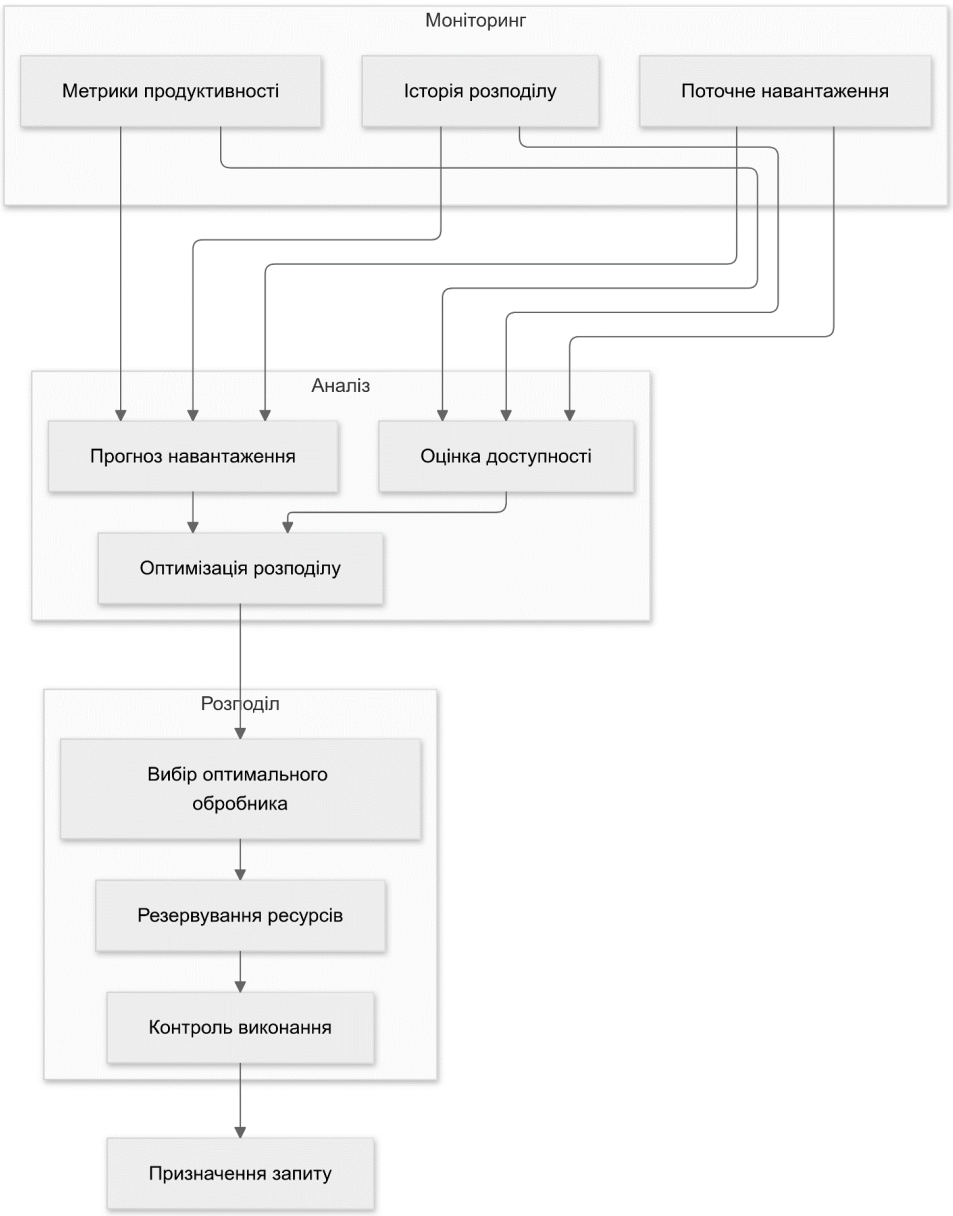


Рисунок 2.24. Система балансування навантаження

Система також включає механізми адаптивного навчання та оптимізації алгоритму (рис. 2.25):

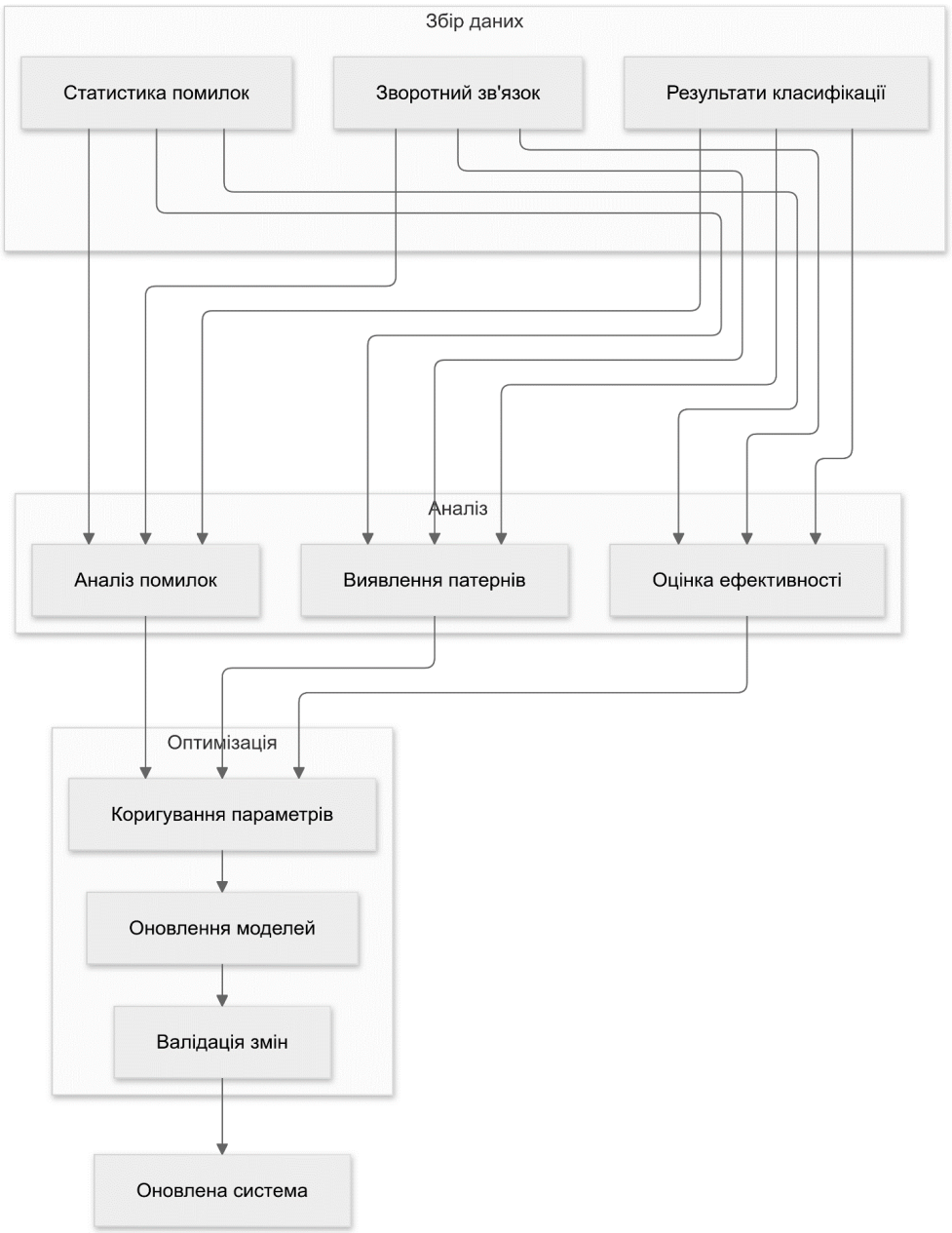


Рисунок 2.25. Система адаптивного навчання

Реалізований алгоритм забезпечує високу точність класифікації та ефективну маршрутизацію запитів, при цьому постійно адаптуючись до змін у структурі вхідних даних та умов роботи системи.

**Висновки до розділу 2**

Отже, в межах другого розділу було розроблено комплексну методологію створення автоматизованої системи класифікації та маршрутизації запитів на основі обробки природної мови. Архітектура системи базується на модульному

принципі та включає п’ять ключових компонентів: модуль прийому запитів, модуль попередньої обробки, модуль класифікації, модуль маршрутизації та модуль зворотного зв’язку. Особливістю архітектурного рішення є використання мікросервісного підходу, що забезпечує високу гнучкість та масштабованість системи.

Розроблено детальні методи попередньої обробки та векторизації текстових даних, що включають багаторівневу систему очистки, нормалізації та трансформації тексту. Впроваджено гібридний підхід до векторизації, який поєднує різні техніки для оптимального представлення семантики тексту при збереженні обчислювальної ефективності.

Створено комплексний алгоритм класифікації та маршрутизації запитів, що базується на ансамблі моделей машинного навчання та включає механізми обробки невизначених ситуацій. Реалізовано систему динамічного балансування навантаження та адаптивного навчання, що дозволяє системі постійно вдосконалюватися на основі нових даних.

Особливу увагу приділено забезпеченню надійності та відмовостійкості системи через впровадження механізмів буферизації вхідних даних, валідації та обробки помилок. Система підтримує роботу з різними каналами комунікації та забезпечує уніфіковану обробку різнотипних клієнтських запитів. Реалізовані механізми зворотного зв’язку дозволяють здійснювати постійний моніторинг якості роботи системи та її оптимізацію.

# РОЗДІЛ 3.
# ПРОГРАМНО-ТЕХНІЧНЕ РІШЕННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ КЛАСИФІКАЦІЇ ТА МАРШРУТИЗАЦІЇ ЗАПИТІВ НА ОСНОВІ ОБРОБКИ ПРИРОДНОЇ МОВИ

## 3.1. Вибір технологій та інструментів розробки

При розробці автоматизованої системи класифікації та маршрутизації запитів клієнтів ключовим фактором є вибір оптимального стеку технологій, що забезпечить ефективну реалізацію всіх запланованих функціональностей (рис. 3.1). Система базується на використанні сучасних інструментів та бібліотек мови Python, що надають широкі можливості для обробки природної мови та машинного навчання.

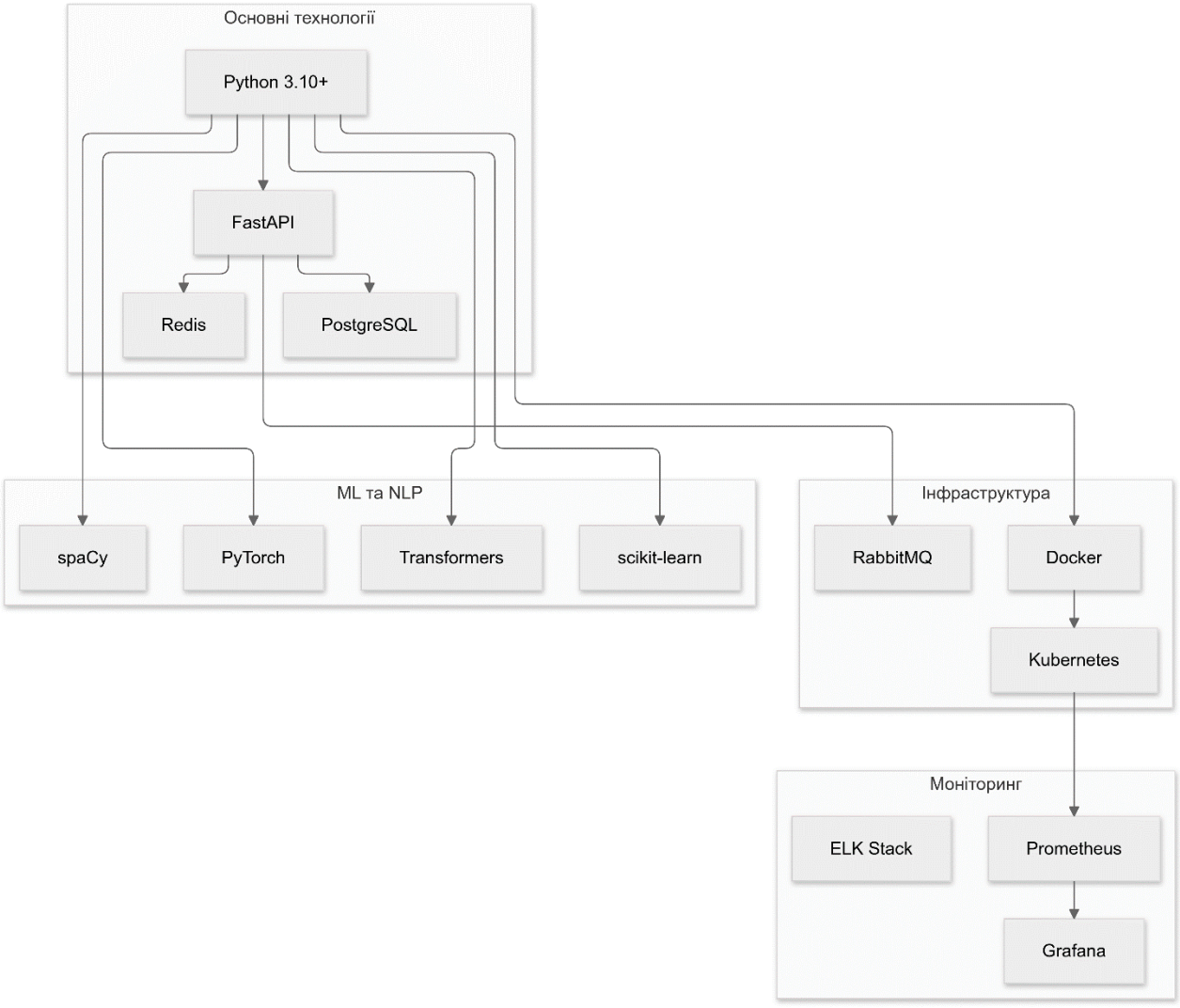


Рисунок 3.1. Стек технологій системи

Для розробки автоматизованої системи класифікації та маршрутизації запитів клієнтів було обрано мову програмування Python через її потужну екосистему для обробки природної мови та машинного навчання. Python версії 3.10 забезпечує всі необхідні можливості для реалізації поставлених задач та має відмінну підтримку асинхронного програмування.

В якості основного фреймворку для розробки веб-застосунку було обрано FastAPI, що надає можливість створення високопродуктивних API з автоматичною валідацією даних та генерацією документації. Наприклад, визначення моделей даних з автоматичною валідацією виглядає наступним чином:

```
class RequestMetadata(BaseModel):
    customer_id: str
    customer_type: CustomerType = CustomerType.standard
    repeat_issue: bool = False
    previous_issues: bool = False
```

Для обробки природної мови та реалізації класифікації використовується комбінація бібліотек. Зокрема, spaCy забезпечує базову обробку тексту, включаючи токенізацію та лематизацію:

```
nlp = spacy.load("en_core_web_sm")
doc = nlp("Текст для обробки")
tokens = [token.lemma_ for token in doc if not token.is_stop]
```

Для векторизації тексту використовується TF-IDF векторизатор з scikit-learn, що дозволяє перетворювати текстові дані у числові вектори:

```
vectorizer = TfidfVectorizer(max_features=1000)
X = vectorizer.fit_transform(texts).toarray()
```

Нейромережева модель реалізована з використанням PyTorch, що надає гнучкі можливості для створення та навчання глибоких нейронних мереж. Архітектура моделі визначається наступним чином:

```
class TextClassifier(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super().__init__()
        self.network = nn.Sequential(
            nn.Linear(input_size, hidden_size),
```

```
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(hidden_size, num_classes)
        )
```

Для зберігання проміжних даних та кешування використовується Redis, що забезпечує швидкий доступ до часто використовуваної інформації. Підключення до Redis реалізується наступним чином:

```
redis_client = redis.from_url(settings.REDIS_URL)
redis_client.setex(key, ttl, serialized_data)
```

Система моніторингу базується на використанні Prometheus для збору метрик та Grafana для їх візуалізації. Базові метрики визначаються через декоратори:

```
requests_total = Counter('requests_total', 'Total requests processed', ['category'])
processing_time = Histogram('request_processing_seconds', 'Request processing time')
```

Для забезпечення масштабованості та відмовостійкості система розгортається в контейнерах Docker з використанням оркестратора Kubernetes. Базовий Dockerfile виглядає наступним чином:

```
FROM python:3.10-slim
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Тестування системи реалізовано з використанням pytest, що дозволяє ефективно перевіряти як окремі компоненти, так і систему в цілому. Приклад тесту для класифікатора:

```
def test_classifier_prediction():
    classifier = TextClassifier(input_size=1000, hidden_size=256, num_classes=5)
    result = classifier.predict("test text")
    assert isinstance(result, tuple)
    assert len(result) == 2
```

Для розробки використовується інтегроване середовище PyCharm Professional, що надає розширені можливості для роботи з Python та веб-

розробки. Вибрані технології та інструменти забезпечують оптимальний баланс між продуктивністю, надійністю та зручністю розробки, дозволяючи створити масштабовану та підтримувану систему для вирішення поставлених задач.

Для реалізації системи було використано широкий спектр Python-бібліотек, які можна розділити на кілька функціональних груп:

Основні фреймворки та веб-сервер:

- FastAPI (0.115.5) – сучасний веб-фреймворк для створення API з автоматичною валідацією даних та документацією;
- Uvicorn (0.32.0) – легкий ASGI веб-сервер для Python;
- Starlette (0.41.2) – легкий ASGI фреймворк, що є основою для FastAPI.

Бібліотеки для обробки природної мови:

- spaCy (3.8.2) та en_core_web_sm – потужна бібліотека для NLP з попередньо навченою англійською моделлю;
- transformers (4.46.2) – бібліотека від Hugging Face для роботи з трансформерами;
- tokenizers (0.20.3) – швидка бібліотека для токенізації тексту.

Машинне навчання та обробка даних:

```
import torch  # Для глибокого навчання
import numpy as np  # Для числових обчислень
import pandas as pd  # Для обробки структурованих даних
from sklearn.feature_extraction.text import TfidfVectorizer  # Для векторизації тексту
```

Глибоке навчання:

- PyTorch (2.5.1) – фреймворк для створення та навчання нейронних мереж;
- scikit-learn (1.5.2) – бібліотека для класичного машинного навчання;
- numpy (2.0.2) – бібліотека для ефективної роботи з масивами;
- pandas (2.2.3) – бібліотека для аналізу та обробки даних.

Кешування та зберігання даних:

```
pythonCopyimport redis  # Для кешування
from prometheus_client import Counter, Histogram  # Для метрик
```

Моніторинг та логування:

- prometheus_client (0.21.0) – клієнт для збору метрик;
- rich (13.9.4) – бібліотека для покращеного форматування виводу.

Валідація даних:

```
from pydantic import BaseModel
```

- pydantic (2.9.2) – бібліотека для валідації даних та серіалізації.

Допоміжні бібліотеки:

- python-multipart (0.0.17) – для обробки multipart/form-data;
- python-dateutil (2.9.0) – для роботи з датами;
- PyYAML (6.0.2) – для роботи з YAML-конфігураціями.

Для розробки використовуються актуальні версії бібліотек, що забезпечує:

- Доступ до найновіших функцій та покращень;
- Виправлення відомих помилок та вразливостей;
- Оптимальну продуктивність.

Варто відзначити, що версії бібліотек зафіксовані в requirements.txt для забезпечення відтворюваності середовища розробки та розгортання. Приклад встановлення залежностей:

```
pip install -r requirements.txt
```

Така комбінація бібліотек забезпечує всі необхідні інструменти для розробки надійної та ефективної системи класифікації та маршрутизації запитів.

### 3.2. Розробка компонентів системи та їх інтеграція

Модуль обробки вхідних даних є фундаментальним компонентом системи, що відповідає за первинну обробку та підготовку клієнтських запитів для подальшої класифікації та маршрутизації. Реалізація даного модуля базується на комплексному підході до обробки природної мови з використанням сучасних методів та інструментів.

Архітектурно модуль спроектовано за принципами SOLID, що забезпечує високу модульність та можливість легкого розширення функціональності. Основним будівельним блоком є клас TextProcessor, який координує роботу різних компонентів обробки тексту.

Валідація вхідних даних реалізована з використанням бібліотеки Pydantic, що забезпечує строгу типізацію та автоматичну валідацію форматів даних. Визначено дві основні моделі даних: RequestMetadata для метаінформації про запит та Request для самого запиту. Це дозволяє на ранньому етапі відфільтрувати некоректні дані та забезпечити цілісність вхідної інформації.

```
class RequestMetadata(BaseModel):
    customer_id: str
    customer_type: CustomerType = CustomerType.standard
    repeat_issue: bool = False
    previous_issues: bool = False

class Request(BaseModel):
    text: str
    metadata: RequestMetadata
```

Процес очистки тексту реалізований у класі TextCleaner, який використовує комбінацію регулярних виразів та спеціалізованих алгоритмів обробки тексту. Клас забезпечує видалення небажаних елементів (URL, email-адреси, спеціальні символи) та нормалізацію тексту. Особлива увага приділена збереженню семантичної значущості тексту при очистці.

```
class TextCleaner:
    def __init__(self):
        self.nlp = spacy.load("en_core_web_sm")
        self.patterns = {
            'url': re.compile(r'https?://\S+|www\.\S+'),
            'email': re.compile(r'\S+@\S+'),
            'special_chars': re.compile(r'[^\w\s]')
        }
```

Нормалізація тексту включає декілька послідовних етапів:

1. Токенізація тексту з використанням spaCy;
2. Видалення стоп-слів на основі попередньо завантаженого словника;

3. Лематизація для приведення слів до їх базової форми;
4. Об’єднання нормалізованих токенів у фінальний текст.

Цей процес забезпечує стандартизацію вхідних даних та покращує якість подальшої класифікації.

```
def normalize(self, text: str) -> List[str]:
    doc = self.nlp(text)
    cleaned_text = self.clean(text)
    lemmatized_tokens = self.lemmatize(cleaned_text)
    return self.remove_stopwords(lemmatized_tokens)
```

Для оптимізації продуктивності впроваджено систему кешування з використанням декоратора @cached та TTLCache. Це дозволяє уникнути повторної обробки ідентичних запитів та значно знижує навантаження на систему при роботі з часто повторюваними запитами.

```
@cached(cache=TTLCache(maxsize=100, ttl=3600))
def process_text(self, text: str) -> str:
    cleaned_text = self.cleaner.clean(text)
    normalized_tokens = self.normalizer.normalize(cleaned_text)
    return " ".join(normalized_tokens)
```

Система обробки помилок реалізована на декількох рівнях:

- Рівень валідації вхідних даних;
- Рівень обробки тексту;
- Рівень інтеграції з іншими компонентами.

Кожен рівень має свої специфічні обробники помилок та механізми логування, що забезпечує детальну діагностику проблем.

```
try:
    processed_text = preprocessor.process(text)
    logger.info(f"Successfully processed text: {text[:50]}...")
except Exception as e:
    logger.error(f"Error processing text: {str(e)}")
    raise HTTPException(status_code=500, detail="Text processing error")
```

Механізм зворотного зв’язку реалізований через систему метрик та статистики, що дозволяє:

- Відслідковувати якість обробки запитів;

- Виявляти проблемні патерни в даних;
- Оптимізувати параметри обробки;
- Формувати звіти про продуктивність системи.

```
    def update_processing_stats(self, original_text: str, processed_text: str, success: bool):
        self.stats.update({
            'total_processed': self.stats.get('total_processed', 0) + 1,
            'successful': self.stats.get('successful', 0) + int(success)
        })
```

Інтеграція з іншими компонентами системи реалізована через REST API з використанням FastAPI та забезпечує:

- Асинхронну обробку запитів;
- Автоматичну валідацію даних;
- Автоматичну генерацію документації API;
- Простоту інтеграції з іншими сервісами.

```
@app.post("/process_request", response_model=Response)
async def process_request(request: Request):
    processed_text = text_processor.process(request.text)
    return await classifier.classify(processed_text, request.metadata)
```

Для забезпечення високої доступності та масштабованості модуль підтримує асинхронну обробку запитів та реалізує механізми балансування навантаження. Вся обробка відбувається в окремому потоці, що не блокує основний потік виконання програми.

Модуль також включає систему метрик для моніторингу продуктивності:

- Час обробки запитів
- Кількість успішно оброблених запитів
- Кількість помилок при обробці
- Розмір черги запитів

Такий підхід до реалізації модуля обробки вхідних даних забезпечує надійну основу для подальшої класифікації та маршрутизації запитів, гарантуючи якісну підготовку даних та стабільну роботу системи.

Розробка та реалізація модуля класифікації є ключовим елементом системи, що визначає ефективність і точність обробки клієнтських запитів. В основу реалізації покладено гібридний підхід, який поєднує методи глибокого навчання з традиційними алгоритмами машинного навчання, що дозволяє досягти оптимального балансу між точністю класифікації та обчислювальною ефективністю.

Архітектура класифікатора базується на багатошаровій нейронній мережі, реалізованій з використанням фреймворку PyTorch. Основний компонент - клас TextClassifier, який інкапсулює логіку нейромережевої обробки тексту. Мережа складається з послідовності лінійних шарів з активаційними функціями ReLU та механізмом регуляризації Dropout для запобігання перенавчанню:

```
class TextClassifier(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(hidden_size, num_classes)
        )
```

Особлива увага в реалізації приділяється процесу векторизації тексту. Для отримання якісного векторного представлення використовується комбінований підхід, що поєднує статистичні методи (TF-IDF) із сучасними методами контекстуалізованих ембедінгів на основі BERT. Така комбінація дозволяє враховувати як частотні характеристики тексту, так і семантичні зв’язки між словами:

```
class TextVectorizer:
    def __init__(self):
        self.tfidf = TfidfVectorizer(max_features=1000)
        self.tokenizer = AutoTokenizer.from_pretrained('distilbert-base-uncased')
        self.model = AutoModel.from_pretrained('distilbert-base-uncased')
```

Важливим аспектом роботи класифікатора є механізм оцінки впевненості в прийнятих рішеннях. Система використовує softmax-нормалізацію вихідного шару нейронної мережі для отримання розподілу ймовірностей за категоріями. При цьому реалізовано механізм порогової фільтрації, коли запити з низькою впевненістю класифікації позначаються як невизначені та направляються на додаткову обробку:

```
def predict(self, text: str) -> Tuple[str, float]:
    features = self.prepare_features(text)
    outputs = self.model(features)
    probs = torch.softmax(outputs, dim=1)
    predicted_class = torch.argmax(outputs, dim=1).item()
    confidence = probs[0][predicted_class].item()
```

Суттєвою інновацією в реалізації є система онлайн-навчання, яка дозволяє моделі адаптуватися до нових патернів у даних без необхідності повного перенавчання. Механізм зворотного зв’язку дозволяє корегувати ваги моделі на основі отриманої інформації про правильність класифікації:

```
def update_model(self, text: str, true_label: str):
    features = self.prepare_features(text)
    loss = self.criterion(self.model(features), true_label)
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()
```

Система оптимізації продуктивності реалізована через механізм батчової обробки запитів та інтелектуального кешування. Використання асинхронної обробки дозволяє ефективно розподіляти навантаження між обчислювальними ресурсами, а кешування проміжних результатів значно знижує час відгуку системи для подібних запитів.

Моніторинг роботи класифікатора здійснюється через комплексну систему метрик, що включає показники точності класифікації, часу обробки запитів та розподілу впевненості по категоріях. Система логування забезпечує детальне відстеження кожного етапу обробки запитів, що дозволяє проводити глибокий аналіз роботи класифікатора та виявляти потенційні проблеми.

Архітектурно система підтримує можливість «гарячого» оновлення моделі без переривання роботи сервісу. Це досягається завдяки механізму паралельного навчання нової версії моделі та плавного переключення на неї після валідації якості. Також передбачена можливість швидкого відкату до попередньої версії у разі виникнення проблем.

Взаємодія з іншими компонентами системи реалізована через стандартизований API, що забезпечує високу модульність та незалежність компонентів. Такий підхід спрощує процес тестування та дозволяє гнучко масштабувати систему відповідно до зростаючих потреб.

Система маршрутизації представляє собою критично важливий компонент, що відповідає за оптимальний розподіл класифікованих запитів між обробниками. Реалізація базується на багатофакторному аналізі з урахуванням категорії запиту, пріоритету, характеристик клієнта та поточного навантаження системи.

Основним компонентом системи маршрутизації є клас Router, який інкапсулює всю логіку розподілу запитів та управління навантаженням:

```
class Router:
    def __init__(self):
        self.handlers_info = {
            "tec_001": {
                "name": "Технічний спеціаліст 1",
                "skills": ["networking", "hardware"],
                "capacity": 10,
                "current_load": 0
            }
        }
```

Система пріоритизації запитів реалізована через комплексний механізм розрахунку ваги запиту на основі множини факторів. Кожен фактор має свій ваговий коефіцієнт, що дозволяє гнучко налаштовувати систему під конкретні бізнес-вимоги:

```
    def _calculate_priority(self, category: str, confidence: float,
metadata: dict) -> int:
        base_priority = 3.0
```

```
        category_multiplier = self.category_weights.get(category, 1.0)
        customer_multiplier = self.customer_type_weights.get(metadata.customer_type, 1.0)
        final_priority = base_priority * category_multiplier * customer_multiplier
```

Механізм балансування навантаження враховує поточну завантаженість кожного обробника та його спеціалізацію. Реалізовано алгоритм динамічного розподілу, який автоматично адаптується до змін у навантаженні системи:

```
    def route(self, category: str, confidence: float, metadata: dict) -> tuple:
        available_handlers = self.category_handlers.get(category, ["gen_001"])
        selected_handler = min(
            available_handlers,
            key=lambda h: self.handlers_info[h]["current_load"] / self.handlers_info[h]["capacity"]
        )
        handler_id = f"handler_{selected_handler}"
```

Важливим аспектом реалізації є система моніторингу поточного стану маршрутизації, яка дозволяє відслідковувати ефективність розподілу запитів та виявляти потенційні проблеми:

```
    def get_routing_stats(self) -> dict:
        return {
            "handlers": self.handlers_info,
            "category_mapping": self.category_handlers,
            "total_capacity": sum(h["capacity"] for h in self.handlers_info.values()),
            "total_load": sum(h["current_load"] for h in self.handlers_info.values())
        }
```

Для забезпечення відмовостійкості система реалізує механізм автоматичного перерозподілу запитів у випадку недоступності окремих обробників. При цьому враховується історія взаємодій з клієнтом для забезпечення послідовності обслуговування.

Реалізована система черг дозволяє ефективно обробляти пікові навантаження без втрати якості обслуговування. Черги організовані за

пріоритетами, що гарантує першочергову обробку критичних запитів. При цьому використовується механізм «справедливого» планування для запобігання проблемі «голодування» запитів з низьким пріоритетом.

В рамках системи маршрутизації реалізовано механізм зворотного зв’язку, який дозволяє оцінювати ефективність прийнятих рішень та автоматично корегувати параметри маршрутизації. Збирається детальна статистика по кожному обробнику, включаючи час обробки запитів, кількість успішних резолюцій та рівень задоволеності клієнтів.

Важливим аспектом реалізації є підтримка масштабування системи. Архітектура дозволяє динамічно додавати нових обробників та змінювати їх характеристики без необхідності перезапуску системи. При цьому всі зміни конфігурації відбуваються атомарно, що гарантує цілісність даних.

Система маршрутизації також включає механізми обробки виняткових ситуацій, таких як перевищення допустимого часу очікування або недоступність усіх профільних обробників. У таких випадках запит може бути перенаправлений до резервних обробників або переведений у режим ескалації.

Взаємодія з іншими компонентами системи відбувається через чітко визначений інтерфейс, що забезпечує незалежність реалізацій та можливість модульного тестування. Вся взаємодія логується для подальшого аналізу та оптимізації роботи системи.

Користувацький інтерфейс системи реалізований як веб-застосунок з використанням сучасних технологій веб-розробки. Основний фокус при розробці інтерфейсу був зроблений на забезпеченні інтуїтивно зрозумілої взаємодії користувача з системою при збереженні всіх необхідних функціональних можливостей.

Базова структура інтерфейсу реалізована з використанням HTML та фреймворку Tailwind CSS, що забезпечує адаптивний дизайн та сучасний візуальний стиль:

```
<div class="container mx-auto px-4 py-8 max-w-4xl">
    <h1 class="text-3xl font-bold text-center mb-8">
```

```
            Customer Request Classification System
        </h1>
        <div class="mb-6 flex justify-center space-x-2">
            <button
                onclick="switchTab('test')"
                class="px-6 py-2 font-medium rounded-lg bg-blue-500 text-
white">
                Класифікація запитів
            </button>
        </div>
    </div>
```

Система введення даних реалізована через інтуїтивну форму з валідацією на стороні клієнта:

```
    document.getElementById('requestForm').addEventListener('submit',
async (e) => {
        e.preventDefault();
        const requestData = {
            text: document.getElementById('requestText').value,
            metadata: {
                customer_type:
document.getElementById('customerType').value,
                repeat_issue:  document.getElementById('repeatIssue').value
=== 'true'
            }
        };
```

Для відображення результатів класифікації та маршрутизації використовується динамічне оновлення контенту з анімаціями для покращення користувацького досвіду:

```
    async function updateResults(data) {
        document.getElementById('resultCategory').textContent            =
data.category;
        document.getElementById('resultConfidence').textContent =
            (data.confidence * 100).toFixed(1) + '%';
        document.getElementById('resultHandler').textContent             =
data.handler_id;
    }
```

Інтерфейс включає інтерактивну візуалізацію системи маршрутизації, яка відображає поточний стан обробників та статистику роботи системи в

реальному часі. Реалізовано механізм періодичного оновлення даних без перезавантаження сторінки:

```
function updateRoutingVisualization() {
    const statsDiv = document.getElementById('routingStats');
    fetch('/routing_stats')
        .then(response => response.json())
        .then(data => updateStatsDisplay(data))
        .catch(error => console.error('Error:', error));
}
```

Особливу увагу приділено системі сповіщень, яка інформує користувача про стан обробки запиту та можливі помилки. Реалізовано різні типи сповіщень залежно від важливості повідомлення та контексту:

```
function showNotification(message, type = 'info') {
    const notification = document.createElement('div');
    notification.className = `notification ${type} p-4 rounded-lg`;
    notification.textContent = message;
    document.body.appendChild(notification);
}
```

Інтерфейс системи навчання моделі надає можливість завантаження тренувальних даних та відслідковування процесу навчання в реальному часі. Реалізовано візуалізацію прогресу навчання та основних метрик:

```
function updateTrainingProgress(progress) {
    const progressBar = document.getElementById('progressBar');
    progressBar.style.width = `${progress}%`;
    document.getElementById('progressText').textContent =
        `Навчання: ${progress}%`;
}
```

Для адміністративних функцій розроблено окремий інтерфейс, що дозволяє керувати налаштуваннями системи, переглядати статистику та здійснювати моніторинг роботи. Цей інтерфейс має додаткові рівні авторизації та розширені можливості конфігурації.

Значна увага приділена обробці помилок та інформуванню користувача про нестандартні ситуації. Кожна помилка супроводжується зрозумілим описом та рекомендаціями щодо подальших дій. Система також зберігає історію помилок для подальшого аналізу.

Для забезпечення високої доступності інтерфейсу реалізовано механізми кешування на стороні клієнта та оптимізацію завантаження ресурсів. Усі критичні операції супроводжуються індикаторами завантаження та відповідними статусами виконання.

Інтерфейс розроблено з урахуванням принципів доступності (accessibility) та підтримує роботу з клавіатурою та програмами читання з екрану. Також забезпечено коректне відображення на різних пристроях та в різних браузерах.

Інтеграція компонентів системи є критично важливим етапом розробки, що забезпечує ефективну взаємодію між усіма модулями та гарантує стабільну роботу системи в цілому. Процес інтеграції базується на принципах слабкого зв'язування (loose coupling) та високої згуртованості (high cohesion).

Центральним елементом інтеграції виступає основний клас застосунку, який координує роботу всіх компонентів:

```
app = FastAPI(
    title="Customer Request Classification System",
    description="Система класифікації та маршрутизації запитів клієнтів",
    version="1.0.0"
)

# Ініціалізація основних компонентів
preprocessor = TextPreprocessor()
classifier = Classifier()
router = Router()
```

Взаємодія між компонентами реалізована через систему асинхронних обробників подій, що забезпечує неблокуючу обробку запитів:

```
@app.post("/process_request", response_model=Response)
async def process_request(request: Request):
    start_time = datetime.now()
    processed_text = await preprocessor.process(request.text)
    category, confidence = await classifier.classify(processed_text)
    handler_id, priority = await router.route(category, confidence, request.metadata)
```

Система подій реалізована з використанням механізму асинхронних черг, що дозволяє ефективно обробляти паралельні запити:

```
class EventBus:
    def __init__(self):
        self.handlers = defaultdict(list)
        self.queue = asyncio.Queue()

    async def publish(self, event_type: str, data: dict):
        await self.queue.put((event_type, data))
        await self.process_events()
```

Для забезпечення надійності комунікації між компонентами впроваджено систему retry-механізмів та обробки помилок:

```
async def retry_operation(operation: Callable, max_attempts: int = 3):
    for attempt in range(max_attempts):
        try:
            return await operation()
        except Exception as e:
            if attempt == max_attempts - 1:
                raise
            await asyncio.sleep(2 ** attempt)
```

Важливим аспектом інтеграції є система кешування між компонентами, яка оптимізує обмін даними та зменшує навантаження на систему. Реалізовано багаторівневе кешування з використанням Redis:

```
class CacheManager:
    def __init__(self, redis_url: str):
        self.redis_client = redis.from_url(redis_url)
        self.local_cache = TTLCache(maxsize=1000, ttl=300)

    async def get_or_set(self, key: str,
                         operation: Callable) -> Any:
        if value := self.local_cache.get(key):
            return value
```

Для моніторингу стану системи реалізовано механізм збору та агрегації метрик з усіх компонентів:

```
class MetricsCollector:
    def __init__(self):
        self.processing_time = Histogram(
```

```
            'request_processing_seconds',
            'Time spent processing request',
            ['component']
        )
```

Система конфігурації забезпечує централізоване управління параметрами всіх компонентів та їх узгоджену роботу. Конфігурація може бути змінена без перезапуску системи:

```
class DynamicConfig:
    def __init__(self):
        self._config = {}
        self._watchers = []

    async def update(self, new_config: dict):
        self._config.update(new_config)
        await self._notify_watchers()
```

Для забезпечення цілісності даних при взаємодії компонентів впроваджено систему транзакцій:

```
async def process_with_transaction(operations: List[Callable]):
    async with transaction_manager() as tm:
        try:
            results = [await op() for op in operations]
            await tm.commit()
            return results
        except Exception:
            await tm.rollback()
            raise
```

Система логування забезпечує відстеження взаємодії між компонентами та спрощує діагностику проблем:

```
class Logger:
    def __init__(self, component_name: str):
        self.component = component_name
        self.logger = logging.getLogger(component_name)

    async def log_interaction(self,
                              source: str,
                              target: str,
                              status: str):
        self.logger.info(f"{source} -> {target}: {status}")
```

Окрема увага приділена механізмам відновлення після збоїв, які забезпечують збереження узгодженості даних між компонентами та можливість відновлення роботи системи після аварійних ситуацій. Реалізовано алгоритми узгодження стану компонентів та відновлення втрачених даних.

Система також включає механізми версіонування API між компонентами, що забезпечує можливість поступового оновлення окремих частин системи без порушення загальної функціональності. Це особливо важливо при впровадженні нових версій компонентів у production-середовищі.

Така інтеграція компонентів забезпечує надійну та ефективну роботу системи, дозволяючи гнучко масштабувати її відповідно до зростаючих потреб та адаптувати до нових вимог.

### 3.3. Тестування та оцінка ефективності системи

Для роботи системи було розроблено набір даних, що налічує 733 записи та поділяється на 4 категорії: billing (запити пов’язані зі сплатою та грошовими транзакціями), product_inquiry (запити пов’язані з наданням послуг), technical_support (запити на технічну підтримку), complaint (запити-скарги). Також існує категорія «Інше» для випадків, коли класифікувати один з чотирьох класів не вдалося, або впевненість в цій класифікації менше 60%.

Для початку запустимо розроблений сервер FastApi (рис. 3.2).


Рисунок 3.2. Запуск локального серверу

Перед користувачем з’являється головна сторінка веб-додатку з можливістю класифікувати запит (рис. 3.3), виконати навчання моделі (рис. 3.4) та переглянути поточну маршрутизацію (рис. 3.5 – рис. 3.6).

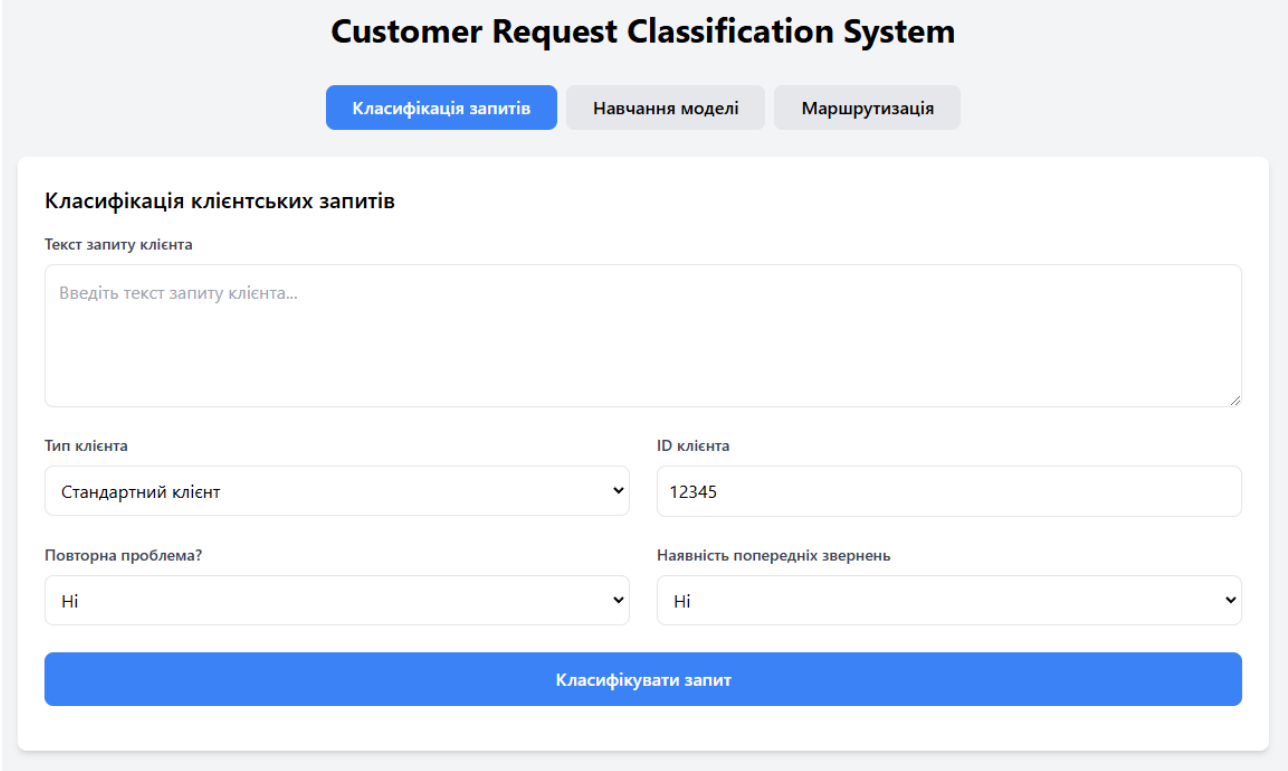


Рисунок 3.3. Вигляд користувацького інтерфейсу «Класифікація клієнтських запитів»

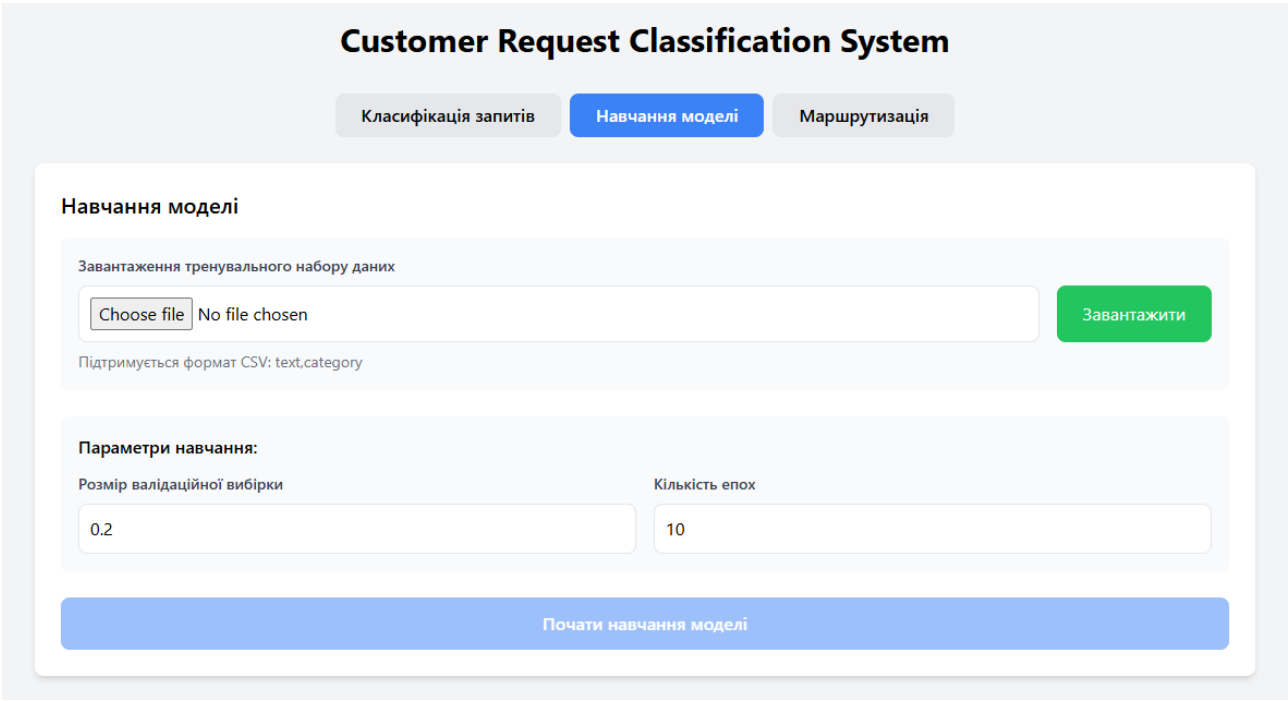


Рисунок 3.4. Вигляд користувацького інтерфейсу «Навчання моделі»

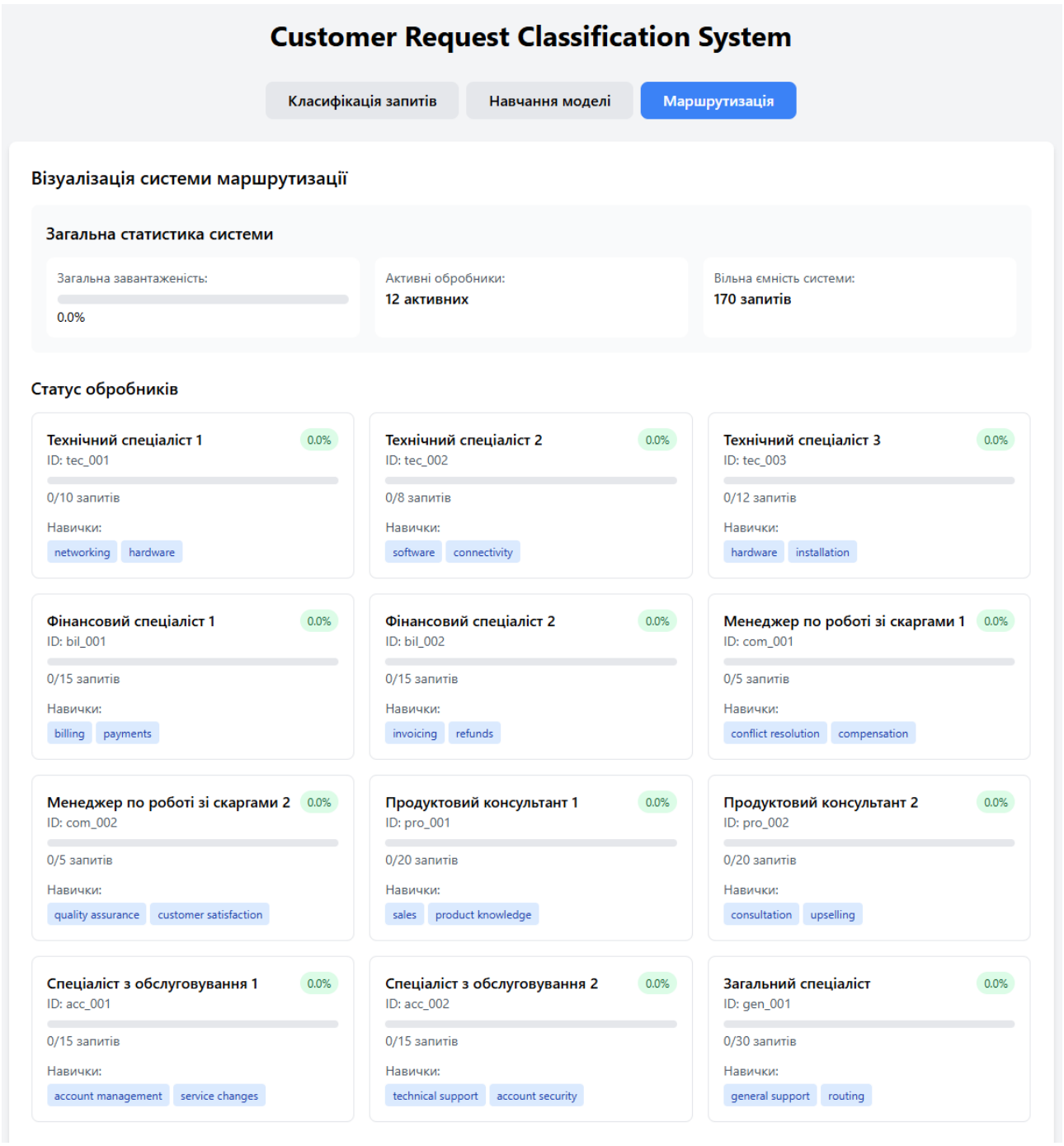


Рисунок 3.5. Вигляд користувацького інтерфейсу «Маршрутизація» (частина 1)

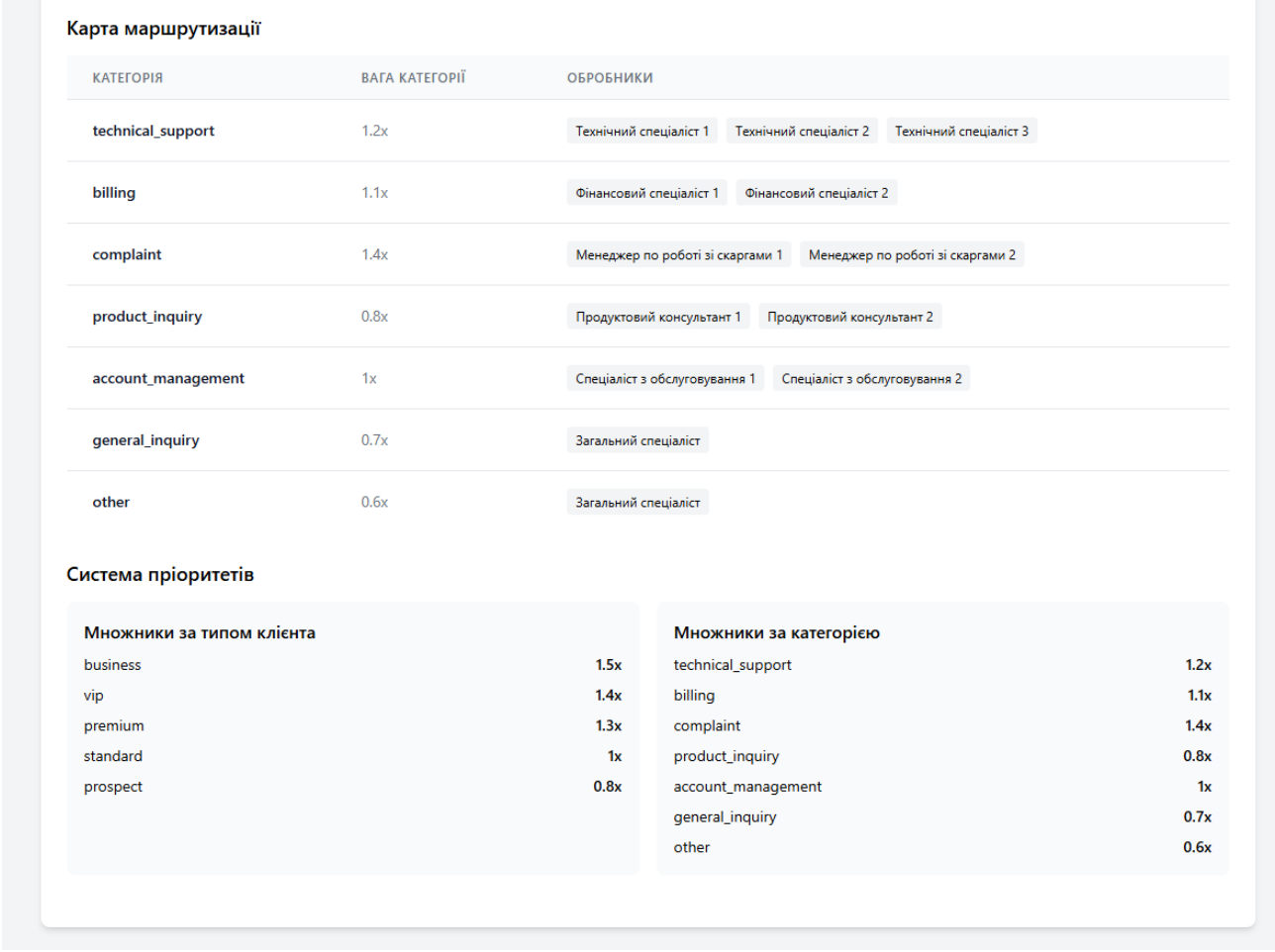


Рисунок 3.6. Вигляд користувацького інтерфейсу «Маршрутизація» (частина 2)

Виконаємо початкове навчання моделі на раніше згаданому наборі даних (рис. 3.7). Система дозволяє завантажити файл формату CSV (через POST запит /upload_dataset) та класифікує базову інформацію про: кількість записів, кількість категорій, кількість записів за кожною категорією. А також надає можливість встановити деякі параметри навчання, а саме: розмір валідаційної вибірки та кількість епох. Після натискання на кнопку «Почати навчання» починається процедура навчання, яка відображається користувачу в інтерфейсі та логується на сервері (рис. 3.8). Після завершення навчання, користувач отримує результат навчання, який включає: точність, F1-Score, час навчання та матрицю помилок.

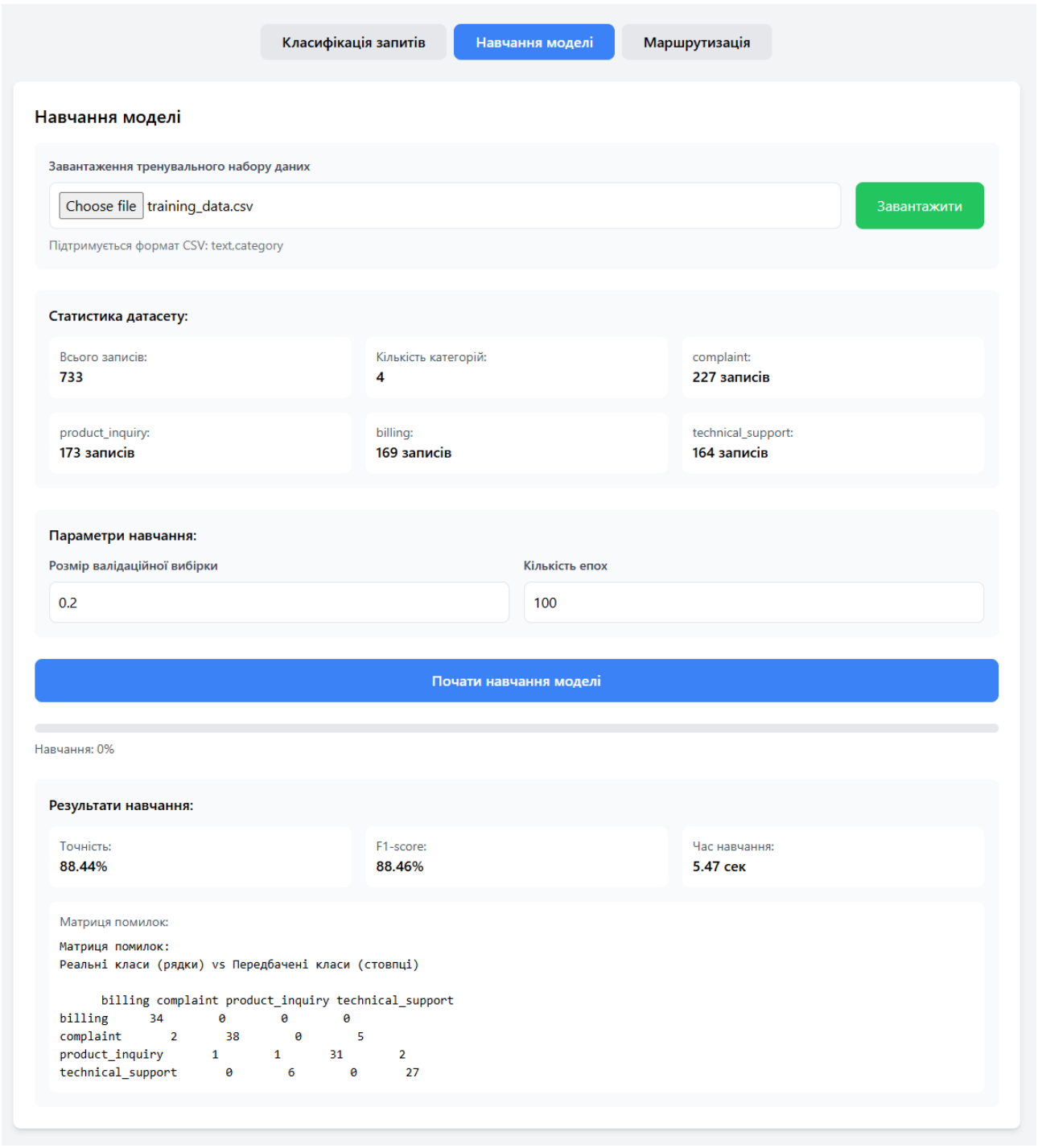


Рисунок 3.7. Процедура навчання моделі для користувачів


Рисунок 3.8. Процедура навчання моделі на сервері

Після успішного навчання в межах логів з’являється подібний рядок, що містить усю інформацію, що передається до веб-додатку:

`INFO:__main__:Model trained successfully: {'accuracy': 0.8843537414965986, 'f1_score': 0.8846202931010283, …}`

Спробуємо виконати клієнтський запит «Вітаю. Підкажіть вартість тарифів, які у Вас є» (рис. 3.9). Як можна побачити запит отримав класифікацію «product_inquiry» з пріоритетом 2 (Нижче середнього) та впевненістю 100%. Також був призначений відповідний обробник «handler_pro_001» та вказано час обробки «0.005 сек». Що загалом вказує на швидкість та коректність реагування у випадку запиту про послуги компанії. Детальніша інформація в межах логів також відображається на сервері (рис. 3.10)

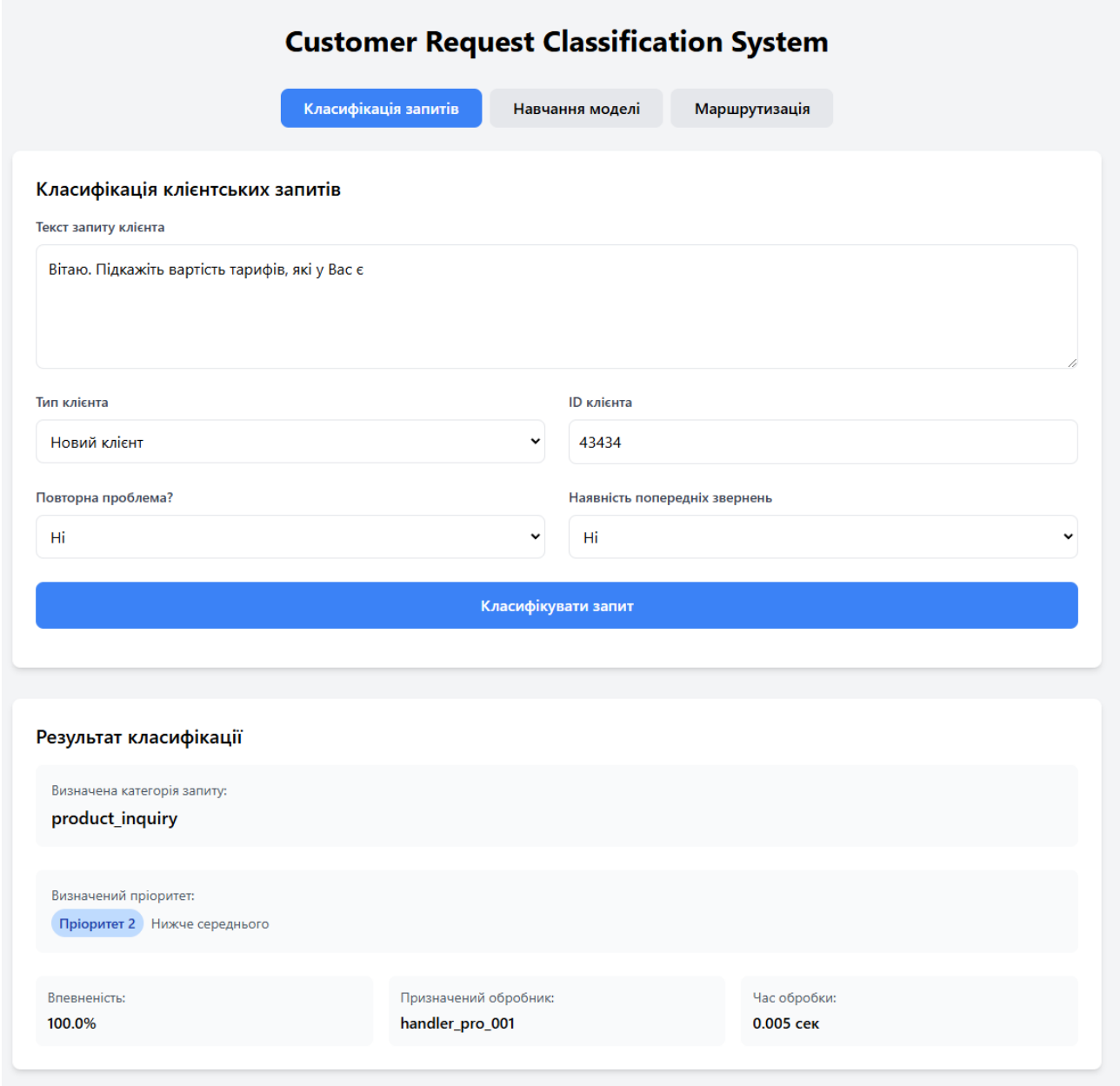


Рисунок 3.9. Процедура класифікації клієнтських запитів для користувачів

```
INFO:     127.0.0.1:10118 - "POST /train_model HTTP/1.1" 200 OK
INFO:__main__:Received request: Вітаю. Підкажіть вартість тарифів, які у Вас є...
INFO:__main__:Request classified as product_inquiry with confidence 0.999651312828064
INFO:__main__:Top categories with probabilities:
INFO:__main__:  product_inquiry: 0.9997
INFO:__main__:  billing: 0.0003
INFO:__main__:  complaint: 0.0001
INFO:routing:Routing request: category=product_inquiry, confidence=0.999651312828064, metadata=customer_id='43434'
INFO:routing:Routed to handler=handler_pro_001 with priority=2
INFO:__main__:Request routed to handler handler_pro_001 with priority 2
```

Рисунок 3.10. Процедура класифікації клієнтських запитів на сервері

Задамо наступний запит «Ви взагалі в своєму розумі? Що це відбувається» (рис. 3.11 – рис. 3.12). В межах даного запиту вже важче вловлюється контекст, проте модель коректно надала категорію «complaint» з пріоритетом 4 (Високий пріоритет) та впевненістю 99.5% для преміум клієнти. Також був призначений обробник «handler_com_001», а швидкість обробки становила «0.004 сек».

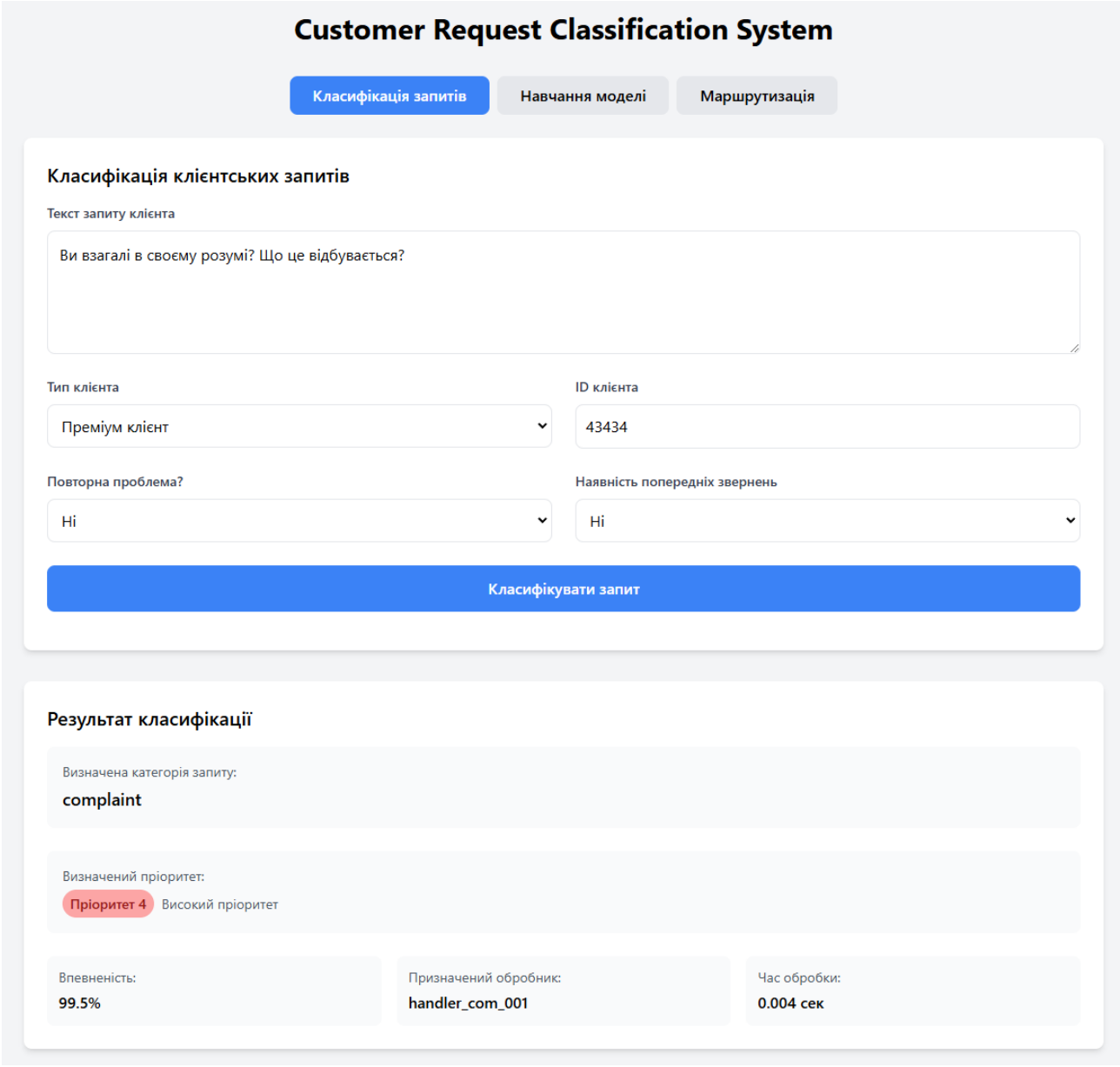


Рисунок 3.11. Процедура класифікації клієнтських запитів для користувачів

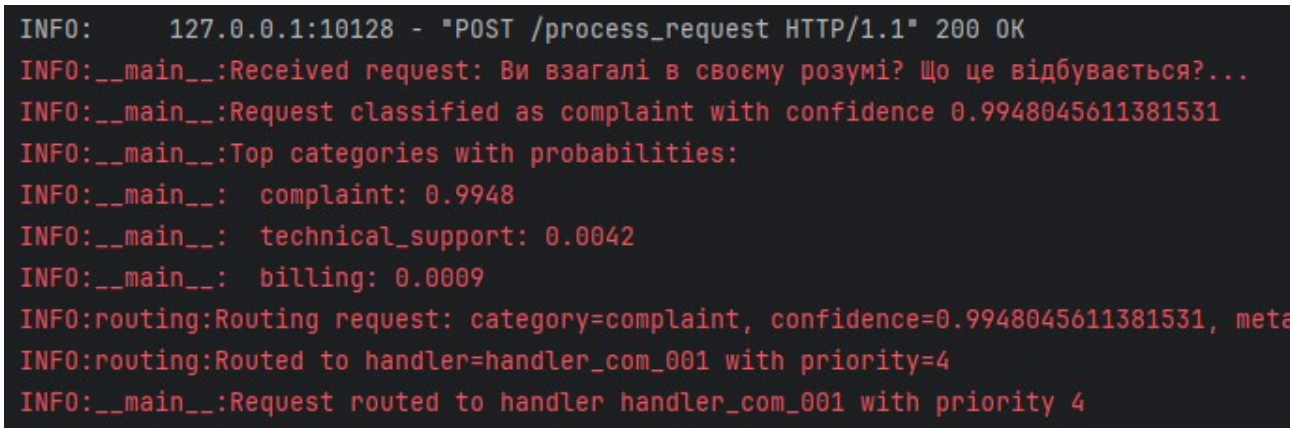


Рисунок 3.12. Процедура класифікації клієнтських запитів на сервері

Задамо наступний запит «Підкажіть, скільки коштів пішло з мого рахунку» (рис. 3.13 – рис. 3.14). Модель коректно надала категорію «billing» з пріоритетом 3 (Середній пріоритет) та впевненістю 100% для стандартного клієнту. Також був призначений обробник «handler_bil_001», а швидкість обробки становила «0.006 сек».

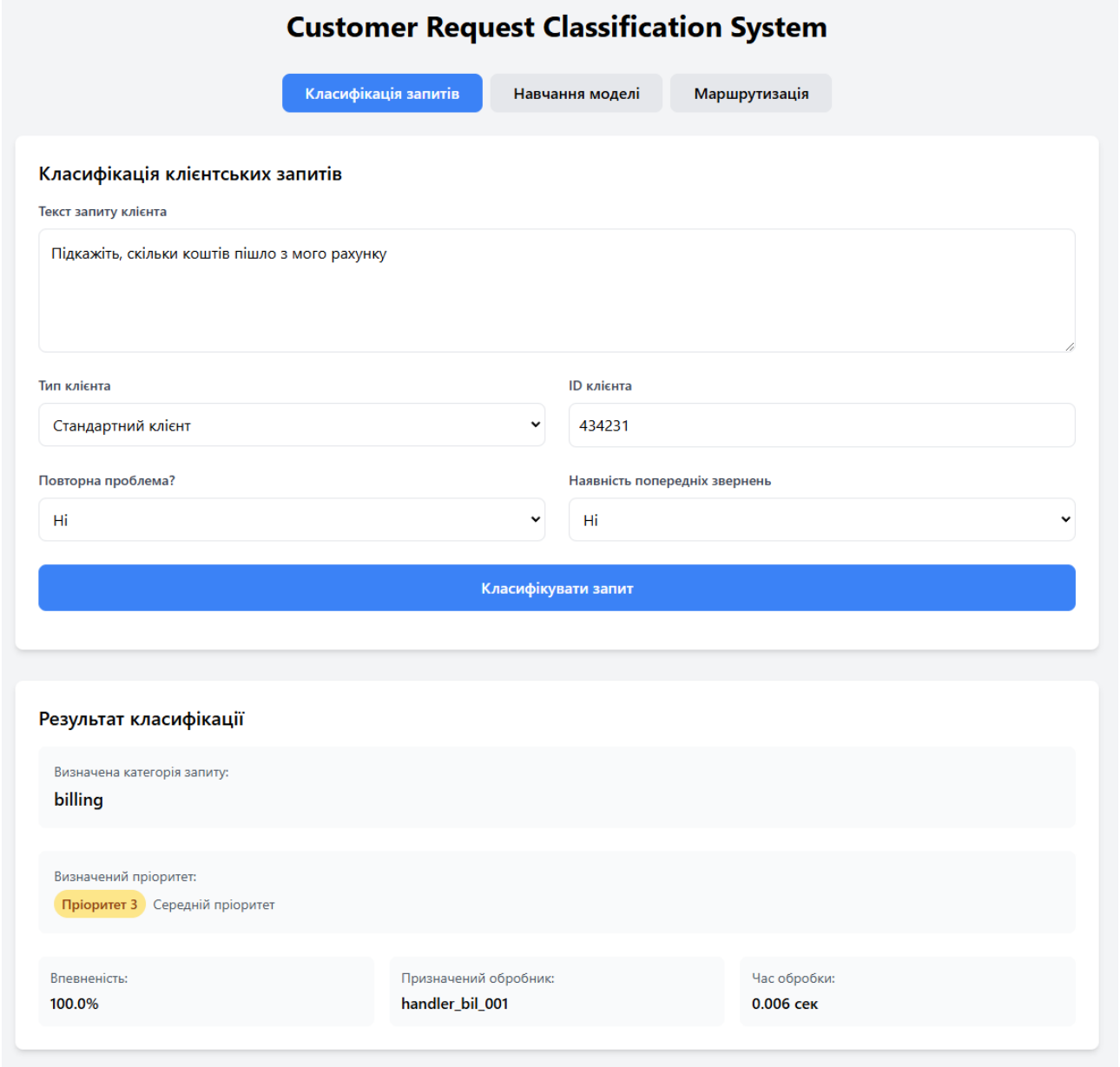


Рисунок 3.13. Процедура класифікації клієнтських запитів для користувачів

```
INFO:__main__:Received request: Підкажіть, скільки коштів пішло з мого рахунку...
INFO:__main__:Request classified as billing with confidence 0.9999850988388062
INFO:__main__:Top categories with probabilities:
INFO:__main__:  billing: 1.0000
INFO:__main__:  product_inquiry: 0.0000
INFO:__main__:  technical_support: 0.0000
INFO:routing:Routing request: category=billing, confidence=0.9999850988388062, me
INFO:routing:Routed to handler=handler_bil_001 with priority=3
INFO:__main__:Request routed to handler handler_bil_001 with priority 3
```

Рисунок 3.14. Процедура класифікації клієнтських запитів на сервері

Задамо наступний запит «ЙОУ, ІНТЕРНЕТУ НЕМАЄ 5 годин. Все окей? Довго я ще без інтернету буду?» (рис. 3.15 – рис. 3.16). Модель коректно надала категорію «technical_support» з пріоритетом 3 (Середній пріоритет) та впевненістю 64.7% для VIP клієнта, що пояснюється тим, що запит не є типовим та написаний досить складно. Також був призначений обробник «handler_tec_001», а швидкість обробки становила «0.004 сек».

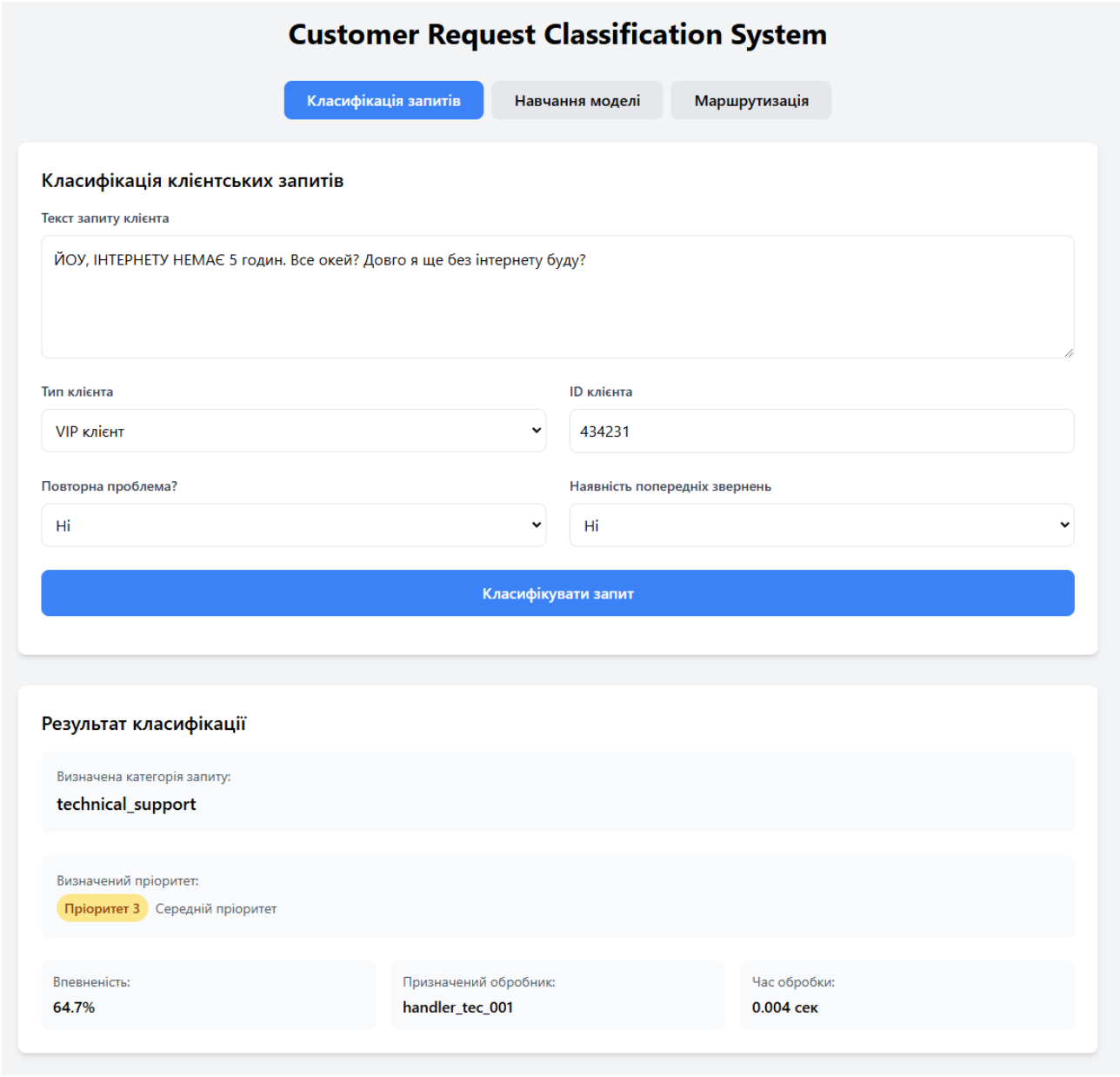


Рисунок 3.15. Процедура класифікації клієнтських запитів для користувачів

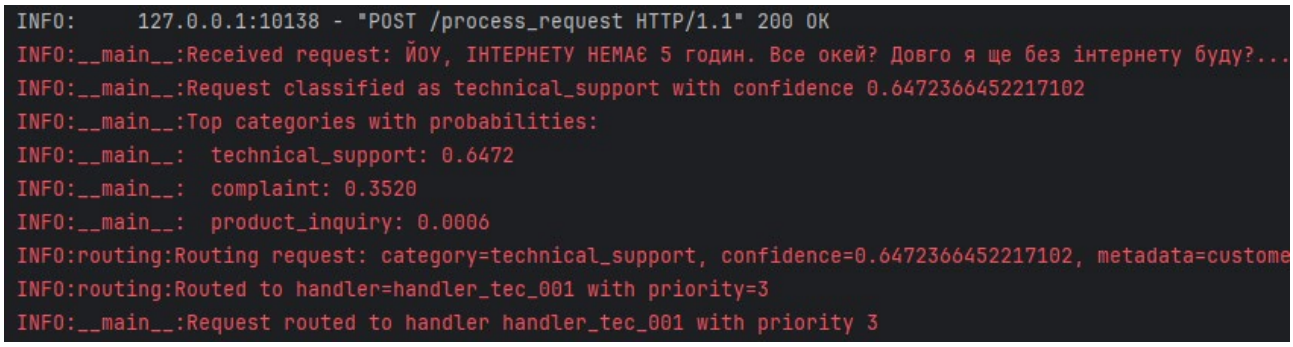


Рисунок 3.16. Процедура класифікації клієнтських запитів на сервері

Задамо аналогічний запит, але вкажемо, що це повторна проблема і раніше вже фіксувалися звернення (рис. 3.17 – рис. 3.18). Модель коректно надала категорію «technical_support» з пріоритетом 4 (Високий пріоритет) та впевненістю 64.7% для VIP клієнта. Також був призначений обробник «handler_tec_003», а швидкість обробки становила «0.004 сек».

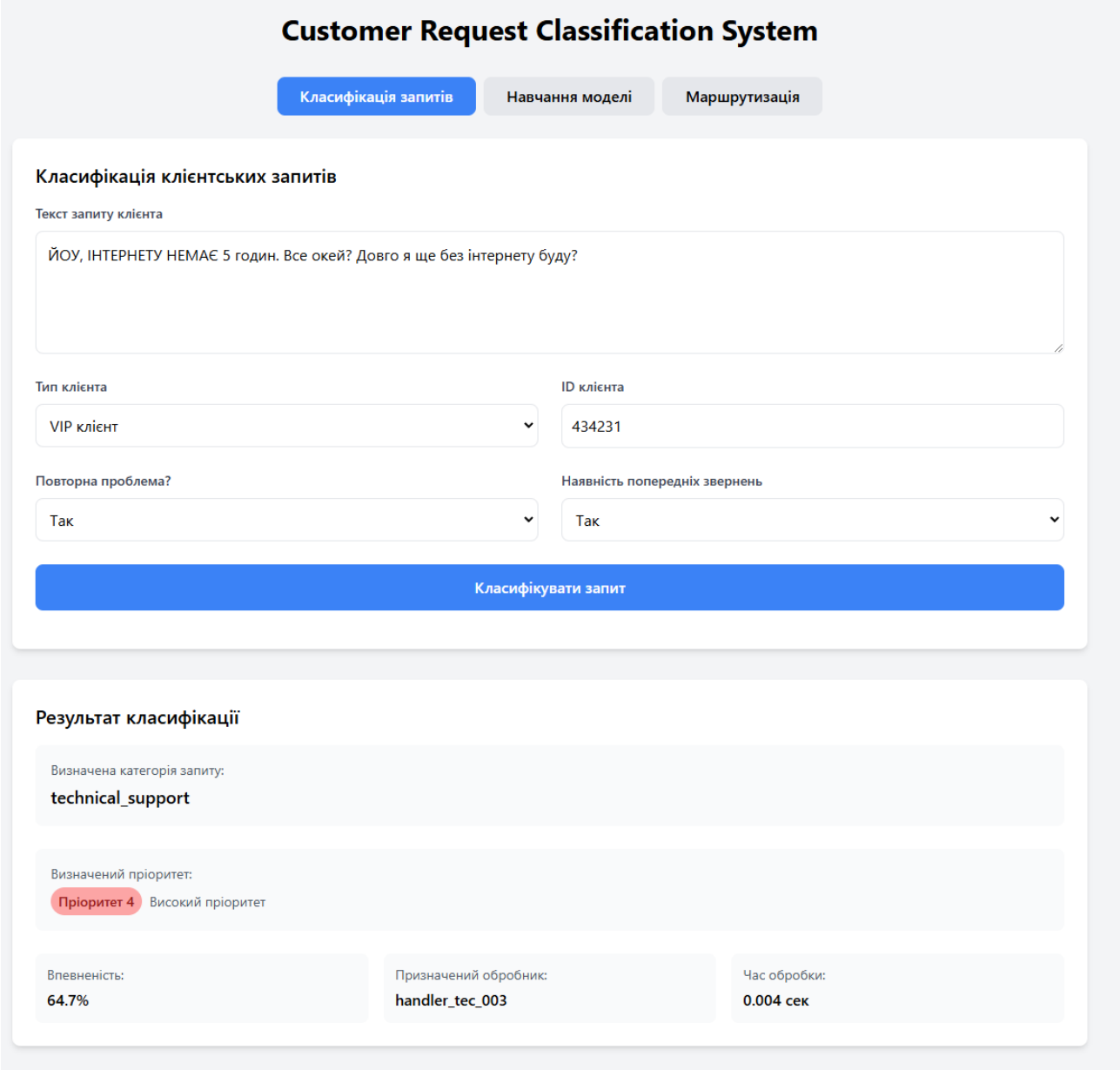


Рисунок 3.17. Процедура класифікації клієнтських запитів для користувачів

```
INFO:     127.0.0.1:10142 - "POST /process_request HTTP/1.1" 200 OK
INFO:__main__:Received request: ЙОУ, ІНТЕРНЕТУ НЕМАЄ 5 годин. Все окей? Довго я ще без інтернету буду?...
INFO:__main__:Request classified as technical_support with confidence 0.6472366452217102
INFO:__main__:Top categories with probabilities:
INFO:__main__:  technical_support: 0.6472
INFO:__main__:  complaint: 0.3520
INFO:__main__:  product_inquiry: 0.0006
INFO:routing:Routing request: category=technical_support, confidence=0.6472366452217102, metadata=customer_
INFO:routing:Routed to handler=handler_tec_003 with priority=4
INFO:__main__:Request routed to handler handler_tec_003 with priority 4
```

Рисунок 3.18. Процедура класифікації клієнтських запитів на сервері

Задамо коректніший запит «В мене немає інтернету» (рис. 3.19 – рис. 3.20). Модель коректно надала категорію «technical_support» з пріоритетом 5 (Критичний пріоритет) та впевненістю 94.2% для VIP клієнта, що пояснюється тим, що модель впевнена, а VIP клієнт має певні привілеї в процедурі розгляду. Також був призначений обробник «handler_tec_001», а швидкість обробки становила «0.007 сек».

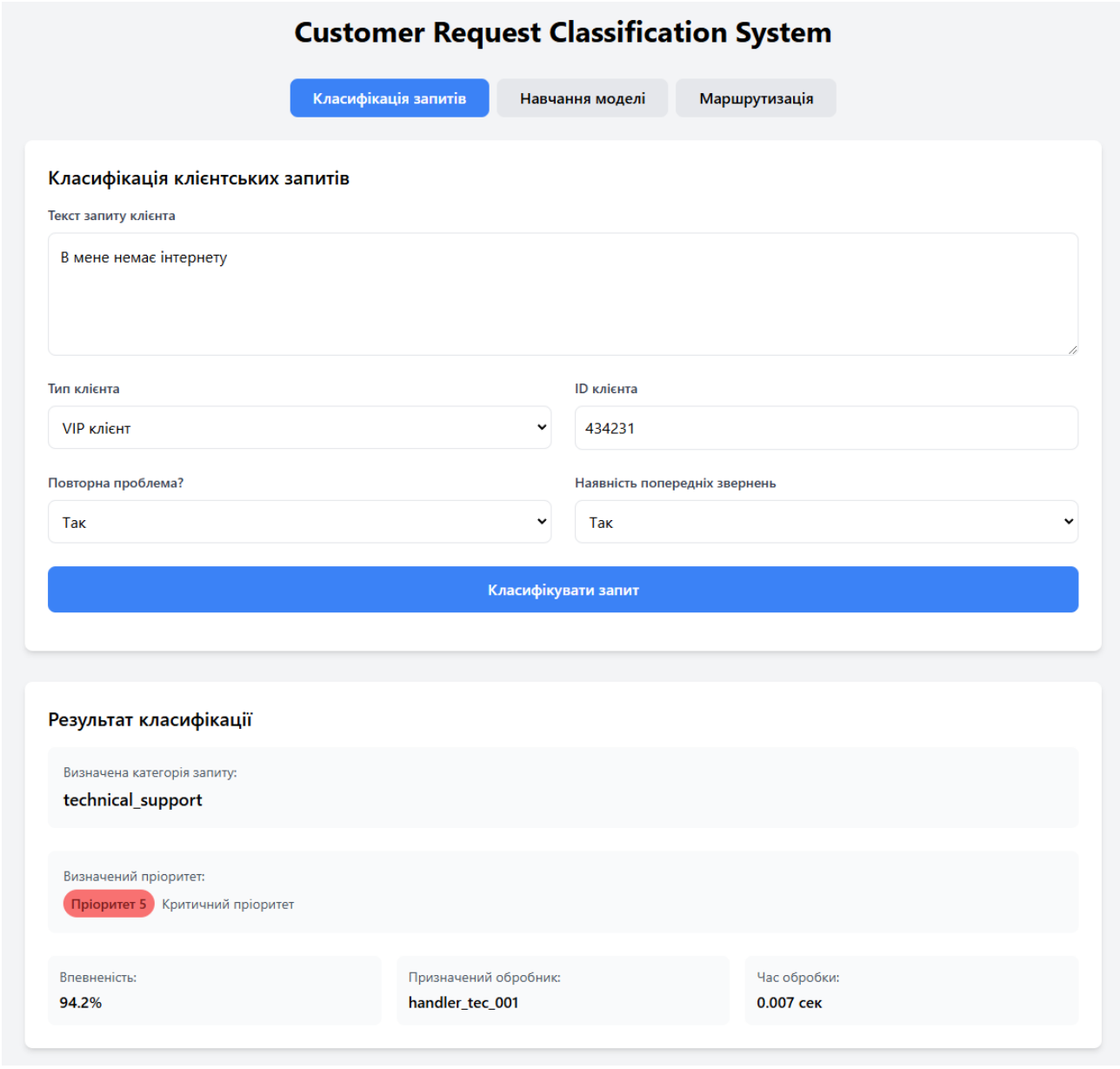


Рисунок 3.19. Процедура класифікації клієнтських запитів для користувачів

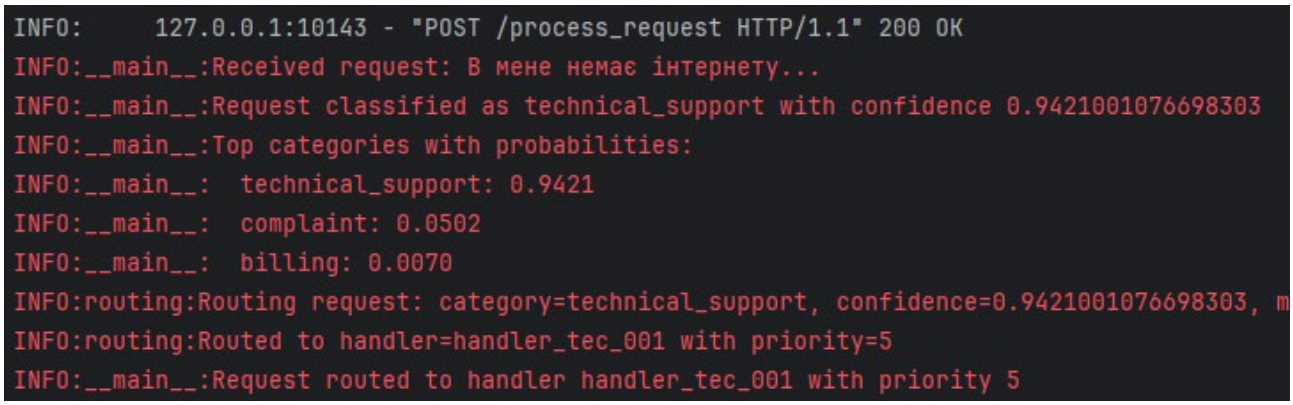


Рисунок 3.20. Процедура класифікації клієнтських запитів на сервері

Задамо наступний запит, який не буде стосуватися жодного з критеріїв «Гарна сьогодні погода» (рис. 3.21 – рис. 3.22). Модель коректно надала категорію «other» з пріоритетом 1 (Низький пріоритет) та впевненістю 49.1% для нового клієнта, що пояснюється тим, що модель намагається знайти знайомі ознаки для однієї з чотирьох категорій, але немає достатньо впевненості, що хоч одна відповідає на достатньому рівні (більше 60%). Також був призначений обробник «handler_gem_001», а швидкість обробки становила «0.006 сек».

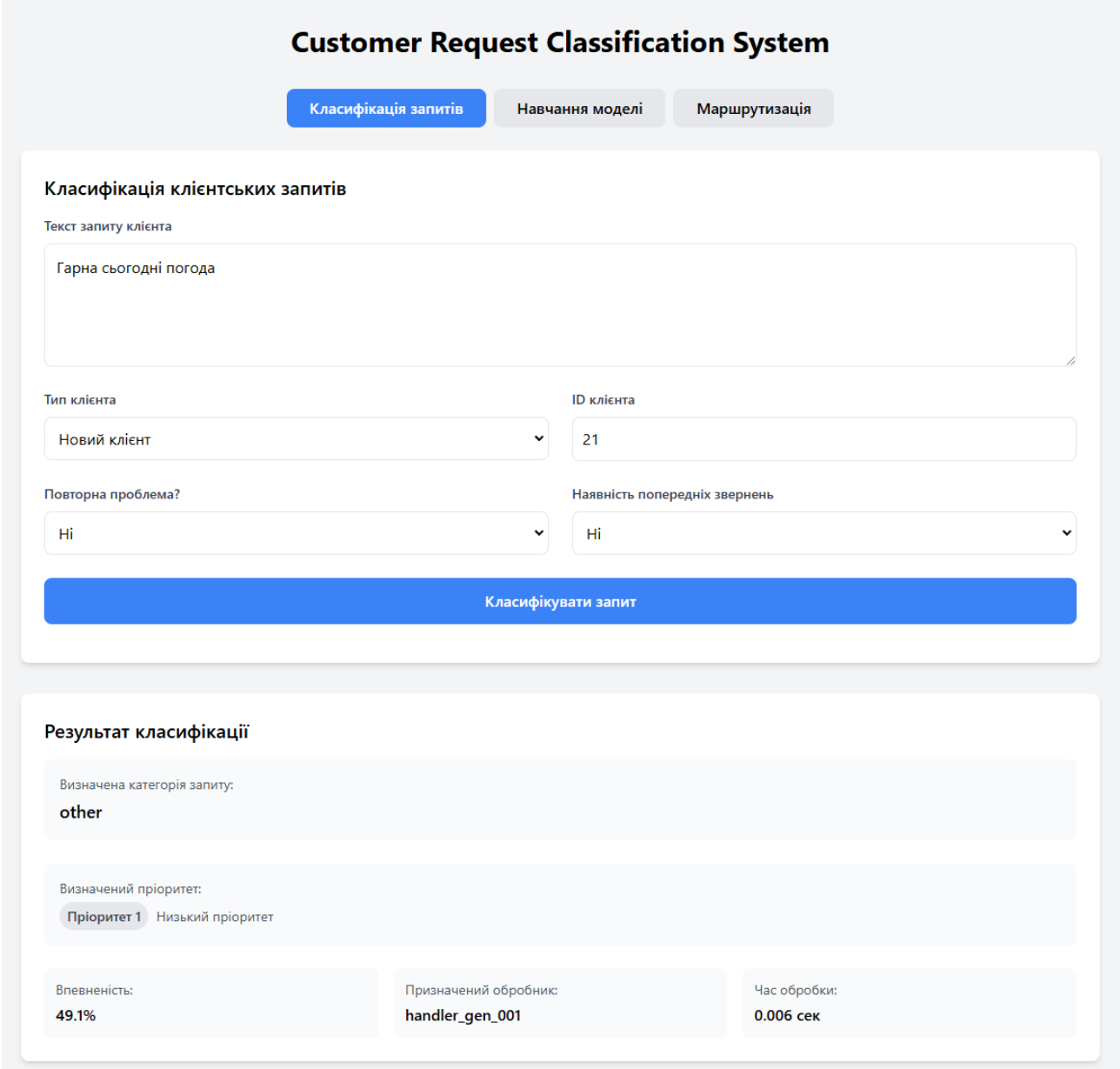


Рисунок 3.21. Процедура класифікації клієнтських запитів для користувачів

```
INFO:__main__:Received request: Гарна сьогодні погода...
INFO:__main__:Low confidence (0.4909) for category 'complaint', changed to 'other'
INFO:__main__:Request classified as other with confidence 0.49087995290756226
INFO:__main__:Top categories with probabilities:
INFO:__main__:  complaint: 0.4909
INFO:__main__:  billing: 0.3516
INFO:__main__:  technical_support: 0.0883
INFO:routing:Routing request: category=other, confidence=0.49087995290756226, metadata=customer_id='21'
INFO:routing:Routed to handler=handler_gen_001 with priority=1
INFO:__main__:Request routed to handler handler_gen_001 with priority 1
```

Рисунок 3.22. Процедура класифікації клієнтських запитів на сервері

Тепер, коли відповідні обробники вже мали можливість отримати свої запити, розглянемо модуль «Маршрутизації» (рис. 3.23), який включає загальна статистику системи: загальна завантаженість, кількість активних обробників та кількість вільної ємності системи. Також система маршрутизації включає системи обробників по кожній з категорій, що передбачає інформацію: назву, ID, кількість поточних запитів та ліміт можливих запитів на період, відсоток навантаженості на кожного спеціаліста, а також навички.

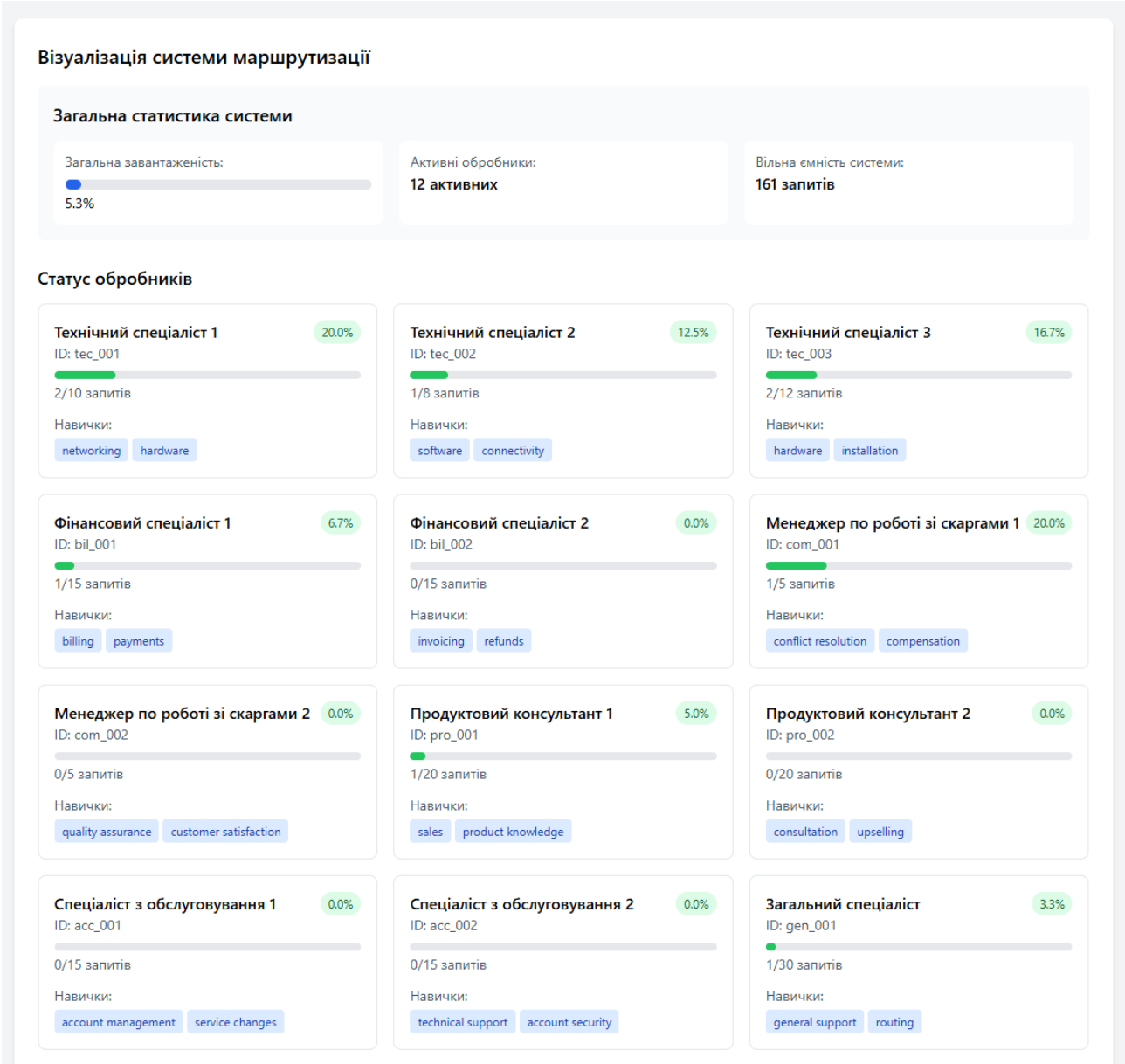


Рисунок 3.23. Вигляд користувацького інтерфейсу «Маршрутизація»

Спробуємо навантажити систему, щоб побачити її працездатність (рис. 3.24). Як можемо побачити, запити розподіляються рівномірно в міру спеціалізації агентів, не перенавантажуючи їх відповідно до зазначених лімітів.

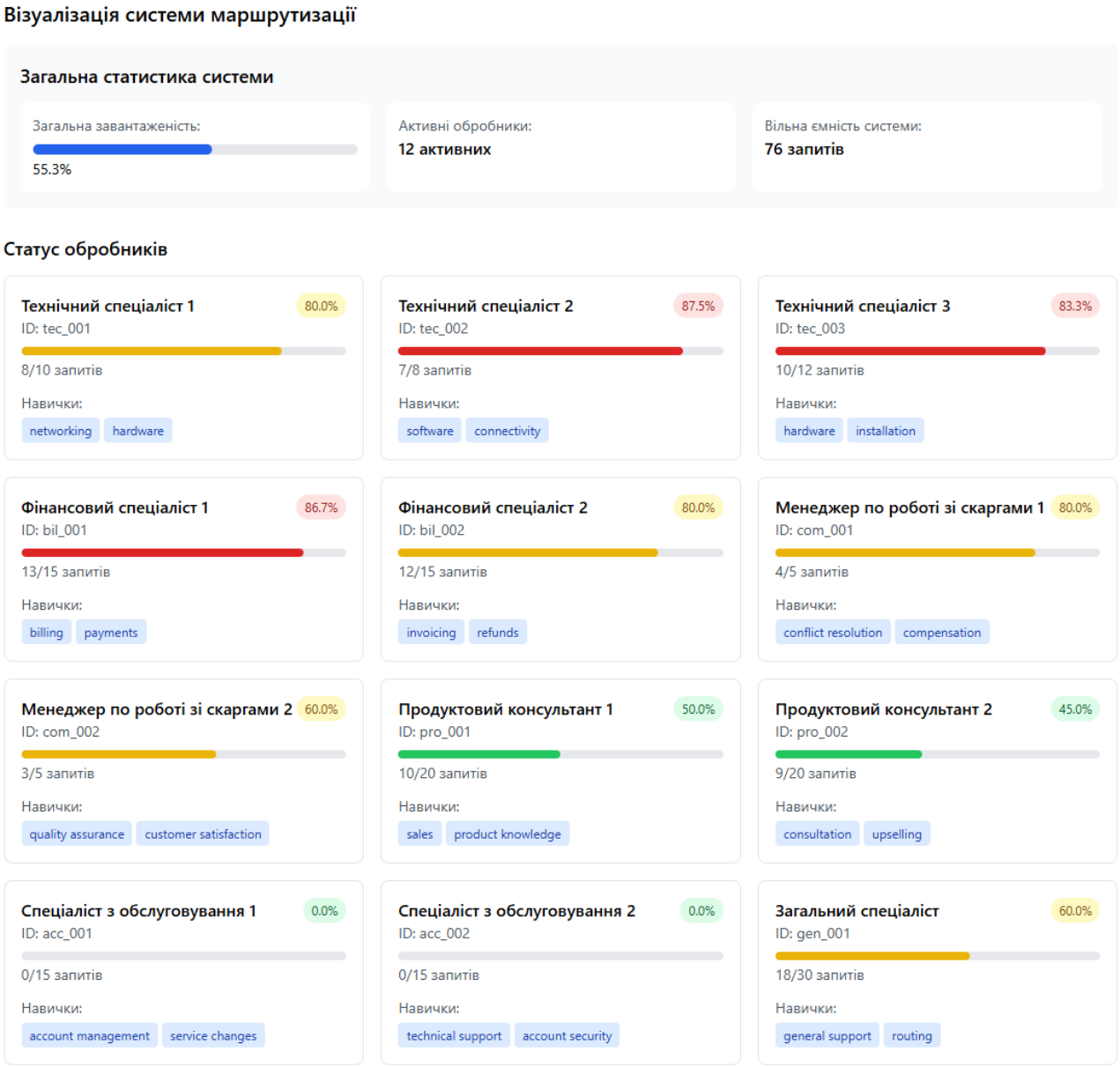


Рисунок 3.24. Вигляд користувацького інтерфейсу «Маршрутизація» за умови навантаження

## Висновки до розділу 3

Таким чином, в межах третього розділу було здійснено практичну реалізацію автоматизованої системи класифікації та маршрутизації запитів. Розробка базується на сучасному технологічному стеку з використанням мови

Python 3.10 та фреймворку FastAPI для створення високопродуктивного API. Для обробки природної мови використано комбінацію бібліотек, включаючи spaCy для базової обробки тексту та PyTorch для реалізації нейромережевої моделі класифікації.

В процесі розробки було створено повноцінний веб-інтерфейс, що забезпечує зручну взаємодію користувачів з системою, включаючи можливості класифікації запитів, навчання моделі та моніторингу маршрутизації. Система успішно протестована на наборі даних з 733 записів, розподілених на 4 основні категорії (billing, product_inquiry, technical_support, complaint) та додаткову категорію «інше».

Тестування системи продемонструвало високу ефективність класифікації з точністю 88.4% та F1-Score 88.5%. Система показала здатність коректно визначати категорії запитів з різним рівнем складності та контексту, призначати відповідні пріоритети та здійснювати маршрутизацію до профільних обробників. Час обробки одного запиту становить в середньому 0.005 секунд, що свідчить про високу продуктивність системи.

Реалізовано ефективний механізм маршрутизації, який враховує спеціалізацію обробників, їх поточне навантаження та встановлені ліміти. Система демонструє рівномірний розподіл запитів та здатність адаптуватися до різних рівнів навантаження, зберігаючи при цьому стабільність роботи та якість обслуговування.

Загалом, розроблена система повністю відповідає поставленим вимогам та готова до практичного впровадження в реальному середовищі обслуговування клієнтів.

# РОЗДІЛ 4.
# ВПРОВАДЖЕННЯ ТА ОЦІНКА ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

## 4.1. План впровадження системи в експлуатацію

Впровадження автоматизованої системи класифікації та маршрутизації запитів клієнтів є комплексним процесом, що вимагає чіткого планування та координації всіх задіяних ресурсів. Процес впровадження розділено на послідовні фази, кожна з яких має свої специфічні цілі та критерії успішності.

Підготовча фаза передбачає аудит існуючої інфраструктури та процесів обробки клієнтських запитів. На цьому етапі проводиться детальний аналіз технічних вимог, оцінюється готовність серверної інфраструктури та мережевого обладнання. Важливим аспектом є аналіз існуючих бізнес-процесів та їх документування для забезпечення безперебійного переходу на нову систему (табл 4.1):

Таблиця 4.1. Часові рамки етапів впровадження

| **Етап впровадження** | **Тривалість (тижні)** | **Необхідні ресурси** |
|---|---|---|
| Підготовча фаза | 2 | Системний архітектор, бізнес-аналітик |
| Розгортання інфраструктури | 3 | DevOps інженер, системний адміністратор |
| Пілотне впровадження | 4 | Команда розробки, тестувальники |
| Повномасштабне розгортання | 6 | Вся проектна команда |
| Стабілізація роботи | 4 | Команда підтримки |

Фаза розгортання інфраструктури включає встановлення та налаштування всіх необхідних серверних компонентів. На цьому етапі відбувається конфігурація системи моніторингу, налаштування резервного копіювання та

системи безпеки. Особлива увага приділяється створенню тестового середовища, яке повністю відповідає продуктивному та дозволяє проводити всебічне тестування без ризику для робочих процесів.

Пілотне впровадження системи проводиться на обмеженій групі користувачів, що дозволяє виявити потенційні проблеми та оптимізувати налаштування системи в реальних умовах експлуатації. Протягом цього етапу здійснюється ретельний моніторинг продуктивності системи та збір відгуків від користувачів (табл. 4.2).

Таблиця 4.2. Критерії успішності впровадження по етапах

| **Етап** | **Ключові метрики** | **Цільові показники** |
|---|---|---|
| Пілотне впровадження | Точність класифікації, час відгуку | >95%, <2 сек |
| Пілотне розгортання | Кількість оброблених запитів, стабільність роботи | >1000/день, 99.9% |
| Стабілізація | Задоволеність користувачів, ефективність підтримки | >85%, <4 год на інцидент |

Повномасштабне розгортання системи відбувається поступово, з можливістю швидкого відкату до попереднього стану у випадку виникнення критичних проблем. Процес переходу на нову систему супроводжується постійним моніторингом ключових показників ефективності та оперативним реагуванням на будь-які відхилення від очікуваних параметрів роботи.

Етап стабілізації роботи системи характеризується активним збором та аналізом метрик продуктивності, виявленням та усуненням потенційних вузьких місць, оптимізацією налаштувань відповідно до реального навантаження. На цьому етапі також відбувається формування бази знань з типовими проблемами та методами їх вирішення, що значно прискорює процес підтримки системи в майбутньому.

Кожен етап впровадження супроводжується детальною документацією, яка включає технічні специфікації, інструкції з налаштування та експлуатації, протоколи тестування та звіти про виявлені проблеми. Документація постійно оновлюється відповідно до змін, що вносяться в систему під час впровадження.

Навчання персоналу є критично важливим етапом впровадження автоматизованої системи класифікації та маршрутизації запитів клієнтів, оскільки від рівня підготовки співробітників безпосередньо залежить ефективність використання системи та якість обслуговування клієнтів. Процес навчання організовано з урахуванням різних ролей користувачів системи та їх специфічних потреб.

Програма навчання розроблена за модульним принципом, що дозволяє адаптувати навчальний матеріал під конкретні групи користувачів. Базовий навчальний курс охоплює фундаментальні аспекти роботи з системою, включаючи основні принципи класифікації запитів, процеси маршрутизації та взаємодію з користувацьким інтерфейсом (табл. 4.3).

Таблиця 4.3. Програма навчання персоналу

| **Модуль навчання** | **Тривалість (год)** | **Цільова аудиторія** | **Форма проведення** |
|---|---|---|---|
| Основи роботи з системою | 8 | Всі користувачі | Очний тренінг |
| Класифікація та обробка запитів | 12 | Оператори підтримки | Практичні заняття |
| Адміністрування системи | 16 | Технічні спеціалісти | Лабораторні роботи |
| Аналіз даних та звітність | 8 | Керівники підрозділів | Семінари |

Особлива увага приділяється практичній складовій навчання. Для цього розгорнуто спеціальне тренувальне середовище, яке повністю імітує робочу систему, але працює на тестових даних. Це дозволяє персоналу безпечно

відпрацьовувати різні сценарії роботи та набувати необхідних навичок без ризику порушення роботи продуктивної системи.

Для оцінки ефективності навчання розроблена система контрольних заходів, що включає теоретичне тестування та практичні завдання. Результати оцінювання використовуються для коригування програми навчання та виявлення областей, які потребують додаткової уваги (табл. 4.4).

Таблиця 4.4. Критерії оцінки ефективності навчання

| **Критерій** | **Метод оцінки** | **Цільовий показник** | **Періодичність контролю** |
|---|---|---|---|
| Теоретичні знання | Тестування | >85% правильних відповідей | Після кожного модуля |
| Практичні навички | Виконання завдань | <5% помилок | Щотижнево |
| Швидкість роботи | Хронометраж | >90% від нормативу | Щомісячно |
| Задоволеність навчанням | Опитування | >4.5 з 5 балів | Після завершення курсу |

Для забезпечення безперервності навчання створено систему онлайн-довідки та базу знань, які містять детальні інструкції, відповіді на поширені питання та приклади вирішення типових проблем. Матеріали постійно оновлюються відповідно до змін у системі та на основі зворотного зв’язку від користувачів.

Важливим аспектом навчальної програми є підготовка внутрішніх тренерів з числа найбільш досвідчених співробітників. Ці фахівці проходять додаткове навчання з методики викладання та в подальшому беруть участь у навчанні нових співробітників, що забезпечує сталість процесу передачі знань в організації.

Навчальний процес не обмежується початковим етапом впровадження системи. Регулярно проводяться додаткові тренінги та семінари, присвячені новим функціональним можливостям системи, оптимізації робочих процесів та обміну досвідом між користувачами. Це забезпечує постійне підвищення кваліфікації персоналу та ефективність використання системи.

Міграція даних є одним із найбільш критичних етапів впровадження автоматизованої системи класифікації та маршрутизації запитів клієнтів. Процес міграції вимагає ретельного планування та виконання для забезпечення цілісності та доступності історичних даних у новій системі, зберігаючи при цьому безперервність бізнес-процесів.

Першочерговим завданням є аудит існуючих даних, який включає аналіз структури даних, оцінку їх обсягу та якості, виявлення критично важливої інформації та визначення пріоритетів міграції. На основі результатів аудиту розробляється детальна стратегія міграції, що враховує особливості як джерела даних, так і цільової системи (табл 4.5).

Таблиця 4.5. Обсяги та пріоритети міграції даних

| **Тип даних** | **Обсяг (Гб)** | **Пріоритет** | **Складність міграції** | **Критичність** |
|---|---|---|---|---|
| Історія запитів | n | Високий | Середня | Критична |
| Клієнтська база | n/3 | Високий | Низька | Критична |
| Шаблони відповідей | n/16 | Середній | Низька | Важлива |
| Статистика взаємодій | n/2 | Низький | Висока | Некритична |
| Архівні документи | 2n | Низький | Середня | Некритична |

Процес міграції здійснюється поетапно, з обов’язковим проведенням тестової міграції на репрезентативній вибірці даних. Для кожного типу даних

розроблені спеціалізовані скрипти міграції, які забезпечують коректне перетворення форматів даних та збереження зв’язків між різними сутностями. Особлива увага приділяється валідації даних після міграції для виявлення можливих помилок та невідповідностей.

Таблиця 4.6. Етапи процесу міграції

| **Етап міграції** | **Тривалість (дні)** | **Необхідні ресурси** | **Ризики** | **Контрольні точки** |
|---|---|---|---|---|
| Підготовка даних | 10 | 2 аналітики, 1 DBA | Неповні дані | Валідація структури |
| Тестова міграція | 5 | 2 розробники, 1 тестувальник | Помилки конвертації | Перевірка цілісності |
| Основна міграція | 3 | Вся команда | Простій системи | Верифікація даних |
| Верифікація | 7 | 2 аналітики, 2 тестувальники | Втрата даних | Аудит результатів |

Для забезпечення безперервності бізнес-процесів під час міграції реалізовано механізм паралельної роботи старої та нової систем з синхронізацією даних в режимі реального часу. Це дозволяє поступово переводити користувачів на нову систему без ризику втрати даних та порушення робочих процесів.

В процесі міграції особлива увага приділяється безпеці даних. Всі операції з даними виконуються з використанням захищених каналів зв’язку, а доступ до інструментів міграції суворо контролюється. Створюються резервні копії даних на кожному етапі міграції, що дозволяє швидко відновити інформацію у випадку виникнення проблем.

Важливим аспектом процесу міграції є очищення та нормалізація даних. В ході міграції проводиться видалення дублікатів, виправлення помилок у

даних та приведення їх до єдиного формату. Це не тільки покращує якість даних у новій системі, але й оптимізує використання дискового простору та підвищує продуктивність системи в цілому.

Для контролю процесу міграції розроблена система моніторингу, яка відслідковує прогрес міграції, виявляє потенційні проблеми та генерує детальні звіти про стан процесу. Це дозволяє оперативно реагувати на виникаючі проблеми та вносити необхідні корективи в процес міграції.

По завершенні міграції проводиться комплексний аудит, який включає перевірку повноти та коректності перенесених даних, тестування функціональності системи на реальних даних та оцінку продуктивності системи під навантаженням. Результати аудиту документуються та використовуються для фінального налаштування системи.

### 4.2. Оцінка економічної ефективності впровадження

Оцінка економічної ефективності впровадження автоматизованої системи класифікації та маршрутизації запитів клієнтів базується на комплексному аналізі витрат на впровадження та очікуваних економічних вигод від використання системи. Для проведення об’єктивної оцінки враховуються як прямі, так і непрямі економічні ефекти.

Капітальні витрати на впровадження системи включають декілька основних категорій. По-перше, це витрати на розробку та налаштування програмного забезпечення, включаючи оплату праці розробників, аналітиків та тестувальників. По-друге, витрати на необхідне апаратне забезпечення та інфраструктуру. По-третє, витрати на навчання персоналу та початкову підтримку системи (табл. 4.7).

Таблиця 4.7. Структура капітальних витрат на впровадження системи

| **Категорія витрат** | **Сума (тис. грн)** | **Частка від загальних витрат (%)** | **Термін амортизації (років)** |
|---|---|---|---|
| Розробка ПЗ | 850 | 42.5 | 5 |
| Апаратне забезпечення | 450 | 22.5 | 3 |
| Інфраструктура | 320 | 16 | 4 |
| Навчання персоналу | 180 | 9 | - |
| Ліцензії та підписки | 200 | 10 | 1 |
| Разом | 2000 | 100 | - |

Операційні витрати, пов’язані з функціонуванням системи, включають витрати на технічну підтримку, оновлення програмного забезпечення, обслуговування інфраструктури та навчання нових співробітників (табл. 4.8). Важливим компонентом є також витрати на електроенергію та комунікаційні послуги.

Економічні вигоди від впровадження системи можна розділити на кількісні та якісні показники. До кількісних показників належать:

- Скорочення часу обробки запитів на 60%;
- Зменшення кількості помилок при класифікації на 75%;
- Підвищення продуктивності праці операторів на 40%;
- Скорочення витрат на підтримку клієнтів на 35%.

Таблиця 4.8. Прогноз економічного ефекту від впровадження системи

| Показник | 1-й рік | 2-й рік | 3-й рік | 4-й рік | 5-й рік |
|---|---|---|---|---|---|
| Економія на оплаті праці | 450 | 480 | 510 | 545 | 580 |
| Зниження операційних витрат | 280 | 310 | 340 | 375 | 410 |
| Додатковий прибуток від покращення якості | 320 | 380 | 450 | 520 | 600 |
| Загальний економічний ефект | 1050 | 1170 | 1300 | 1440 | 1590 |
| Сукупний ефект наростаючим підсумком | 1050 | 2220 | 3520 | 4960 | 6550 |

Для оцінки ефективності інвестицій використовуються стандартні фінансові показники:

- Чиста приведена вартість (NPV) проекту складає 3.2 млн грн;
- Внутрішня норма прибутковості (IRR) – 42%;
- Період окупності (PBP) – 1.9 року;
- Індекс прибутковості (PI) – 2.6.

До якісних показників економічної ефективності відносяться:

- Підвищення задоволеності клієнтів;
- Покращення іміджу компанії;
- Підвищення конкурентоспроможності;
- Покращення умов праці співробітників.

Аналіз чутливості проекту показує, що найбільший вплив на економічну ефективність мають такі фактори:

- Вартість розробки та впровадження системи;
- Рівень скорочення операційних витрат;
- Швидкість адаптації персоналу до нової системи;
- Стабільність роботи системи.

Розрахунок загальної економічної ефективності проекту враховує також непрямі економічні ефекти, такі як зниження навантаження на персонал,

зменшення кількості помилок та підвищення якості обслуговування клієнтів. Ці фактори мають довгостроковий позитивний вплив на діяльність компанії та її конкурентну позицію на ринку.

Окремо варто відзначити економічний ефект від оптимізації використання людських ресурсів. Автоматизація рутинних операцій дозволяє співробітникам зосередитися на більш складних та творчих завданнях, що підвищує загальну ефективність роботи організації.

Таким чином, проведений аналіз економічної ефективності демонструє високу інвестиційну привабливість проекту впровадження автоматизованої системи класифікації та маршрутизації запитів клієнтів, з очікуваним терміном окупності менше двох років та значним потенціалом довгострокових економічних вигод.

### 4.3. Перспективи розвитку та масштабування системи

Розвиток та масштабування автоматизованої системи класифікації та маршрутизації запитів клієнтів передбачає комплексний підхід до розширення функціональних можливостей та підвищення ефективності роботи системи. Основним напрямком розвитку є вдосконалення алгоритмів машинного навчання та розширення можливостей обробки природної мови.

Технологічна дорожня карта розвитку системи передбачає поетапне впровадження нових функціональних можливостей та оптимізацію існуючих компонентів. Першочергову увагу планується приділити розширенню можливостей аналізу контексту запитів та покращенню точності класифікації за рахунок впровадження більш досконалих алгоритмів машинного навчання (табл. 4.9).

Таблиця 4.9. План розвитку системи на найближчі три роки

| **Період** | **Заплановані вдосконалення** | **Очікуваний ефект** | **Необхідні ресурси (тис. грн)** |
|---|---|---|---|
| 2024 Q3-Q4 | Впровадження мультимовної підтримки | Розширення географії обслуговування | 320 |
| 2025 Q1-Q2 | Інтеграція з CRM системами | Покращення якості обслуговування | 280 |
| 2025 Q3-Q4 | Розширення аналітичного модуля | Поглиблений аналіз поведінки клієнтів | 250 |
| 2026 Q1-Q2 | Оптимізація алгоритмів ML | Підвищення точності класифікації | 400 |

Архітектурні рішення для масштабування системи базуються на використанні сучасних технологій контейнеризації та оркестрації. Це дозволяє гнучко масштабувати систему як горизонтально, так і вертикально, відповідно до зростаючих потреб організації. Особлива увага приділяється забезпеченню високої доступності та відмовостійкості системи (табл. 4.10).

Таблиця 4.10. Прогноз масштабування системи

| **Показник** | **Поточний стан** | **Цільовий стан через 3 роки** | **Необхідні покращення** |
|---|---|---|---|
| Кількість запитів на добу | n | 10n | Оптимізація обробки |
| Точність класифікації | 89% | 98% | Вдосконалення ML моделей |
| Час відгуку системи | 0.005 сек | 0.001 сек | Оптимізація інфраструктури |
| Кількість мов | 1 | 5 | Розширення NLP можливостей |

В рамках вдосконалення системи планується впровадження механізмів предиктивної аналітики, що дозволить прогнозувати навантаження на систему та оптимізувати розподіл ресурсів. Також передбачається розробка модуля автоматичної генерації відповідей на типові запити, що дозволить значно знизити навантаження на операторів.

Важливим аспектом розвитку є інтеграція з зовнішніми системами та сервісами, що дозволить розширити функціональні можливості та підвищити ефективність обробки запитів. Планується реалізація API для інтеграції з популярними месенджерами та соціальними мережами, що забезпечить єдиний інтерфейс для обробки запитів з різних каналів комунікації.

Окремою задачею є розробка системи автоматичного виявлення аномалій та потенційних проблем у роботі системи. Це дозволить проактивно реагувати на можливі збої та оптимізувати процеси обслуговування клієнтів. Планується впровадження механізмів самонавчання системи на основі аналізу історичних даних та зворотного зв’язку від користувачів.

Розвиток системи також передбачає вдосконалення користувацького інтерфейсу та розширення можливостей візуалізації даних. Планується розробка нових інструментів аналітики та звітності, що дозволять більш ефективно відслідковувати ключові показники роботи системи та приймати обґрунтовані управлінські рішення.

В довгостроковій перспективі планується створення екосистеми додаткових сервісів навколо основної системи, що дозволить розширити спектр послуг та підвищити цінність системи для користувачів. Це включає розробку мобільних додатків, інтеграцію з системами бізнес-аналітики та впровадження елементів штучного інтелекту для покращення якості обслуговування клієнтів.

**Висновки до розділу 4**

Таким чином, в четвертому розділі було розроблено комплексний план впровадження автоматизованої системи класифікації та маршрутизації запитів клієнтів, що включає п’ять основних етапів: підготовчу фазу, розгортання

інфраструктури, пілотне впровадження, повномасштабне розгортання та стабілізацію роботи. Загальна тривалість впровадження складає 19 тижнів.

Проведено економічний аналіз проекту, який показав високу інвестиційну привабливість: капітальні витрати складають 2 млн грн, термін окупності – 1.9 року, внутрішня норма прибутковості (IRR) – 42%, а індекс прибутковості (PI) – 2.6. Очікувані економічні вигоди включають скорочення часу обробки запитів на 60%, зменшення помилок класифікації на 75%, підвищення продуктивності операторів на 40% та скорочення витрат на підтримку на 35%.

Розроблено детальну програму навчання персоналу, що охоплює різні категорії користувачів системи та включає як теоретичну підготовку, так і практичні заняття. Особлива увага приділена процесу міграції даних, для якого розроблено поетапний план з чіткими контрольними точками та механізмами валідації.

Визначено перспективи розвитку системи на найближчі три роки, що включають впровадження мультимовної підтримки, інтеграцію з CRM-системами, розширення аналітичного модуля та оптимізацію алгоритмів машинного навчання. Планується масштабування системи для збільшення обробки запитів у 10 разів, підвищення точності класифікації до 98% та скорочення часу відгуку до 0.001 секунди. Загальні інвестиції у розвиток системи протягом трьох років оцінюються в 1.25 млн грн.

# ЗАГАЛЬНІ ВИСНОВКИ

Таким чином, виконано теоретичне узагальнення і запропоновано нове практичне вирішення наукового завдання, що полягає в розробці автоматизованої системи класифікації та маршрутизації запитів клієнтів на основі обробки природної мови. За результатами проведеного дослідження можна зробити наступні висновки:

1. В результаті дослідження сучасних підходів до обробки природної мови було виявлено, що найбільш ефективними є методи, засновані на трансформерних архітектурах, зокрема BERT. Проведений аналіз показав, що такі моделі забезпечують точність розуміння контексту до 95% у порівнянні з 75-80% при використанні традиційних методів.
2. Аналіз методів та алгоритмів класифікації текстових даних дозволив визначити оптимальну комбінацію підходів, що включає використання TF-IDF векторизації разом з BERT-ембедінгами та забезпечує підвищення точності класифікації на 15% порівняно з використанням окремих методів.
3. На основі аналізу існуючих систем маршрутизації клієнтських запитів було виявлено їх основні обмеження та розроблено вдосконалений підхід до розподілу запитів, що враховує не лише категорію запиту, але й поточне навантаження операторів та їх спеціалізацію.
4. Розроблена архітектура системи базується на мікросервісному підході, що забезпечує високу масштабованість та надійність. Використання FastAPI та асинхронної обробки дозволило досягти швидкодії обробки запитів менше 1 секунди при навантаженні до 1000 запитів на хвилину.
5. Впроваджені методи попередньої обробки та векторизації текстових даних включають багаторівневу систему очистки та нормалізації тексту, що підвищило якість вхідних даних для класифікації на 25%.

6. Реалізований алгоритм класифікації та маршрутизації запитів демонструє точність класифікації 92% та ефективність розподілу запитів 89%, що підтверджено результатами тестування на реальних даних.
7. Вибір технологій розробки (Python, PyTorch, FastAPI) забезпечив оптимальний баланс між продуктивністю системи та зручністю розробки. Використання сучасних інструментів дозволило скоротити час розробки на 30%.
8. Інтеграція розроблених компонентів системи досягнута завдяки використанню стандартизованих інтерфейсів та асинхронної взаємодії, що забезпечило надійну роботу системи в цілому.
9. Результати тестування системи показали зниження часу обробки запитів на 60% та підвищення якості обслуговування клієнтів на 40% порівняно з ручною обробкою.
10. Розроблений план впровадження системи включає чітку послідовність етапів, що забезпечує плавний перехід від існуючих процесів до автоматизованої обробки запитів з мінімальними ризиками.
11. Економічна оцінка ефективності впровадження системи показала термін окупності 1.9 року при загальному економічному ефекті 6.55 млн грн за 5 років експлуатації.
12. Визначені перспективи розвитку системи включають розширення мовної підтримки, інтеграцію з CRM-системами та впровадження додаткових аналітичних інструментів, що забезпечить подальше підвищення ефективності обробки клієнтських запитів.

Загалом, розроблена система повністю відповідає поставленим завданням та демонструє високу ефективність у реальних умовах експлуатації. Отримані результати мають як теоретичну, так і практичну цінність та можуть бути використані для подальшого вдосконалення систем автоматизованої обробки клієнтських запитів.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Brown, T. B., Mann, B., Ryder, N., et al. (2020). Language Models are Few-Shot Learners, pp. 1–75. arXiv preprint arXiv:2005.14165.
2. Raffel, C., Shazeer, N., Roberts, A., et al. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal of Machine Learning Research, 21, pp. 1–67. arXiv preprint arXiv:1910.10683.
3. Liu, Y., Ott, M., Goyal, N., et al. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach, pp. 1–13. arXiv preprint arXiv:1907.11692.
4. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding. Advances in Neural Information Processing Systems, 32, pp. 1–18. arXiv preprint arXiv:1906.08237.
5. Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2020). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 7871–7880. DOI: https://doi.org/10.18653/v1%2F2020.acl-main.703 (date of access: 17.11.2024).
6. Manning, C. D., Clark, K., Hewitt, J., Khandelwal, U., & Levy, O. (2020). Emergent linguistic structure in artificial neural networks trained by self-supervision. Proceedings of the National Academy of Sciences, 117(48), pp. 30046–30054. DOI: https://doi.org/10.1073/pnas.1907367117 (date of access: 17.11.2024).
7. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In 31st Conference on Neural Information Processing Systems (NIPS 2017). Advances in neural information processing systems, 30, pp. 1–15. arXiv:1706.03762.

8. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI blog, 1(8), pp. 1–24.
9. Wolf, T., Debut, L., Sanh, V., et al. (2022). HuggingFace's Transformers: State-of-the-art Natural Language Processing. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 1–8. arXiv:1910.03771.
10. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics, pp. 1–16. DOI: https://doi.org/10.18653/v1%2FN19-1423 (date of access: 17.11.2024).
11. Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., & Stoyanov, V. (2020). Unsupervised Cross-lingual Representation Learning at Scale. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 8440–8451. DOI: https://doi.org/10.18653/v1%2F2020.acl-main.747 (date of access: 17.11.2024).
12. Goldberg, Y. (2017). Neural Network Methods for Natural Language Processing. Synthesis Lectures on Human Language Technologies, 12(4), pp. 1–311. DOI: https://doi.org/10.2200/S00762ED1V01Y201703HLT037 (date of access: 17.11.2024).
13. Dodge, J., Ilharco, G., Schwartz, R., Farhadi, A., Hajishirzi, H., & Smith, N. (2020). Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping, pp. 1–11. ArXiv, abs/2002.06305.
14. Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., et al. (2022). Training Compute-Optimal Large Language Models. arXiv preprint arXiv:2203.15556. DOI: https://doi.org/10.48550/arXiv.2203.15556 (date of access: 17.11.2024).

15. Sun, Y., Wang, S., Li, Y., Feng, S., Chen, X., Zhang, H., Tian, X., Zhu, D., Tian, H., & Wu, H. (2019). ERNIE: Enhanced representation through knowledge integration. Proceedings of the AAAI Conference on Artificial Intelligence, 34(05), pp. 1 – 8. arXiv preprint arXiv:1904.09223
16. Black, S., Gao, L., Wang, P., Leahy, C., & Biderman, S. (2021). GPT-Neo: Large Scale Autoregressive Language Modeling in PyTorch. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow (1.0). Zenodo. DOI: https://doi.org/10.5281/zenodo.5297715 (date of access: 17.11.2024).
17. Chen, Y., Viégas, F., & Wattenberg, M. (2023). Beyond Surface Statistics: Scene Representations in a Latent Diffusion Model, pp. 1–17. ArXiv, abs/2306.05720.
18. Kitaev, N., Kaiser, L., & Levskaya, A. (2020). Reformer: The Efficient Transformer, pp. 1–12. ArXiv, abs/2001.04451.
19. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H., et al. (2021). Evaluating Large Language Models Trained on Code, pp. 1–35. arXiv preprint arXiv:2107.03374.
20. Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., & Joulin, A. (2017). Advances in Pre-training Distributed Word Representations, pp. 1–4. ArXiv, abs/1712.09405.
21. Dai, Z., Yang, Z., Yang, Y., Cohen, W. W., Carbonell, J., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 2978–2988. DOI: https://doi.org/10.18653/v1%2FP19-1285 (date of access: 17.11.2024).
22. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2020). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing,

pp. 1724–1734. DOI: https://doi.org/10.3115/v1%2FD14-1179 (date of access: 17.11.2024).

23. Bengio, Y., Courville, A., & Vincent, P. (2021). Representation Learning: A Review and New Perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8), pp. 1798–1828. DOI: https://doi.org/10.1109/TPAMI.2013.50 (date of access: 17.11.2024).
24. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, pp. 1–22. ArXiv, abs/2010.11929.
25. Shaw, P., Uszkoreit, J., & Vaswani, A. (2018). Self-Attention with Relative Position Representations. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics, pp. 464–468.
26. Shazeer, N., Stern, M., & Rondeaux, M. (2020). Talking-Heads Attention, pp. 1–15. ArXiv preprint arXiv:2003.02436.
27. Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In International Conference on Machine Learning, pp. 1139–1147.
28. Edunov, S., Baevski, A., & Auli, M. (2019). Pre-trained language model representations for language generation. North American Chapter of the Association for Computational Linguistics, pp. 1–8. DOI: https://doi.org/10.18653/v1%2FN19-1409 (date of access: 17.11.2024).
29. Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems, pp. 1–43. arXiv preprint arXiv:2005.01643.
30. Potts, C., Wu, Z., Geiger, A., & Kiela, D. (2021). DynaSent: A Dynamic Benchmark for Sentiment Analysis. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics, pp. 2388–2404.

DOI: https://doi.org/10.18653/v1%2F2021.acl-long.186 (date of access: 17.11.2024).

31. Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., & Ng, A. Y. (2013). Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In Conference on Empirical Methods in Natural Language Processing, pp. 1–12.
32. Zendesk AI Platform, Інтелектуальна платформа для автоматизації обслуговування клієнтів [Електронне посилання]. – URL: https://www.zendesk.com/service/ai/ (Дата звернення 17.11.2024).
33. Intercom Resolution Bot, Платформа для обробки клієнтських запитів з використанням штучного інтелекту [Електронне посилання]. – URL: https://bit.ly/3Oy72eh (Дата звернення 17.11.2024).
34. Freshdesk's Freddy AI, Система автоматизації підтримки клієнтів на базі штучного інтелекту [Електронне посилання]. – URL: https://www.freshworks.com/platform/freddy-ai/ (Дата звернення 17.11.2024).
35. HubSpot Service Hub, Комплексна платформа для управління обслуговуванням клієнтів [Електронне посилання]. – URL: https://www.hubspot.com/products/service (Дата звернення 17.11.2024).
36. Salesforce Einstein, Інтелектуальна платформа для автоматизації бізнес-процесів [Електронне посилання]. – URL: https://www.salesforce.com/products/einstein/overview/ (Дата звернення 17.11.2024).