МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ОВРАМЕЦЬ ІЛЛЯ ВОЛОДИМИРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2025 р.

АВТОМАТИЗОВАНА СИСТЕМА ГЕНЕРАЦІЇ ВІДПОВІДЕЙ НА ПОВІДОМЛЕННЯ НА ОСНОВІ ІСТОРІЇ ПЕРЕПИСКИ

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
Юрій АНТОНОВ, доцент кафедри
інформаційних технологій,
к. фіз.– мат. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2025

АНОТАЦІЯ

Оврамець І. В. Автоматизована система генерації відповідей на повідомлення на основі історії переписки. Спеціальність 122 «Комп'ютерні науки», Освітня програма «Комп'ютерні технології обробки даних». Донецький національний університет імені Василя Стуса, Вінниця, 2025.

У кваліфікаційній (бакалаврській) роботі розроблено автоматизовану систему, що забезпечує генерацію відповідей на повідомлення на основі історії переписки користувача. Основною метою роботи є підвищення ефективності комунікації шляхом часткової автоматизації процесу листування. Система реалізована мовою програмування Python та розгорнута у Docker – контейнері, що забезпечує портативність і простоту розгортання. Для взаємодії з зовнішніми сервісами передбачено REST API, який дозволяє інтеграцію системи з іншими програмними рішеннями та клієнтськими застосунками.

Ключові слова: Docker, Python, Django, REST API.

73 с., 3 рис., 60 джерел.

ABSTRACT

Ovramets I. V. Automated System for Generating Message Replies Based on Chat History. Specialty 122 «Computer Science», Programme «Computer technologies for data processing». Vasyl Stus Donetsk National University, Vinnytsia, 2025.

The bachelor's qualification paper presents the development of an automated system designed to generate message replies based on a user's chat history. The main goal of the work is to improve communication efficiency by partially automating the correspondence process. The system is implemented using the Python programming language and deployed within a Docker container, which ensures portability and ease of deployment. For interaction with external services, a REST API is provided, enabling integration with other software systems and client applications.

Keywords: Docker, Python, Django, REST API.

73 p., 3 fig., 60 source.

ЗМІСТ

ВСТУП

У сучасному світі цифрових комунікацій обсяг інформації, що передається через месенджери, електронну пошту та соціальні мережі, зростає у геометричній прогресії [1]. Щодня користувачі обмінюються мільярдами повідомлень, значна частина яких вимагає швидких та змістовних відповідей. У таких умовах автоматизація процесу відповіді на повідомлення стає не лише актуальним, а й необхідним напрямом розвитку інформаційних систем.

Завдяки сучасним методам машинного навчання, обробки природної мови (NLP) [2, 3] та глибокого навчання, з'явилася можливість створювати інтелектуальні системи, здатні аналізувати контекст діалогу, історію переписки та формувати релевантні відповіді з урахуванням стилю та змісту попередніх повідомлень. Такі системи можуть істотно підвищити ефективність комунікацій -як у бізнес-середовищі, так і в особистому користуванні.

Особливої ваги це набуває в умовах інформаційного перевантаження, коли людина фізично не встигає відповідати на всі повідомлення. Автоматизовані системи генерації відповідей дозволяють економити час, покращують якість обслуговування клієнтів у чатах підтримки, сприяють оперативності в бізнес-комунікаціях і навіть можуть адаптуватися під індивідуальний стиль користувача.

Крім того, у контексті сучасних тенденцій розвитку штучного інтелекту важливим є створення таких систем, які можуть навчатися на основі попередньої історії переписки. Це забезпечує більш персоналізований підхід до спілкування та підвищує точність і природність сформованих відповідей.

Актуальність теми: Зростання обсягів цифрової комунікації, розвиток штучного інтелекту та необхідність підвищення ефективності взаємодії між користувачем і системою роблять розробку автоматизованої системи генерації відповідей на основі історії переписки надзвичайно актуальною. Така система здатна зменшити навантаження на користувачів, підвищити рівень автоматизації

цифрових процесів і забезпечити більш природну взаємодію між людиною та комп'ютером.

Мета дослідження є реалізувати прототип системи, генерації відповідей на повідомлення на основі історії переписки з використанням сучасних технологій обробки природної мови та машинного навчання.

Завдання дослідження:

- Проаналізувати автоматизацію комунікаційних систем;
- Сформувати технічне завдання;
- Проаналізувати доступні технології та підходи;
- Реалізувати прототип системи, що автоматично формує відповіді на повідомлення.

Об'єкт дослідження є процес автоматизації комунікацій за допомогою системи генерації відповідей на повідомлення.

Предмет дослідження є процес аналізу тексту.

Наукова новизна полякає в використанні модульної архітектури системи автоматичної генерації відповідей, у якій механізми аналізу тексту та генерації повідомлень виділено в автономний сервіс. Це забезпечує незалежність логіки генерації від основної інфраструктури та можливість використання системи як окремого мікросервісу у будь-яких комунікаційних платформах.

Практичне значення дослідження полягає у створенні прототипу автоматизованої системи генерації відповідей, який може бути інтегрований у реальні комунікаційні платформи, корпоративні месенджери та сервіси підтримки користувачів. Розроблена модульна архітектура забезпечує можливість розгортання системи як окремого мікросервісу, що спрощує масштабування, оновлення та впровадження у різні програмні середовища.

Апробація результатів дослідження: результати досліджень апробовані на VI Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених, м. Вінниця, 8 грудня 2025 р. під назвою «Автоматизована система генерації відповідей на повідомлення з використанням Docker контейнерів», IV Міжнародної науково-практичної конференції, м. Вінниця: 05

листопада 2025 р., ДонНУ імені Василя Стуса, 2025. під назвою «Особливості архітектурної реалізації автоматизованої системи генерації відповідей на повідомлення», III Міжнародної науково-практичної конференції, м. Вінниця: 01 листопада 2024 р., ДонНУ імені Василя Стуса під назвою «Використання штучного інтелекту в комп'ютерних іграх».

Структура роботи: кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел.

Робота містить 73 сторінки, 3 рисунки та список літератури з 60 джерела.

РОЗДІЛ 1
ОГЛЯД АВТОМАТИЗАЦІЇ КОМУНІКАЦІЙНИХ СИСТЕМ

1.1 Актуальність автоматизації комунікаційних систем

У сучасному інформаційному суспільстві комунікація стала одним із ключових чинників ефективного функціонування як окремих користувачів, так і організацій. З розвитком інтернет-технологій та поширенням цифрових засобів спілкування з'явилася величезна кількість каналів комунікації – електронна пошта, месенджери, соціальні мережі, корпоративні чати, системи підтримки клієнтів тощо [1]. З одного боку, це забезпечує швидкість обміну інформацією, а з іншого – призводить до перевантаження користувачів величезним потоком повідомлень, які потребують своєчасної та змістовної відповіді.

Рисунок 1.1 Статистика обміну повідомленнями станом на 2024 рік [4]

У таких умовах зростає потреба в автоматизації процесів обробки та аналізу повідомлень. Традиційні підходи, коли користувач особисто відповідає на кожне звернення, стають неефективними, особливо у великих організаціях чи сервісах, де обсяг комунікацій обчислюється тисячами запитів щоденно. Саме тому все більшої популярності набувають системи, здатні автоматично генерувати відповіді на основі контексту розмови та попередньої історії переписки.

Використання автоматизованих комунікаційних систем дозволяє не лише зменшити навантаження на персонал, а й забезпечити більш стабільну якість обслуговування користувачів. Наприклад, у сфері технічної підтримки автоматизовані системи можуть оперативно відповідати на типові запитання, а в корпоративних середовищах – підтримувати внутрішню комунікацію між працівниками, генеруючи рекомендації або відповіді за шаблоном, адаптованим до конкретного контексту.

Додатковим чинником актуальності є розвиток штучного інтелекту (ШІ) та технологій обробки природної мови (Natural Language Processing, NLP)[5, 6, 7], які зробили можливим розуміння смислового контексту повідомлень, а не лише їх поверхневу обробку. Завдяки цьому сучасні системи можуть не просто підставляти заготовлені фрази, а формувати динамічні, осмислені відповіді, враховуючи стиль, тон і зміст попередніх повідомлень користувача.

Крім того, у сучасному світі зростає значення персоналізації комунікацій. Користувачі очікують, що системи спілкування розумітимуть контекст, пам’ятатимуть попередні взаємодії та реагуватимуть відповідно до їхніх уподобань. Автоматизовані системи, що використовують алгоритми машинного навчання, здатні накопичувати знання про стиль спілкування конкретного користувача, адаптуючи відповіді під його манеру поведінки [8, 9].

Не менш важливою складовою є економічний аспект. В умовах ринку компанії прагнуть скорочувати витрати на підтримку клієнтів і водночас підвищувати рівень їхнього задоволення. Автоматизовані комунікаційні системи дозволяють значно оптимізувати ці процеси, забезпечуючи цілодобову доступність сервісу без необхідності залучення великої кількості персоналу.

У контексті сучасних подій, зокрема військових дій в Україні, важливим аспектом стає також безпека та надійність систем комунікації. Автоматизація відповіді на повідомлення може зменшити ризики людського фактору, який часто стає причиною витоку або неправильного трактування конфіденційної інформації.

Таким чином, актуальність автоматизації комунікаційних систем зумовлена низкою чинників – стрімким ростом обсягів цифрових комунікацій, потребою у швидкому реагуванні на повідомлення, розвитком технологій штучного інтелекту, а також вимогами до персоналізації, безпеки та ефективності інформаційних процесів. Створення системи, здатної автоматично генерувати відповіді на основі історії переписки, є логічним кроком у напрямку підвищення інтелектуальності та зручності сучасних засобів комунікації [10, 11].

1.2 Огляд систем автоматизованих відповідей

1.2.1 Чат-боти та системи підтримки користувачів

Одним із найпоширеніших напрямів автоматизації комунікацій є використання чат-ботів – програмних агентів, здатних імітувати людське спілкування в режимі реального часу. Вони застосовуються у різних сферах: від обслуговування клієнтів і технічної підтримки до навчання, медицини, фінансів, електронної комерції та внутрішніх корпоративних систем. Основна мета чат-ботів – забезпечити користувача швидкою, зручною та автоматизованою взаємодією без участі людини – оператора [12, 13].

Ранні покоління чат-ботів були побудовані на основі фіксованих сценаріїв (rule-based systems). Їхня робота зводилася до пошуку ключових слів у повідомленні користувача та вибору заздалегідь підготовленої відповіді. Такі системи мали обмежений функціонал і не могли ефективно розуміти природну мову або контекст діалогу. Прикладом таких ботів є ELIZA (1966 р.) [14] – один із перших чат-ботів, який імітував поведінку психотерапевта шляхом підстановки шаблонів фраз.

Сучасні чат-боти базуються на технологіях обробки природної мови (NLP – Natural Language Processing), машинного навчання (Machine Learning) та нейронних мереж. Завдяки цим підходам вони здатні не лише реагувати на ключові слова, а й аналізувати семантику повідомлення, розуміти контекст, визначати інтенцію користувача (intent recognition) та формувати релевантні відповіді.

Розвиток систем підтримки користувачів безпосередньо пов'язаний із впровадженням таких ботів у бізнес– процеси. Автоматизація обслуговування клієнтів дозволяє значно скоротити витрати на персонал, підвищити швидкість реакції на запити та зменшити людський фактор. Наприклад, чат-боти у сфері електронної комерції можуть відповідати на типові питання про замовлення, статус доставки чи політику повернення, а у фінансовій сфері – надавати користувачеві інформацію про баланс, останні транзакції або тарифи.

Багато сучасних компаній інтегрують чат– боти в популярні платформи – Telegram, Facebook Messenger, Slack, Microsoft Teams, WhatsApp, а також на власні вебсайти через веб– інтерфейси. Такі системи часто поєднують базові сценарії спілкування з елементами штучного інтелекту, що дозволяє досягати більшої гнучкості та персоналізації.

Важливим напрямом розвитку чат-ботів є гібридна модель, у якій автоматизована система виконує основну частину обробки запитів, але у складних або неоднозначних випадках передає діалог оператору. Такий підхід широко використовується у службах підтримки великих компаній, оскільки поєднує швидкість і масштабованість автоматизації з експертністю людини.

Крім того, сучасні чат– боти мають функцію контекстної пам'яті, яка дозволяє їм враховувати попередні повідомлення користувача для формування послідовної розмови. Це забезпечує більш природну взаємодію та створює відчуття діалогу з "розумною" системою, здатною вести осмислену бесіду.

Ще одним важливим аспектом є персоналізація. Системи підтримки користувачів усе частіше інтегруються з базами даних клієнтів, що дає змогу ботам пропонувати індивідуальні рішення, орієнтовані на історію взаємодії, попередні запити або навіть уподобання користувача.

Завдяки інтеграції технологій глибокого навчання (Deep Learning), особливо моделей типу Transformer (наприклад, GPT, BERT, LLaMA, Claude), чат-боти отримали здатність не лише реагувати на запити, але й генерувати зв'язний, логічний та граматично правильний текст, що відкриває новий рівень автоматизації комунікацій тож розгляньмо їх детальніше.

GPT (Generative Pre-trained Transformer)

Моделі сімейства GPT є одними з найбільш відомих генеративних моделей, побудованих на архітектурі трансформера. Вони працюють за принципом autoregressive language modeling, тобто генерують текст по одному токену, прогнозуючи наступний на основі попереднього контексту. Завдяки цьому GPT чудово підходить для ведення діалогів, створення узгоджених відповідей, контекстної генерації текстів, резюмування й творчих задач. GPT-моделі здатні підтримувати довгі діалоги, адаптувати стиль відповідей та враховувати історію повідомлень. Їхня універсальність робить GPT оптимальним вибором для чат-ботів, систем підтримки, голосових асистентів і автоматичних консультантів.

BERT (Bidirectional Encoder Representations from Transformers)

На відміну від GPT, модель BERT не є генеративною, а виконує двонаправлений аналіз тексту. Вона розглядає слова як у прямому, так і в зворотному напрямку, що дозволяє їй глибше розуміти контекст і значення фраз. BERT особливо ефективний у задачах класифікації тексту, визначення намірів користувача (intent detection), виділення сутностей (NER) та виявлення тональності повідомлень. Його використання у чат-ботах дозволяє значно покращити якість розпізнавання запитів, що є критично важливим для коректної генерації відповідей. Проте для створення відповідей BERT потребує поєднання з іншими моделями або механізмами генерації.

LLaMA (Large Language Model Meta AI)

Моделі LLaMA, розроблені Meta AI, є відкритими генеративними моделями, які забезпечують високу якість тексту при значно менших вимогах до апаратних ресурсів порівняно з традиційними GPT-моделями. LLaMA також базується на трансформерній архітектурі та може бути адаптована до локального або корпоративного використання, що робить її привабливою для систем, де важлива конфіденційність даних. Завдяки оптимізації моделі LLaMA демонструють хорошу здатність до ведення діалогів, узагальнення інформації,

контекстної генерації та роботи з довгими історіями переписки. Вони часто використовуються в автономних чат-ботах, які працюють без хмарних сервісів.

Claude (Anthropic Claude Model)

Claude – це трансформерна модель, створена компанією Anthropic і орієнтована на безпечну, стабільну та передбачувану генерацію тексту. Claude вирізняється високою здатністю працювати з надвеликими контекстами, інколи до сотень тисяч токенів, що робить її надзвичайно ефективною для аналізу або ведення довгих діалогів. Модель розроблена на принципах "constitutional AI", які передбачають зменшення упередженості та небажаних відповідей, тому Claude часто використовується у корпоративних чат-ботах, навчальних платформах, системах підтримки та інформаційних сервісах. Завдяки точності й послідовності, Claude демонструє одну з найвищих якостей генерації тексту серед сучасних моделей.

Отже, чат-боти та системи підтримки користувачів сьогодні є невід'ємною частиною цифрового середовища. Вони демонструють значний потенціал у напрямку оптимізації комунікаційних процесів, підвищення ефективності взаємодії та зменшення навантаження на персонал. Надалі ці технології розвиватимуться у напрямку глибшої контекстної обробки, багатомовності, інтеграції з нейронними моделями та підвищення рівня персоналізації користувацького досвіду.

1.2.2 Використання штучного інтелекту у генерації відповідей

З розвитком технологій штучного інтелекту (ШІ) можливості автоматизованих комунікаційних систем вийшли далеко за межі простих сценаріїв взаємодії. Сучасні системи здатні не лише реагувати на заздалегідь визначені фрази, а й самостійно формувати змістовні, контекстуально доречні відповіді, спираючись на історію переписки та стиль спілкування користувача. Цей підхід ґрунтується на застосуванні моделей машинного навчання, нейронних мереж і обробки природної мови (Natural Language Processing, NLP).

Основна ідея таких систем полягає у створенні моделі, яка може аналізувати текст повідомлення, розуміти його зміст, емоційне забарвлення, намір користувача (intent) і, виходячи з цього, генерувати відповідь, максимально схожу на людську. На відміну від традиційних чат-ботів, що працюють за наперед визначеними сценаріями, інтелектуальні системи здатні адаптуватися до контексту, враховувати попередні повідомлення та динаміку розмови.

Одним із ключових етапів у розвитку таких технологій стало створення мовних моделей на основі архітектури Transformer, серед яких найвідомішими є GPT (Generative Pre – trained Transformer), BERT (Bidirectional Encoder Representations from Transformers), T5 (Text – to – Text Transfer Transformer), LLaMA (Large Language Model Meta AI), Claude, Gemini тощо. Ці моделі пройшли попереднє навчання на великих корпусах текстів і здатні розуміти граматику, семантику та навіть логічні зв'язки між реченнями [15, 16].

Такі системи, як ChatGPT від OpenAI, Google Bard (нині Gemini) або Claude від Anthropic, продемонстрували новий рівень якості автоматичної генерації відповідей. Вони не лише підтримують зв'язний діалог, а й можуть пояснювати, узагальнювати інформацію, адаптувати стиль спілкування під користувача, перекладати тексти та виконувати аналітичні завдання. Цей рівень гнучкості зробив моделі штучного інтелекту ефективними інструментами у сфері автоматизованої комунікації.

Застосування ШІ для генерації відповідей має низку переваг [17]:

- Контекстна обізнаність: система може враховувати попередні повідомлення й реагувати логічно в межах діалогу;
- Персоналізація: відповіді формуються з урахуванням стилю, лексики та поведінкових особливостей конкретного користувача;
- Масштабованість: один алгоритм може одночасно обслуговувати тисячі користувачів без втрати якості комунікації;

- Навчання на основі досвіду: система вдосконалюється в процесі використання, з кожною взаємодією підвищуючи точність і природність відповіді.

Особливу роль у таких системах відіграє аналіз історії переписки. На відміну від одноразових запитів, історія дозволяє побудувати контекст спілкування, визначити тему, тональність та мету розмови. Саме це є фундаментом для створення автоматизованої системи генерації відповідей на основі історії переписки, яка не просто реагує на поточне повідомлення, а розуміє увесь попередній контекст і формує осмислену, логічну та релевантну відповідь.

Крім того, сучасні системи можуть використовувати гібридний підхід, який поєднує генеративні моделі з базами знань або спеціалізованими алгоритмами пошуку інформації. Наприклад, модель може не лише згенерувати текст, але й звернутися до бази даних чи API, щоб уточнити фактичні відомості. Такий підхід суттєво підвищує точність і корисність сформованих відповідей [18].

Однак, попри високий рівень розвитку, застосування ШІ у комунікаційних системах супроводжується низкою викликів. Серед них – забезпечення безпеки даних, запобігання витоку конфіденційної інформації, етичність використання штучного інтелекту та контроль якості згенерованих повідомлень. Тому розробка подібних систем потребує комплексного підходу, що поєднує інтелектуальні алгоритми з архітектурними рішеннями, орієнтованими на безпеку, масштабованість та стабільність роботи [19, 20].

Таким чином, використання штучного інтелекту у генерації відповідей є ключовим етапом розвитку сучасних комунікаційних систем. Воно забезпечує не лише автоматизацію, а й інтелектуалізацію процесу спілкування, наближаючи взаємодію користувача з комп’ютером до природної людської розмови. Це відкриває нові можливості для створення систем, здатних ефективно аналізувати історію переписки та формувати змістовні відповіді.

1.3 Технології обробки природної мови (NLP).

Обробка природної мови (англ. Natural Language Processing, скорочено NLP) є одним із ключових напрямів штучного інтелекту, що забезпечує взаємодію людини з комп'ютером за допомогою природної мови. Основна мета NLP полягає у тому, щоб надати комп'ютерним системам здатність розуміти, інтерпретувати та генерувати людську мову в різних контекстах. Саме завдяки цим технологіям стало можливим створення сучасних інтелектуальних чат-ботів, систем автоматичного перекладу, голосових асистентів та платформ підтримки користувачів [3, 21].

Процес обробки природної мови включає кілька послідовних етапів. Спочатку здійснюється токенізація, тобто розбиття тексту на окремі слова або фрази. Далі виконується лематизація або стемінг, які дозволяють звести слова до їх базової форми. Наступним кроком є синтаксичний аналіз, що полягає у виявленні граматичних зв'язків між словами, після чого проводиться семантичний аналіз, який дозволяє системі зрозуміти зміст повідомлення. Завершальним етапом є визначення наміру користувача (intent detection) та генерація відповіді (response generation), де система формує релевантну та логічно пов'язану відповідь.

Для реалізації зазначених етапів використовуються різні інструменти та бібліотеки. Найпоширенішими є NLTK (Natural Language Toolkit) – набір інструментів для базового аналізу тексту, spaCy – продуктивна бібліотека для лінгвістичної обробки, Stanford NLP – потужний інструментарій для синтаксичного розбору, а також сучасні бібліотеки на базі трансформерних моделей, такі як BERT, RoBERTa, GPT, та фреймворки Hugging Face Transformers і LangChain. Вони забезпечують глибоке контекстне розуміння тексту та використовуються для генерації осмислених відповідей у системах діалогів розгляньмо його переваги згідно джерел [22, 23].

Переваги застосування технологій NLP:

Використання методів обробки природної мови має низку суттєвих переваг, які визначають їхню актуальність у сучасних інформаційних системах,

зокрема у сфері автоматизації комунікацій. Насамперед, NLP дозволяє значно підвищити ефективність обробки текстової інформації. Комп'ютерні системи здатні аналізувати великі обсяги даних у реальному часі, виконуючи такі операції, на які людині знадобилося б значно більше часу. Це робить NLP незамінним у сфері обслуговування клієнтів, моніторингу соціальних мереж, автоматизації внутрішніх комунікацій в організаціях та обробці повідомлень у цифрових сервісах.

Другим важливим плюсом є стабільність та передбачуваність роботи NLP-систем, оскільки вони не залежать від людського фактора – втоми, емоційного стану чи суб'єктивних рішень. Алгоритми забезпечують однакову якість відповідей за будь-яких умов, що критично важливо для сервісів підтримки користувачів та чат-ботів, які обслуговують тисячі звернень на добу.

Ще однією суттєвою перевагою є можливість масштабування. У той час як кількість операторів у кол-центрі або службі підтримки обмежена, системи NLP можуть одночасно обробляти практично необмежену кількість запитів. Це особливо актуально для бізнесів, які працюють у режимі високого навантаження або мають міжнародну аудиторію.

Важливим є й те, що NLP дозволяє проводити глибший аналіз тексту, включно з виділенням емоційного забарвлення повідомлень, намірів користувача, рівня терміновості та можливих ризиків. Наприклад, за допомогою методів sentiment analysis система може визначити, чи є повідомлення негативним, нейтральним або позитивним, і відповідно адаптувати відповідь. Це значно покращує якість взаємодії та наближує комунікацію до живого, людського спілкування.

Не менш важливою перевагою є підвищення якості генерації відповідей. Сучасні мовні моделі здатні будувати зв'язні, граматично правильні та змістовні відповіді, враховуючи контекст усієї розмови, а не лише останнього повідомлення. Такі системи здатні підтримувати довгі діалоги, пам'ятати попередні репліки та адаптувати стиль спілкування до конкретного користувача. Це є ключовою особливістю для систем, які працюють з історією переписки,

виконуючи аналіз попередніх повідомлень і формуючи логічно узгоджену реакцію.

Ще однією важливою перевагою є здатність NLP-систем до самоудосконалення. Багато сучасних моделей навчаються на основі зворотного зв'язку, що дозволяє підвищувати точність прогнозів та якість відповідей у процесі експлуатації. Такий підхід забезпечує безперервне покращення роботи системи та її адаптацію до нових сценаріїв використання.

Практичні приклади застосування NLP:

Завдяки переліченим перевагам технології обробки природної мови стали основою для широкого спектра прикладних рішень. У сфері бізнес-комунікацій вони використовуються для побудови розумних чат-ботів, які здатні відповідати на типові запитання, надавати технічну підтримку, здійснювати первинну діагностику проблем та перенаправляти складніші запити оператору.

У банківській сфері NLP застосовується для автоматичного створення звітів, аналізу електронних листів клієнтів, виявлення шахрайських транзакцій на основі аналізу текстових описів та чату з клієнтом. У медицині ці технології дозволяють автоматично обробляти медичні записи, підтримувати лікарів у діагностиці та відповідати на запити пацієнтів щодо симптомів чи процедур.

У державному секторі NLP використовується для аналізу звернень громадян, класифікації документів, моніторингу інформаційного простору та виявлення загроз у системах кібербезпеки. Окрему роль ці технології відіграють у сфері освіти, де чат-боти забезпечують підтримку студентів, відповідають на організаційні питання та формують індивідуальні рекомендації щодо навчального процесу.

У контексті автоматизованих систем комунікації NLP відіграє вирішальну роль. Технології дозволяють не лише класифікувати запити користувачів і розпізнавати контекст повідомлення, а й враховувати історію переписки, емоційне забарвлення тексту та індивідуальний стиль спілкування. Завдяки цьому сучасні чат–боти та системи підтримки можуть формувати більш природні

та адаптивні відповіді, наближені до людського спілкування. Тож розгляньмо його переваги в нашій системі.

Переваги NLP у системах генерації відповідей на основі історії переписки.

Для автоматизованої системи генерації відповідей, яка працює з історією повідомлень, NLP є критично важливим елементом. По-перше, ця технологія дає змогу коректно інтерпретувати контекст повідомлень, що є ключовим у діалогових системах. Без урахування попередніх повідомлень відповідь може бути некоректною, надто загальною або такою, що не відповідає потребам користувача.

По-друге, NLP забезпечує врахування стилістики та тону переписки. Система може ідентифікувати емоційний стан співрозмовника, розрізняти формальний і неформальний стиль, виявляти сарказм або приховані наміри. Це робить взаємодію значно природнішою та ефективнішою.

По-третє, завдяки методам семантичного аналізу та векторного представлення тексту (embeddings) система може визначати приховані зв'язки між повідомленнями, навіть якщо вони не мають очевидної ключової лексики. Це дозволяє об'єднувати фрази за змістом і робить генерацію відповіді набагато точнішою.

По-четверте, сучасні NLP-моделі здатні виявляти наміри користувачів, що особливо важливо для автоматизованих сервісів підтримки. Наприклад, система може розпізнати намір «отримати довідку», «скасувати замовлення» або «дізнатися статус заявки», навіть якщо користувач формулює запит неявно.

Таким чином, NLP забезпечує основу для побудови інтелектуальних систем комунікації, здатних адаптуватися до поведінки користувача, відповідати більш природно та забезпечувати високий рівень якості діалогової взаємодії.

Подальший розвиток NLP пов'язаний із використанням великих мовних моделей (Large Language Models, LLM) [24], які демонструють здатність до глибокого розуміння контексту, узагальнення інформації та прогнозування наступних фраз у діалозі. Ці моделі забезпечують основу для створення систем, здатних вести тривалі діалоги, запам'ятовувати попередні повідомлення та

адаптуватися до потреб користувача. Таким чином, технології NLP є фундаментом для розробки автоматизованої системи генерації відповідей на повідомлення на основі історії переписки, оскільки саме вони забезпечують можливість аналізу, розуміння та побудови змістовної реакції в процесі спілкування.

1.4 Моделі машинного навчання для аналізу тексту.

Машинне навчання є однією з ключових технологій, що забезпечує можливість комп'ютерам самостійно виявляти закономірності у даних та приймати рішення без явного програмування. У сфері обробки природної мови моделі машинного навчання застосовуються для розпізнавання структури тексту, класифікації повідомлень, визначення емоційного тону, виявлення ключових слів та формування осмислених відповідей [25]. Саме завдяки використанню таких моделей сучасні системи можуть ефективно аналізувати велику кількість текстової інформації та адаптуватися до стилю спілкування користувача.

На початкових етапах розвитку текстової аналітики використовувалися традиційні статистичні методи, такі як наївний байєсівський класифікатор (Naive Bayes), метод опорних векторів (SVM), логістична регресія та дерева рішень. Вони добре підходили для простих завдань – наприклад, класифікації електронних листів на спам і не спам, або визначення теми повідомлення. Проте такі підходи мали обмежену здатність враховувати контекст і семантичні зв'язки між словами, що знижувало їх ефективність у складних лінгвістичних задачах.

Подальший розвиток технологій привів до широкого впровадження нейронних мереж, які здатні моделювати складні нелінійні залежності в даних. Зокрема, рекурентні нейронні мережі (RNN) та їхні модифікації, такі як LSTM (Long Short– Term Memory) і GRU (Gated Recurrent Unit), стали основою для аналізу послідовних текстових даних. Вони дозволяють враховувати порядок слів у реченні, що є надзвичайно важливим для коректного розуміння змісту повідомлення. Наприклад, саме ці архітектури використовувалися у перших

інтелектуальних чат– ботах для прогнозування наступних слів та побудови зв'язних речень [26].

Наступним кроком еволюції стало впровадження трансформерних моделей, які змінили підхід до аналізу тексту. На відміну від рекурентних мереж, трансформери використовують механізм уваги (attention mechanism), що дозволяє моделі одночасно враховувати всі слова у реченні та визначати, які з них найбільш впливають на значення поточного фрагмента. Цей підхід дав змогу суттєво підвищити точність розуміння контексту та зробив можливим створення потужних мовних моделей, таких як BERT (Bidirectional Encoder Representations from Transformers), RoBERTa, T5, GPT (Generative Pre– trained Transformer) тощо.

Моделі типу BERT використовуються переважно для завдань аналізу тексту – класифікації, вилучення сутностей, оцінки схожості тощо. Натомість GPT– моделі, орієнтовані на генерацію, застосовуються у чат– ботах, системах автоматичної відповіді, написанні текстів і навіть у кодуванні. Вони здатні розуміти довготривалий контекст, будувати зв'язні діалоги та адаптувати стиль спілкування до конкретного користувача.

Сучасні дослідження у сфері машинного навчання спрямовані на створення мультимодальних моделей, які обробляють не лише текст, а й зображення, аудіо або відео, що відкриває нові можливості для розширення функціональності систем спілкування. Крім того, активний розвиток отримали методи тонкого донавчання (fine– tuning), які дозволяють адаптувати великі мовні моделі під конкретні задачі – наприклад, генерацію відповідей у межах історії переписки користувачів [26, 27].

Отже, моделі машинного навчання є основою для побудови інтелектуальних систем аналізу тексту. Вони забезпечують можливість глибокого розуміння змісту повідомлень, урахування контексту спілкування та формування релевантних відповідей. У поєднанні з технологіями NLP такі моделі створюють підґрунтя для розробки автоматизованих систем генерації відповідей на основі історії переписки, що є актуальним напрямом розвитку сучасних комунікаційних платформ.

1.5 Інтеграція моделей з месенджерами та вебсервісами.

Інтеграція моделей штучного інтелекту з месенджерами та вебсервісами є ключовим етапом у створенні сучасних систем автоматизованого спілкування. Вона забезпечує можливість практичного застосування алгоритмів обробки природної мови та машинного навчання безпосередньо в середовищі, де відбувається взаємодія користувача з системою. Такий підхід дозволяє не лише підвищити ефективність комунікації, а й забезпечити персоналізовану взаємодію з урахуванням контексту переписки та історії повідомлень.

Сучасні месенджери, такі як Telegram, WhatsApp, Facebook Messenger, Slack чи Microsoft Teams, надають відкриті API (Application Programming Interface), за допомогою яких можна інтегрувати зовнішні сервіси або інтелектуальні модулі. Через API система отримує повідомлення користувача, надсилає їх на обробку до моделі машинного навчання або NLP– сервісу, після чого повертає сформовану відповідь назад у чат. Така архітектура забезпечує гнучкість, масштабованість і можливість підключення різних моделей без необхідності втручання у роботу самого месенджера.

Для забезпечення стабільної взаємодії між клієнтською частиною (інтерфейсом користувача) та серверною частиною (де відбувається обробка даних) використовуються вебхуки (webhooks) або REST API – запити [28]. У більш складних системах застосовується багаторівнева архітектура, що включає сервер обробки повідомлень, модуль логіки діалогу, базу даних для збереження історії спілкування, а також модель штучного інтелекту для аналізу і генерації відповідей.

Інтеграція моделей з вебсервісами передбачає використання хмарних платформ і спеціалізованих інструментів, які дозволяють масштабувати обчислення. Наприклад, платформи Google Cloud AI, Microsoft Azure Cognitive Services, AWS AI Services, Hugging Face Inference API або OpenAI API надають готові середовища для розгортання мовних моделей, їх навчання, донавчання та використання у реальному часі. Це значно спрощує процес впровадження

штучного інтелекту у вебдодатки, забезпечуючи стабільну продуктивність навіть при великій кількості користувачів [29, 30].

У процесі інтеграції систем штучного інтелекту з комунікаційними платформами важливо враховувати особливості та можливості різних мовних моделей, оскільки саме від них залежить якість взаємодії, швидкість обробки запитів та здатність адаптуватися до контексту діалогу. Сучасні LLM-моделі (Large Language Models), такі як GPT, BERT, RoBERTa, LLaMA, Falcon або Mistral, відрізняються архітектурою, продуктивністю, ресурсомісткістю та точністю виконання завдань, що впливає на їхню ефективність у конкретних сценаріях використання.

До прикладу давайте порівняємо їх між собою:

Однією з ключових відмінностей між моделями є їхнє призначення. Наприклад, моделі сімейства BERT, RoBERTa та їх похідні оптимізовані переважно для аналізу тексту (класифікація, витяг сутностей, визначення наміру користувача). Вони виконують завдання у форматі "розуміння тексту", проте не пристосовані до генерації зв'язних відповідей. Натомість моделі типу GPT, LLaMA або Mistral є генеративними і здатні виконувати як аналіз тексту, так і побудову зв'язних діалогів, що робить їх більш придатними для інтеграції у месенджерні системи, де потрібна генерація відповідей у реальному часі.

Важливим параметром, що впливає на вибір моделі для інтеграції, є обчислювальна складність і вимоги до ресурсів. Так, потужні моделі, що містять десятки мільярдів параметрів, демонструють високу точність, однак вимагають значних апаратних ресурсів, що ускладнює їх використання в локальних системах і на серверних середовищах із обмеженою потужністю. У таких випадках перевага надається моделям середнього розміру, наприклад LLaMA-2-7B або Mistral-7B, які забезпечують компроміс між якістю генерації та продуктивністю системи.

Ще одним аспектом є швидкість генерації відповідей, яка відіграє важливу роль у месенджерах, де користувач очікує негайної реакції. У порівнянні з великими моделями, компактніші моделі забезпечують меншу затримку, що

покращує користувацький досвід. Наприклад, BERT-подібні моделі виконують аналіз тексту майже миттєво, тоді як GPT-моделі потребують більше часу на генерацію, але при цьому формують більш природні та контекстуальні відповіді.

Також суттєвою характеристикою є здатність моделей працювати з довгими контекстами, що є критичним для систем, які використовують історію переписки. Моделі GPT-4, GPT-3.5, Claude, Mistral Large або LLaMA-3 підтримують розширені контексти (від 8 до 200 тисяч токенів), що дозволяє аналізувати тривалі діалоги та формувати узгоджені відповіді. Натомість старі моделі мали суттєві обмеження щодо довжини вхідного тексту, що знижувало якість генерації при довготривалому спілкуванні.

Окрему увагу варто приділити точності адаптації моделі до домену використання. Наприклад, у техпідтримці, медицині чи юридичній сфері важливо забезпечити точність термінології та коректність відповідей. Моделі типу BERT краще виконують задачі класифікації та витяг інформації, тоді як GPT-моделі демонструють вищу гнучкість у генерації текстів, але можуть потребувати додаткового налаштування (fine-tuning) або використання фреймворків на кшталт LangChain для керованої генерації.

З точки зору інтеграції з месенджерами, важливо оцінити підтримку сторонніх API та інструментів розгортання. GPT-моделі (наприклад, OpenAI API) пропонують широкий спектр готових інтерфейсів, що спрощує впровадження у вебсервери або боти Telegram/WhatsApp. Моделі LLaMA чи Mistral часто вимагають локального розгортання або використання спеціалізованих сервісів (Hugging Face, Replicate), але забезпечують більший контроль над даними та можливість повністю автономної роботи.

Таким чином, вибір моделі залежить від конкретних вимог до системи:

- Якщо необхідно швидко аналізувати повідомлення та класифікувати запити, доцільно використовувати моделі типу BERT або RoBERTa;
- Якщо система повинна вести діалог та формувати відповіді, найбільш ефективними є GPT-подібні моделі;

- Якщо важлива автономність та конфіденційність даних – оптимальним буде локальне розгортання моделей LLaMA, Falcon або Mistral.

У результаті порівняльний аналіз дозволяє зробити висновок, що жодна модель не є універсальною. Для інтеграції з месенджерами оптимальним є використання генеративних моделей із підтримкою довгого контексту, тоді як для окремих задач аналізу тексту доцільно застосовувати спеціалізовані моделі. Комбінування цих підходів у межах модульної архітектури забезпечує максимальну гнучкість, масштабованість і якість взаємодії з користувачем.

Важливим аспектом інтеграції є забезпечення безпеки та конфіденційності даних. Оскільки обробка повідомлень часто передбачає роботу з приватною інформацією, необхідно застосовувати протоколи шифрування, автентифікацію користувачів, контроль доступу та анонімізацію текстових даних перед передачею до моделі. У випадках, коли система розгортається локально (on–premise), можливо реалізувати повністю автономну обробку повідомлень без передачі даних у зовнішні хмарні сервіси.

Інтеграція моделей з месенджерами також відкриває можливості для аналітики комунікацій. За допомогою моделей машинного навчання можна відстежувати ефективність взаємодії користувачів, виявляти типові запити, оцінювати рівень задоволення спілкуванням та виявляти проблемні ділянки діалогу. Такі дані можуть бути використані для подальшого донавчання моделей і підвищення якості автоматичних відповідей.

Таким чином, інтеграція моделей обробки природної мови та машинного навчання з месенджерами й вебсервісами є необхідною умовою для створення сучасних інтелектуальних систем комунікації. Вона забезпечує можливість автоматичного аналізу, обробки й генерації відповідей у реальному часі, сприяючи підвищенню ефективності цифрової взаємодії, зручності користування та рівня безпеки переданих даних.

Однак для забезпечення гнучкості, масштабованості та повторного використання розроблених рішень доцільно реалізовувати подібні системи у вигляді окремих модулів. Такий підхід дозволяє ізолювати логіку обробки

повідомлень, алгоритми машинного навчання та механізми інтеграції від основної бізнес– логіки застосунку. Завдяки цьому модуль може бути легко впроваджений у різні платформи – від корпоративних чатів до публічних месенджерів і вебпорталів підтримки клієнтів – без необхідності повної переробки системи.

1.6 Переваги модульної реалізації системи.

Беручи до уваги швидкі темпи розвитку технологій штучного інтелекту, машинного навчання та обробки природної мови, усе більшої актуальності набуває потреба створювати системи, які можна легко адаптувати, розширювати й інтегрувати у вже наявні цифрові платформи.

Модульний підхід дає змогу розробникам забезпечити гнучкість і незалежність окремих компонентів системи. У такій архітектурі кожен модуль виконує свою чітко визначену функцію: один відповідає за обробку вхідних повідомлень, інший – за аналіз контексту діалогу, ще один – за формування відповіді на основі історії переписки та моделей машинного навчання. Це дозволяє ізолювати логіку системи від зовнішніх інтерфейсів і зменшує ризик порушення роботи при внесенні змін до певних частин коду.

Окремий модуль автоматичної генерації відповідей може бути використаний у різних середовищах комунікації: від корпоративних чатів та CRM– систем до мобільних месенджерів і вебплатформ підтримки клієнтів. Такий підхід сприяє створенню універсального рішення, яке може бути легко вбудоване в інші проєкти без необхідності повної перебудови їхньої архітектури. Це, у свою чергу, значно скорочує витрати часу на інтеграцію, спрощує оновлення та забезпечує можливість масштабування в майбутньому.

Ще однією важливою перевагою модульної структури є зручність експериментування та навчання моделей. Оскільки модуль функціонує незалежно, дослідники можуть змінювати алгоритми, додавати нові моделі машинного навчання або розширювати набір даних для тренування без втручання в роботу інших частин системи. Це відкриває простір для безпечного

вдосконалення та тестування нових підходів до генерації відповідей, у тому числі із залученням різних архітектур нейронних мереж [31].

Для забезпечення гнучкості розгортання та ізоляції середовища виконання пропонується розміщення модуля в **Docker– контейнері**. Такий підхід надає низку переваг [32, 33]:

- **Портативність** – модуль можна розгортати на будь– яких серверах чи хмарних платформах незалежно від операційної системи;
- **Масштабованість** – при збільшенні навантаження можливо підняти кілька контейнерів і розподілити запити між ними;
- **Безпечність і стабільність** – контейнери ізолюють середовище виконання, що мінімізує ризик конфліктів версій бібліотек чи втручань ззовні;
- **Зручність інтеграції** – контейнерний модуль можна підключити до будь– якої системи через REST API або інші стандартизовані протоколи.

Модульний підхід також дозволяє забезпечити високу надійність і безпеку обробки даних. Оскільки процес формування відповідей можна винести в окреме середовище, де обмежено доступ до користувацької інформації, це знижує ризики витоку або несанкціонованого використання даних. Крім того, така архітектура полегшує застосування технологій шифрування та контроль доступу до внутрішніх ресурсів системи, що є надзвичайно важливим у сучасних умовах кіберзагроз.

Висновки до першого розділу

У першому розділі було здійснено комплексний огляд сучасних підходів до автоматизації комунікаційних систем, зокрема систем автоматизованих відповідей, чат-ботів та технологій генерації тексту на основі штучного інтелекту. Проведений аналіз підтверджує, що цифровізація та стрімке зростання обсягів інформаційних потоків роблять автоматизацію комунікацій не просто бажаною, а критично необхідною як для бізнесу, так і для державних чи приватних сервісів.

Показано, що традиційні методи обробки користувацьких запитів, які передбачають ручне опрацювання кожного повідомлення, втрачають ефективність у масштабних системах. У цьому контексті актуальним стає впровадження інтелектуальних рішень, здатних адаптивно реагувати на звернення, враховувати контекст та підтримувати персоналізовану комунікацію. Зростання попиту на такі системи додатково обумовлено потребою підвищення швидкості взаємодії, зменшення навантаження на персонал, оптимізації витрат, а також вимогами до безпеки інформації, особливо в умовах сучасних загроз.

Було проаналізовано розвиток чат-ботів – від систем із фіксованими сценаріями до інтелектуальних моделей, що застосовують машинне навчання, глибокі нейронні мережі та архітектуру Transformer. Саме вони забезпечили перехід до якісно нового рівня взаємодії, у якому автоматизована система здатна вести зв'язний діалог, розуміти наміри користувача, зберігати контекст та використовувати персоналізований стиль комунікації.

Окрему увагу приділено використанню технологій обробки природної мови (NLP), які формують основу сучасних систем генерації відповідей. Розглянуто ключові етапи NLP – токенізацію, лематизацію, синтаксичний і семантичний аналіз, визначення інтенції та генерацію тексту. Показано, що поєднання класичних NLP-підходів із великими мовними моделями (LLM) забезпечує можливість створення високоточної, гнучкої та осмисленої автоматизованої комунікації.

Узагальнюючи, можна стверджувати, що автоматизовані комунікаційні системи на основі ШІ стають одним із ключових елементів сучасної цифрової інфраструктури. Вони дають змогу суттєво підвищити якість обслуговування, забезпечити масштабованість, адаптивність та персоналізацію взаємодії з користувачем. Це створює міцне підґрунтя для подальшої розробки системи автоматичної генерації відповідей на основі історії переписки, яка є предметом подальших розділів цієї роботи.

РОЗДІЛ 2
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Постановка задачі

Беручи до уваги зростаючий обсяг цифрових комунікацій та активне використання месенджерів у повсякденному житті, виникає потреба у створенні систем, здатних автоматично формувати релевантні відповіді на повідомлення користувачів. Такий функціонал дозволяє підвищити ефективність взаємодії, скоротити час реакції та створити більш персоналізований досвід спілкування.

Однією із особливостей програми у виділенні **системи рекомендацій та генерації відповідей** в **окремий програмний модуль**, який функціонуватиме незалежно від основного застосунку. Цей модуль реалізує логіку обробки текстових повідомлень, аналізує історію переписки, формує контекстні відповіді та може бути інтегрований у будь-яке клієнт-серверне середовище або зовнішню систему комунікації.

Отже, виокремлення механізму автоматичної генерації відповідей у самостійний модуль не лише підвищує гнучкість та масштабованість системи, а й забезпечує чітке розмежування відповідальностей між її компонентами. Такий підхід дозволяє легко розширювати функціонал, оновлювати модель та інтегрувати нові алгоритми без втручання в інші частини застосунку.

З огляду на це, модульна архітектура постає найбільш ефективним рішенням для побудови інтелектуальної системи генерації відповідей. Структуру цієї архітектури представлено на ілюстрації нижче Рис. 2.1.

Вона поєднує у собі простоту інтеграції, гнучкість масштабування та високу ступінь незалежності компонентів. Завдяки цьому розроблена система може слугувати як окремий продукт або стати складовою частиною ширшої платформи обміну повідомленнями, підтримки користувачів чи корпоративної взаємодії.

Рисунок 2.1 Схема окремого модулю рекомендацій

Таким чином, майбутня система буде містити окремий модуль генерації відповідей. Вимоги до цього модуля сформуєм наступним чином:

1. Підсистему обробки вхідних повідомлень та попереднього аналізу тексту;
2. Компонент машинного навчання на основі вже готових архітектур та рішень, який здійснюватиме вибір або генерацію відповіді на основі історії переписки та контексту;
3. API– інтерфейс для взаємодії з клієнтськими застосунками, що забезпечить інтеграцію модуля в зовнішні сервіси;
4. Середовище контейнеризації (Docker), яке гарантує універсальність і незалежність розгортання.

Реалізація такого рішення створює можливість використання системи як мікросервісу, який може стати частиною більш масштабної архітектури – наприклад, корпоративного месенджера, служби підтримки або платформи онлайн-консультацій. Крім того, модульна структура дозволить розширювати функціонал у майбутньому: додавати нові алгоритми генерації тексту,

інтегрувати інші моделі обробки мови чи адаптувати систему під специфіку різних користувачів або організацій.

2.2 Аналіз вхідних даних: історія переписки

Одним із ключових етапів побудови системи автоматичної генерації відповідей є аналіз вхідних даних, що у даному випадку представлений історією переписки між користувачами. Саме на основі історії спілкування система може відтворювати логіку діалогу, виявляти закономірності у взаємодії та формувати відповіді, максимально наближені до природного людського мовлення.

Для забезпечення повноцінного функціонування модуль повинен мати можливість отримувати дані про попередні повідомлення, незалежно від того, де саме вони зберігаються – у локальній базі даних чи у віддаленій системі. Тому при проєктуванні передбачено два основні механізми отримання вхідних даних:

По-перше, це прямий доступ до бази даних, який використовується у випадках, коли система є частиною єдиної клієнт–серверної архітектури. Такий підхід забезпечує високий рівень швидкодії, оскільки модуль може напряму звертатися до таблиць, у яких зберігаються повідомлення користувачів. Залежно від потреби, запити можуть фільтрувати дані за параметрами – наприклад, ідентифікатором користувача, сеансом спілкування або часовим проміжком. Це дозволяє формувати цілісний контекст діалогу, включаючи послідовність запитів, відповіді системи, часові мітки, роль відправника (користувач чи оператор) та метадані.

По-друге, у разі інтеграції з зовнішніми сервісами або сторонніми месенджерами застосовується отримання даних через REST API [34]. У цьому випадку модуль надсилає HTTP– запити до зовнішнього джерела, а у відповідь отримує структуровані дані – зазвичай у форматі JSON. Такий підхід робить систему більш універсальною та незалежною від конкретної інфраструктури, адже дозволяє легко підключати її до різних платформ без зміни архітектури. Крім того, API-інтерфейс може бути розширений додатковими параметрами, що

дозволяє здійснювати фільтрацію повідомлень на стороні сервера, зменшуючи навантаження на сам модуль.

Для забезпечення безпеки передачі даних REST API підтримує автентифікацію за токенами доступу, що гарантує захист персональної інформації користувачів і запобігає несанкціонованим зверненням до системи. Завдяки цьому підхід із використанням API є більш придатним для масштабованих або розподілених систем, які функціонують у хмарних середовищах.

Після отримання даних із будь-якого з джерел виконується етап попередньої обробки історії переписки, що має на меті перетворити "сирі" тексти у структуровану та придатну до аналізу форму. Цей процес включає кілька послідовних кроків [35]:

- Нормалізацію тексту – видалення спеціальних символів, HTML– тегів, смайлів, повторюваних пробілів тощо;
- Уніфікацію регістру для усунення відмінностей між великими та малими літерами;
- Токенізацію (розбиття речень на окремі слова або фрази);
- Лематизацію або стемінг – приведення слів до початкової форми;
- Вилучення неінформативних елементів, таких як стоп– слова, службові фрази чи технічні повідомлення;

Виділення ключових слів і визначення намірів (intent detection), що допомагає системі розуміти зміст запитів користувача.

У результаті формується структурований набір даних, який описує кожне повідомлення через основні атрибути:

- ідентифікатор діалогу;
- роль автора (користувач, система, оператор);
- час надсилання;
- текст повідомлення після попередньої обробки;
- тематичну або контекстну мітку (якщо така є).

Далі ці дані передаються у модуль аналітики та генерації контексту, який формує репрезентацію поточного діалогу для моделі машинного навчання. Завдяки цьому система може враховувати не лише останнє повідомлення, а й попередні елементи діалогу – тобто працювати з повним контекстом, що істотно підвищує якість і доречність згенерованих відповідей [5, 36].

Важливим аспектом є також адаптивність моделі до різних джерел даних. Незалежно від того, чи отримує система інформацію з локальної бази, чи через API– запит, алгоритм має уніфікований механізм інтерпретації структури діалогу. Це дозволяє забезпечити стабільність роботи та узгодженість результатів навіть при зміні формату вхідних даних.

У підсумку, етап аналізу історії переписки відіграє ключову роль у функціонуванні системи, оскільки саме він формує основу для контекстного сприйняття та послідовної генерації відповідей. Якісна обробка вхідних даних забезпечує не лише технічну стабільність роботи модуля, а й визначає рівень його «інтелектуальності» – здатність сприймати діалог як цілісний контекст.

Важливо також, що система може навчатися на будь– яких наборах даних, які надає користувач, незалежно від джерела їх отримання (файл, база даних, API чи інша структура). Перед використанням ці дані автоматично приводяться до уніфікованого формату, який розуміє модель, наприклад:

```
[{ "instruction": "Як підняти Django сервер?",
  "input": "",
  "output": "Запустіть команду python manage.py runserver..."},
{"instruction": "Проаналізуй код і скажи, де помилка.",
  "input": "def add(a, b):\n    return a -  b",
  "output": "У функції використано оператор віднімання замість
додавання."}]
```

- **instruction** – сама команда/завдання/питання
- **input** – додатковий контекст або дані, які потрібні для виконання завдання
- **output** – відповідь моделі

Завдяки цьому забезпечується єдина логіка інтерпретації діалогів і максимальна сумісність із механізмами навчання модуля. У наступних підрозділах буде розглянуто архітектуру модуля обробки повідомлень,

механізми формування контекстних векторів та інтеграцію цього компонента в загальну систему.

2.3 Архітектурні та проектувальні патерни

Розробка системи безпечного месенджера з модульною структурою та інтегрованою системою рекомендацій вимагає ретельного вибору архітектурних і проектувальних патернів. Від цього залежить не лише ефективність і стабільність роботи застосунку, а й можливість його подальшого масштабування, підтримки та інтеграції з іншими сервісами.

Враховуючи вимоги до масштабованості, ізольованості модулів і підтримки взаємодії з базами даних або зовнішніми API, за основу обрано багаторівневу клієнт-серверну архітектуру з елементами модульності та можливістю контейнеризації окремих компонентів.

Багаторівнева (або трирівнева) архітектура – це структура програмної системи, яка розділяє логіку роботи на окремі рівні: Presentation Layer, Application Layer та Data Layer. Такий поділ дозволяє створити більш організовану, гнучку та безпечну систему, де кожен рівень виконує чітко визначену роль [36, 37]. Розглянемо ці рівні:

Presentation Layer (рівень представлення) – відповідає за взаємодію з користувачем. У випадку з месенджером це може бути вебінтерфейс або мобільний застосунок, який надсилає запити до серверної частини та отримує згенеровані відповіді.

Application Layer (рівень бізнес-логіки) – обробляє запити від клієнта, взаємодіє з моделями машинного навчання, виконує аналіз історії переписки й формує результати.

Data Layer (рівень даних) – відповідає за зберігання, обробку та отримання даних із бази даних або зовнішніх джерел (наприклад, через REST API).

Переваги багаторівневої архітектури:

1. Чітке розділення обов'язків – кожен рівень відповідає лише за свою частину функціоналу, що підвищує зрозумілість і передбачуваність коду

2. Масштабованість – можна розширювати або оптимізувати лише ті рівні, які відчувають навантаження (наприклад, серверний рівень).

3. Безпека – клієнт не має прямого доступу до бази даних, що знижує ризик несанкціонованого втручання.

4. Гнучкість інтеграції – архітектура дозволяє легко підключати нові модулі, наприклад, систему рекомендацій або аналітику повідомлень.

5. Простота тестування та обслуговування – окремі рівні можна перевіряти незалежно один від одного.

Варто зазначити, що клієнтом у цій архітектурі може бути не лише користувач через інтерфейс застосунку, але й інший сервер або сервіс, який використовує REST API системи для генерації повідомлень чи аналітики переписок. Такий підхід дозволяє інтегрувати систему у вже існуючі рішення – наприклад, корпоративні месенджери, CRM–системи або платформи підтримки клієнтів.

Для забезпечення комунікації між клієнтом і сервером використовується REST API (Representational State Transfer Application Programming Interface) – архітектурний стиль, який базується на протоколі HTTP і принципах простоти, масштабованості та незалежності компонентів.

Основні принципи REST [37]:

Клієнт-серверна архітектура – клієнт і сервер розділені, що дозволяє розвивати їх незалежно.

Відсутність стану (Stateless) – кожен запит є самодостатнім і не зберігає інформацію про попередні. Це спрощує масштабування.

Кешування – відповіді сервера можуть бути кешовані, що підвищує ефективність системи.

Єдиний інтерфейс (Uniform Interface) – всі ресурси системи доступні через чітко визначені URL та стандартні методи HTTP.

Шарувана система (Layered System) – REST API може взаємодіяти з проміжними сервісами, наприклад, балансувальниками навантаження чи проксі.

Код за запитом (Code on demand) – за потреби сервер може передавати клієнту виконуваний код (наприклад, скрипти).

REST API використовує стандартні методи HTTP для взаємодії з ресурсами:

- GET – отримання даних (наприклад, історії переписки або поточного статусу користувача).
- POST – створення нового ресурсу, наприклад, надсилання нового повідомлення для генерації відповіді.
- PUT / PATCH – оновлення існуючих ресурсів, наприклад, зміна параметрів користувача або налаштувань моделі.
- DELETE – видалення ресурсу, наприклад, видалення певного діалогу або сесії.

Оскільки система обробляє особисту комунікацію користувачів, безпека API є ключовим аспектом. Для цього реалізуються такі механізми:

Аутентифікація та авторизація за допомогою токенів (наприклад, JWT – JSON Web Token), які забезпечують безпечний доступ до ресурсів.

Шифрування трафіку через HTTPS, що унеможливлює перехоплення або модифікацію даних.

Обмеження швидкості запитів (**Rate Limiting**) для запобігання атакам типу DoS.

Валідація запитів на сервері для перевірки коректності даних, отриманих від клієнтів.

У контексті даної системи REST API використовується для:

1. Отримання історії переписки з бази даних або зовнішніх джерел;
2. Надсилання текстових повідомлень для обробки алгоритмом;
3. Отримання згенерованих відповідей або рекомендацій;
4. Інтеграції із зовнішніми застосунками чи сервісами.

Переваги REST API:

- **Простота та гнучкість** – взаємодія відбувається через стандартні HTTP–запити, зрозумілі будь– якій мові програмування.

- **Масштабованість** – REST– сервіси легко розподіляються між кількома серверами або контейнерами.
- **Незалежність клієнтів і серверів** – клієнти можуть оновлюватися незалежно від серверної логіки.
- **Можливість інтеграції** – REST API дає змогу підключати систему до зовнішніх платформ або робити її частиною мікросервісної архітектури.

Недоліки REST API:

Відсутність стану ускладнює підтримку довготривалих діалогів без додаткового механізму збереження контексту.

Надмірність запитів – при великій кількості звернень може збільшуватися навантаження на сервер.

Безпека залежить від реалізації – неправильно налаштована автентифікація або відкриті ендпоінти створюють ризики.

Передача великих обсягів даних через JSON може бути менш ефективною порівняно з бінарними форматами, як у gRPC.

REST API виступає ключовим елементом зв'язку між клієнтом і сервером у розроблюваній системі. Завдяки своїй гнучкості, масштабованості та незалежності від мови програмування він дозволяє створити єдиний комунікаційний інтерфейс, який може бути використаний як людьми, так і іншими сервісами.

Таким чином, REST API робить можливим підключення модуля автоматичної генерації відповідей як окремого мікросервісу, який можна розгорнути у Docker– контейнері та використовувати у складі різних клієнт–серверних або хмарних систем.

2.4 Контейнеризація та мікросервісний підхід

Беручи до уваги потребу в масштабованості, ізоляції компонентів і простоті розгортання, архітектура розроблюваної системи спирається на контейнеризацію та мікросервісний підхід. Це дозволяє створювати гнучку,

незалежну та легко масштабовану інфраструктуру, у якій кожен компонент виконує чітко визначену роль і може розгортатися окремо.

Контейнеризація – це технологія, яка дозволяє упакувати застосунок разом з усіма його залежностями, конфігураціями та бібліотеками у спеціальний ізольований середовище – контейнер. Найпоширенішим інструментом для цього є Docker, який став стандартом де– факто у світі DevOps і сучасної розробки.

На відміну від віртуальних машин, які створюють повноцінні копії операційної системи, контейнери спільно використовують ядро ОС, але ізольовано виконують свої процеси. Завдяки цьому вони запускаються швидше, споживають менше ресурсів і легко переносяться між різними середовищами – від локального комп'ютера до хмарних платформ.

У межах даного проєкту контейнеризація дозволяє розгортати окремо такі компоненти:

- Модуль генерації повідомлень (рекомендаційна система);
- Сервер REST API;
- Базу даних;
- Інтерфейс користувача.

Мікросервісний підхід – це архітектурний стиль, у якому велика система поділяється на невеликі, незалежні сервіси (мікросервіси). Кожен із них відповідає за певний функціональний блок і спілкується з іншими через стандартизовані інтерфейси, найчастіше – REST API або gRPC.

Для даного проєкту цей підхід дозволяє виділити в окремі сервіси:

- **Core-сервіс** – основна логіка обробки повідомлень та взаємодії з клієнтами;
- **Recommendation-сервіс** – модуль машинного навчання, який аналізує історію переписки та генерує відповіді;
- **Database-сервіс** – робота з базами даних (PostgreSQL, MongoDB тощо);
- **API Gateway** – компонент, який координує запити клієнтів і розподіляє їх між сервісами;
- **Auth-сервіс** – реалізація авторизації та аутентифікації користувачів.

Завдяки цьому можливо розгортати та оновлювати окремі частини системи незалежно, не порушуючи роботу всього застосунку.

Переваги контейнеризації та мікросервісного підходу:

1. **Масштабованість** – кожен мікросервіс можна розгортати у кількох екземплярах, збільшуючи продуктивність при високому навантаженні.
2. **Гнучкість розробки** – різні команди можуть працювати над окремими мікросервісами, навіть використовуючи різні мови програмування.
3. **Простота оновлення** – оновлення одного контейнера не потребує перезапуску всієї системи.
4. **Ізоляція помилок** – якщо один контейнер виходить з ладу, інші продовжують стабільно працювати.
5. **Легке розгортання** – завдяки Docker та Docker Compose можна швидко запускати всі компоненти на будь– якій платформі, зберігаючи однакове середовище.
6. **Сумісність із хмарними платформами** – контейнеризовані застосунки можна розміщувати у середовищах на зразок AWS, Azure, Google Cloud або Kubernetes.
7. **Автоматизація CI/CD** – контейнери легко інтегруються у процеси безперервної інтеграції та доставки оновлень.

Приклад Docekr compose файлу:

```
services:
 web:
   build: .
   command: python manage.py runserver 0.0.0.0:8000
   volumes:
     - .:/app
   ports:
     - "8000:8000"
   environment:
     - DEBUG=1
     - DB_HOST=db
     - DB_NAME=myapp
     - DB_USER=postgres
     - DB_PASSWORD=password
   depends_on:
     - db
```

```
  db:
    image: postgres:15
    environment:
      POSTGRES_DB: myapp
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
    ports:
      -  "5432:5432"
    volumes:
      -  postgres_data:/var/lib/postgresql/data/
volumes:
     postgres_dat:
```

Таким чином, поєднання контейнеризації та мікросервісної архітектури забезпечує гнучку, модульну та масштабовану інфраструктуру, яка спрощує процес розробки, тестування та розгортання системи. Кожен компонент працює у власному контейнері, що гарантує стабільність роботи, незалежність від середовища та можливість швидкого оновлення без простоїв.

Завдяки використанню Docker та Docker Compose розробники отримують уніфікований спосіб керування сервісами, а також можливість легкого перенесення проєкту між різними середовищами – від локальної розробки до хмарних рішень. Це створює міцне підґрунтя для подальшої автоматизації процесів CI/CD та ефективного масштабування системи у міру зростання навантаження.

2.5 Вибір стеку технологій для реалізації.

Після визначення архітектури та загальних принципів побудови системи постає питання вибору оптимального технологічного стеку. Враховуючи вимоги до масштабованості, модульності, безпеки, зручності інтеграції та швидкості розробки, для реалізації запропонованої системи було обрано такі основні технології: Python, Django, PostgreSQL та Git.

Python – це високорівнева мова програмування, яка відзначається простотою синтаксису, широкими можливостями роботи з даними та розвиненою екосистемою бібліотек. Завдяки цьому вона ідеально підходить для реалізації систем, пов’язаних із машинним навчанням, обробкою природної мови (NLP) та інтеграцією з вебсервісами. Python має численні фреймворки для

веброзробки, серед яких Django є одним із найпотужніших і найзручніших для створення багаторівневих клієнт– серверних систем [38].

Розгляньмо переваги мови програмування python в контексті цієї роботи:

Python є стандартом де-факто у сфері штучного інтелекту. Він підтримує величезний набір бібліотек, які прискорюють розробку та забезпечують високий рівень продуктивності:

- TensorFlow, PyTorch – для моделювання та навчання глибоких нейронних мереж;
- Hugging Face Transformers – для використання GPT, BERT, LLaMA, RoBERTa та інших сучасних моделей;
- spaCy, NLTK – для синтаксичного аналізу, токенізації, лематизації, побудови словників та роботи з корпусами;
- LangChain – для побудови складних діалогових систем і інтеграції LLM.

Це дозволяє швидко створювати прототипи систем NLP, здійснювати експерименти з різними моделями та оптимізувати процес обробки тексту.

Висока швидкість розробки та читабельність коду:

Синтаксис Python є максимально простим та інтуїтивним, що значно прискорює процес розробки. Це особливо важливо при створенні систем автоматизованої комунікації, де необхідно швидко тестувати різні архітектури, обробники повідомлень, API-інтерфейси або компоненти логіки діалогу.

Python знижує поріг входу, полегшує командну розробку та мінімізує кількість помилок завдяки читабельності коду.

Можливість інтеграції з будь-якими вебсервісами та APIД:

- Python підтримує стандартизовані бібліотеки для роботи з мережевими протоколами (HTTP, WebSocket), що дозволяє:
- отримувати повідомлення з месенджерів (Telegram API, Slack API, WhatsApp Cloud API);
- відправляти запити на сервери моделей (OpenAI API, Hugging Face API);

- реалізовувати REST та GraphQL-сервіси через фреймворки Django, FastAPI чи Flask.

У задачах твого типу Python дозволяє безперешкодно зв'язати модель NLP, бекенд-систему та інтерфейс месенджера в єдиний модуль.

Гнучкість архітектури та модульність:

Python підтримує масштабовані архітектурні підходи – модульність, розподілені сервіси, мікросервісну структуру, використання Docker та Kubernetes.

Це дозволяє:

- виносити обчислювальні частини системи в окремі контейнери,
- розмежовувати логіку генерації відповідей від логіки маршрутизації,
- легко розширювати проєкт новими модулями (обробка контексту, бази даних, класифікація намірів тощо).

Підтримка асинхронних обчислень у реальному часі:

Для чат-ботів критично важливою є швидка реакція на повідомлення користувача. Python забезпечує це через:

- asyncio – асинхронну бібліотеку для роботи з неблокуючими операціями;
- FastAPI/Quart/AIOHTTP – фреймворки для створення високопродуктивних асинхронних серверів;
- вбудовану підтримку роботи з WebSocket.

Таким чином Python легко справляється із задачами реального часу.

Багатий набір інструментів для роботи з даними та аналітики:

У процесі автоматизації комунікацій потрібно зберігати та аналізувати історію листування. Python має потужний стек:

- Pandas – аналіз даних;
- NumPy – робота з великими масивами;
- Matplotlib/Seaborn – візуалізація;
- SQLite/PostgreSQL інтеграції – для ефективного керування даними.

Це дозволяє вести логування повідомлень, аналізувати ефективність чат-бота, оптимізувати модель.

Універсальність і кросплатформеність:

Python працює на:

- Windows,
- Linux,
- macOS,
- серверних хмарних середовищах (AWS, Azure, Google Cloud).

Це дозволяє запускати моделі локально, у Docker-контейнері або на хмарних сервісах без зміни коду.

Велика та активна спільнота:

Python має одну з найбільших світових спільнот, завдяки чому:

- постійно оновлюються бібліотеки;
- випускаються нові NLP-інструменти;
- легко знайти приклади, документацію, поради та рішення проблем.

Активний розвиток екосистеми робить Python стабільним вибором для довгострокових проєктів.

Інтеграція з моделями глибокого навчання без додаткових шарів:

Python є рідною мовою для більшості нейронних фреймворків. Моделі GPT, BERT, LLaMA та інші в першу чергу мають Python-інтерфейси. Це дає змогу:

- працювати з моделями безпосередньо;
- швидко завантажувати pre-trained моделі;
- застосовувати fine-tuning;
- використовувати пайплайни для генерації тексту та класифікації.

Для нашого типу завдань це означає, що інтеграція моделі – максимально проста.

Загалом переваги Python – від розвиненої екосистеми для NLP до простої інтеграції з вебсервісами – роблять його одним із найкращих виборів для систем

автоматизованої генерації відповідей. Завдяки універсальності, модульності та потужним ML-інструментам Python забезпечує ефективну розробку та масштабування розумних діалогових систем.

Django – це фреймворк із принципом "Don't Repeat Yourself" (не повторюй себе), який забезпечує чітку структуру проєкту, високий рівень безпеки, автоматичне управління базою даних через ORM (Object– Relational Mapping) і вбудовану систему аутентифікації. Django також надає зручний інтерфейс для створення REST API, що дає змогу легко реалізувати взаємодію між модулями або зовнішніми клієнтами, включно з іншими серверами, які можуть звертатися до системи для генерації повідомлень [38].

Розгляньмо його основні переваги більш детально:

Django є одним із найпопулярніших веб-фреймворків для розробки сучасних застосунків на мові Python. Його основна філософія ґрунтується на принципі «Don't Repeat Yourself» (не повторюй себе), що забезпечує максимальну ефективність розробки та дозволяє уникати дублювання коду. Завдяки цьому розробники можуть зосередитися на логіці проєкту та функціональності, не витрачаючи час на повторне написання стандартних шаблонів або модулів.

У Django є чітка структура проєкту. Вона передбачає розділення коду на логічні блоки, включаючи модулі для роботи з базою даних, представленнями та шаблонами інтерфейсу користувача. Такий підхід сприяє не лише швидкій розробці, але й подальшому обслуговуванню системи, адже нові учасники команди можуть швидко зрозуміти архітектуру проєкту та інтегруватися у процес розробки без тривалого навчання.

Django надає комплексний набір механізмів захисту від поширених веб-загроз, таких як SQL-ін'єкції, XSS-атаки, CSRF-атаки та підробка сесій. Вбудовані системи аутентифікації та управління користувачами дозволяють швидко реалізувати контроль доступу, а шифрування паролів та підтримка сучасних методів аутентифікації підвищують загальний рівень захищеності

застосунку. Це особливо важливо для систем, які працюють з конфіденційною інформацією, фінансовими даними або персональними даними користувачів.

Django також спрощує роботу з базою даних завдяки ORM (Object–Relational Mapping), який дозволяє взаємодіяти з таблицями та записами у вигляді об'єктів Python. Це означає, що розробнику не потрібно писати складні SQL-запити вручну, оскільки ORM автоматично генерує їх на основі моделей. Крім того, ORM забезпечує незалежність від конкретної системи управління базами даних, що дозволяє легко змінювати СУБД у разі потреби, не порушуючи логіку застосунку.

Підтримка Django для створення REST API. Це дає змогу організувати ефективну взаємодію між модулями застосунку та зовнішніми клієнтами, включно з мобільними застосунками, веб-сервісами або іншими серверами. Наявність таких можливостей дозволяє швидко інтегрувати систему у складні екосистеми, де потрібно передавати або отримувати дані в стандартизованому форматі, наприклад JSON. Вбудовані інструменти та бібліотеки для побудови REST API значно скорочують час розробки та мінімізують кількість помилок у процесі інтеграції.

Крім того, Django має активну та розвинену спільноту, що забезпечує регулярне оновлення фреймворку, випуск нових версій із покращеною продуктивністю та безпекою, а також доступ до численних сторонніх пакетів і розширень. Це дозволяє інтегрувати складні функції, такі як обробка повідомлень, аналітика даних, інтеграція з платіжними системами або соціальними мережами, без необхідності реалізовувати все з нуля.

Важливою характеристикою Django є його масштабованість. Фреймворк однаково ефективно працює як із невеликими веб-порталами, так і з високонавантаженими корпоративними платформами, де одночасно обробляються тисячі запитів. Оптимізована робота з кешуванням, можливість горизонтального масштабування та підтримка різних стратегій розгортання роблять його надійним вибором для проєктів будь-якого масштабу.

Завдяки поєднанню чіткої архітектури, безпеки, зручності управління базами даних і підтримки сучасних API Django дозволяє розробникам швидко створювати стабільні та функціональні веб-системи. Цей фреймворк особливо корисний у ситуаціях, коли важлива швидкість розробки, підтримка складної логіки та інтеграція з іншими сервісами, при цьому забезпечується висока надійність та довговічність створеної системи.

У підсумку, Django поєднує в собі ефективність, безпеку та масштабованість, що робить його оптимальним рішенням для сучасних веб-застосунків, інтегрованих систем і проектів із високими вимогами до архітектури та підтримки користувачів. Фреймворк не лише прискорює розробку, але й забезпечує стабільність та зручність подальшого розвитку системи, що особливо цінно для команд, які працюють над довгостроковими проєктами.

PostgreSQL було обрано як основну систему управління базами даних завдяки її стабільності, високій продуктивності та підтримці складних запитів. Вона чудово поєднується з Django ORM, що дозволяє швидко розробляти моделі даних і забезпечує надійність зберігання інформації, включно з історією переписки, журналами активності користувачів і результатами аналітики [39].

Розгляньмо переваги використання PostgreSQL більш детально:

PostgreSQL є однією з найпотужніших реляційних систем управління базами даних (РСУБД), яка відзначається винятковою стабільністю та надійністю. Її ядро розроблене таким чином, щоб підтримувати складні транзакції та одночасну роботу великої кількості користувачів без втрати цілісності даних. Завдяки цьому PostgreSQL підходить як для невеликих проектів, так і для масштабних корпоративних систем, де критично важлива безпека й стійкість роботи бази [40, 41].

PostgreSQL має повну підтримку стандартів SQL. Це дозволяє розробникам швидко адаптувати свої навички та застосовувати відомі методи роботи з даними без потреби навчатися новим, специфічним синтаксисам. При цьому система не обмежує користувача стандартними функціональними можливостями – вона надає широкий спектр розширень, які можна інтегрувати

під конкретні потреби проекту. Розширюваність PostgreSQL дозволяє створювати власні типи даних, функції, оператори та навіть мови програмування для виконання процедур у базі, що робить її ідеальним інструментом для інноваційних рішень.

Підтримка транзакцій та механізму ACID, який гарантує повну надійність операцій над даними. Це критично для систем, де будь-яка втрата або неконсистентність інформації може призвести до фінансових або репутаційних втрат. PostgreSQL забезпечує як атомарність транзакцій, так і відновлення бази після збоїв [42], підтримуючи журнали WAL (Write-Ahead Logging), що дає змогу відтворювати стан системи навіть після серйозних збоїв апаратного забезпечення.

PostgreSQL ефективно працює як на окремих серверах, так і в розподілених кластерах, підтримуючи паралельне виконання запитів і реплікацію даних у реальному часі. Це дозволяє використовувати її для аналітичних задач та обробки великих обсягів даних, не жертвуючи швидкістю відповіді системи. Система також підтримує індексацію практично будь-яких типів даних, включно з повнотекстовим пошуком, геопросторовими даними та JSON, що розширює сферу її застосування далеко за межі традиційних реляційних моделей [43].

Активна та широка спільнота, що підтримує PostgreSQL. Це забезпечує постійний розвиток системи, випуск оновлень із новими функціями та покращеннями продуктивності, а також доступ до численних бібліотек, фреймворків та інструментів для інтеграції з різними технологіями. Система є відкритою та безкоштовною, що значно знижує витрати на ліцензії у порівнянні з комерційними аналогами, і водночас не обмежує можливості використання в комерційних проектах [44].

Що стосується безпеки, PostgreSQL надає комплексний набір механізмів контролю доступу. Це включає аутентифікацію на основі ролей, шифрування даних, управління правами на рівні рядків та таблиць, а також інтеграцію з зовнішніми системами ідентифікації. Такі можливості дозволяють створювати

системи з різним рівнем доступу до даних для користувачів, забезпечуючи відповідність сучасним вимогам до безпеки та конфіденційності.

PostgreSQL підтримує гібридну роботу з різними типами даних. Це не тільки традиційні числові та текстові типи, але й структуровані документи у форматі JSON, XML, масиви, геопросторові об’єкти та користувацькі типи [45]. Така гнучкість дозволяє розробникам створювати складні, багатовимірні моделі даних без необхідності використовувати кілька різних систем для зберігання інформації.

Завдяки цим властивостям PostgreSQL стає оптимальним вибором для систем, які потребують високої надійності, масштабованості та гнучкості [46]. Вона здатна обслуговувати як високонавантажені веб-застосунки, так і аналітичні платформи, інтернет-сервіси та корпоративні системи управління даними. Поєднання відкритості, потужного функціоналу та активної спільноти робить її конкурентоспроможною навіть у порівнянні з комерційними рішеннями.

В кінцевому підсумку, PostgreSQL поєднує у собі стабільність класичних реляційних баз даних та гнучкість сучасних технологій обробки даних, що дозволяє реалізовувати складні задачі без компромісів між надійністю, продуктивністю та функціональністю [47]. Для будь-якого проєкту, який потребує високого рівня контролю над даними та довгострокової масштабованості, PostgreSQL є одним із найефективніших рішень, яке не вимагає додаткових витрат на ліцензії та надає широкі можливості для інтеграції та розвитку системи.

Оскільки PostgreSQL є реляційною базою даних, вона повинна відповідати нормальним формам, а саме принципам нормалізації, які забезпечують логічну цілісність та ефективність структури даних. Нормалізація бази даних являє собою процес організації таблиць та їхніх взаємозв'язків таким чином, щоб мінімізувати надмірність інформації та запобігти виникненню різноманітних аномалій під час операцій з даними. Цей підхід базується на математичній теорії реляційних баз даних, розробленій Едгаром Коддом, і передбачає послідовне

застосування кількох рівнів нормалізації, кожен з яких вирішує специфічні проблеми структурування даних.

Перша нормальна форма встановлює базові вимоги до організації даних, визначаючи, що кожна таблиця повинна містити лише атомарні значення, тобто неподільні елементи інформації. Це означає, що в жодному полі таблиці не може зберігатися кілька значень одночасно, як-от списки або масиви даних, що не можуть бути далі декомпозовані без втрати змісту. Наприклад, якщо в таблиці клієнтів зберігається інформація про номери телефонів, і один клієнт має декілька контактних номерів, неправильним підходом буде зберігати всі номери в одному полі через кому або крапку з комою. Натомість слід створити окрему таблицю для телефонних номерів, де кожен запис міститиме один номер, пов'язаний із відповідним клієнтом через зовнішній ключ. Додатково перша нормальна форма вимагає, щоб кожен запис у таблиці був унікально ідентифікований первинним ключем, що забезпечує можливість однозначного звернення до конкретного рядка даних. Порушення цих правил призводить до ускладнення запитів до бази даних, оскільки обробка складених значень потребує додаткових операцій розбору та перетворення, а також унеможливлює ефективне використання індексів та інших механізмів оптимізації [48].

Друга нормальна форма розвиває принципи першої та усуває проблему часткової функціональної залежності, яка виникає тоді, коли неключові атрибути залежать лише від частини складеного первинного ключа. Ця ситуація характерна для таблиць, де первинний ключ складається з декількох полів, і деякі атрибути визначаються не всією комбінацією ключових полів, а лише окремими її компонентами. Розглянемо приклад таблиці замовлень, де первинний ключ складається з ідентифікатора замовлення та коду товару, а також зберігається назва товару, його ціна та дата замовлення. У цьому випадку назва товару та його ціна залежать виключно від коду товару, а не від повного ключа, що порушує другу нормальну форму. Така структура створює надмірність даних, адже інформація про один товар дублюватиметься в кожному записі замовлення, де цей товар фігурує. Для приведення до другої нормальної форми необхідно

розділити таблицю на дві: одну для замовлень, що містить ідентифікатор замовлення, код товару, кількість та дату, і другу для товарів, де зберігатиметься код товару, його назва та ціна. Такий підхід усуває дублювання інформації та пов'язані з ним аномалії модифікації [49].

Третя нормальна форма спрямована на усунення транзитивних функціональних залежностей, коли неключовий атрибут залежить від іншого неключового атрибута, а не безпосередньо від первинного ключа. Це означає, що всі атрибути таблиці повинні залежати виключно від первинного ключа і не повинні залежати один від одного опосередковано. Припустимо, в таблиці співробітників зберігається ідентифікатор працівника як первинний ключ, код відділу, де він працює, та назва цього відділу. У такій структурі назва відділу залежить не від ідентифікатора співробітника, а від коду відділу, що створює транзитивну залежність. Якщо назва відділу зміниться, доведеться оновлювати інформацію в усіх записах співробітників, які належать до цього відділу, що збільшує ризик помилок та неузгодженості даних. Правильним рішенням буде створення окремої таблиці відділів, де зберігатиметься код відділу та його назва, а в таблиці співробітників залишиться лише посилання на код відділу. Таким чином забезпечується пряма залежність усіх атрибутів від первинного ключа їхньої таблиці [50].

Нормальна форма Бойса-Кодда є посиленим варіантом третьої нормальної форми і застосовується до таблиць, де існують множинні потенційні ключі або складні залежності між атрибутами. Вона вимагає, щоб для кожної функціональної залежності в таблиці лівий бік цієї залежності був суперключем, тобто унікально визначав рядок. Ця форма розв'язує специфічні аномалії, які можуть виникати навіть у таблицях, що відповідають третій нормальній формі, але мають перекриваються потенційні ключі. Наприклад, в таблиці викладання, де зберігається інформація про викладачів, предмети та аудиторії, може існувати правило, що кожен викладач веде лише один предмет, і кожна аудиторія призначена для одного предмета. У такій ситуації можуть виникати конфлікти та аномалії при спробі оновлення або вставки даних, які усуваються шляхом

подальшої декомпозиції таблиці відповідно до вимог нормальної форми Бойса-Кодда [51].

Четверта нормальна форма розглядає ситуації, коли в таблиці існують багатозначні залежності, не пов'язані одна з одною. Багатозначна залежність виникає тоді, коли один атрибут визначає множину значень інших атрибутів незалежно від решти даних у записі. Типовим прикладом є таблиця, яка зберігає інформацію про навички співробітників та мови, якими вони володіють. Якщо один співробітник має кілька навичок та знає кілька мов, і ці дві множини не пов'язані між собою, виникає декартовий добуток усіх можливих комбінацій, що створює значну надмірність та ускладнює підтримку даних. Для приведення до четвертої нормальної форми необхідно розділити таку таблицю на дві окремі: одну для зв'язку співробітників з їхніми навичками та другу для зв'язку співробітників з мовами. Це усуває непотрібні комбінації та спрощує операції додавання, оновлення та видалення інформації.

П'ята нормальна форма, також відома як проекційно-з'єднувальна нормальна форма, є найвищим рівнем нормалізації і застосовується до таблиць, які мають складні залежності з'єднання. Вона вимагає, щоб таблиця не могла бути декомпозована на менші таблиці без втрати інформації, і що будь-яка декомпозиція має бути оборотною через природне з'єднання. Ця форма зустрічається рідко і стосується дуже специфічних випадків, де існують тернарні або більш складні зв'язки між атрибутами, які не можуть бути адекватно представлені через бінарні відношення. На практиці більшість систем баз даних рідко потребують нормалізації до п'ятої форми, оскільки перші три нормальні форми вже забезпечують достатній рівень усунення надмірності та аномалій для типових застосувань [52].

Аномалії в базах даних виникають саме через порушення принципів нормалізації і проявляються у вигляді проблем під час виконання операцій вставки, оновлення та видалення даних. Аномалії вставки виникають тоді, коли неможливо додати новий запис без наявності інформації, яка логічно не повинна бути обов'язковою. Наприклад, у ненормалізованій таблиці, яка зберігає

інформацію про студентів, їхні курси та викладачів, неможливо додати інформацію про нового викладача, який ще не веде жодного курсу, оскільки таблиця вимагає наявності зв'язку з конкретним курсом та студентом. Це змушує або залишати поля пустими, що порушує цілісність даних, або створювати штучні записи з фіктивними даними.

Аномалії оновлення проявляються у необхідності модифікувати один і той самий факт у багатьох місцях бази даних через дублювання інформації. Якщо в таблиці замовлень зберігається адреса клієнта разом з іншими даними замовлення, то при зміні адреси клієнта доведеться оновлювати всі записи його замовлень, що створює ризик неузгодженості, коли деякі записи будуть оновлені, а інші ні. Така ситуація призводить до суперечливих даних у системі, коли один і той самий клієнт фігурує з різними адресами в різних записах. Правильна нормалізація усуває цю проблему шляхом зберігання адреси клієнта один раз у окремій таблиці, що вимагає оновлення лише одного запису при зміні інформації [43].

Аномалії видалення виникають тоді, коли видалення одного запису призводить до непередбаченої втрати іншої важливої інформації, яка логічно не повинна бути видалена. У ненормалізованій структурі, де інформація про відділи компанії зберігається разом з даними про співробітників, видалення останнього працівника відділу автоматично призведе до втрати інформації про сам відділ, навіть якщо відділ продовжує існувати як організаційна одиниця. Це створює ситуацію, коли видалення даних про сутність одного типу спричиняє небажане видалення даних про сутність іншого типу, що порушує логічну незалежність інформації.

Процес нормалізації не завжди доводиться до найвищих форм у реальних системах, оскільки іноді надмірна нормалізація може призвести до зниження продуктивності через необхідність виконання численних операцій з'єднання таблиць при отриманні даних. Тому в практиці проектування баз даних застосовується баланс між нормалізацією та денормалізацією, коли певні відхилення від строгих правил допускаються заради підвищення швидкості

виконання запитів. Однак такі рішення приймаються усвідомлено, з розумінням наслідків та компромісів, і лише після того, як базова структура була спроектована відповідно до принципів нормалізації. PostgreSQL надає всі необхідні інструменти для реалізації як строго нормалізованих структур, так і оптимізованих схем з контрольованою денормалізацією, включаючи матеріалізовані представлення, індекси та механізми кешування, що дозволяє знайти оптимальне рішення для конкретних вимог системи.

Docker – це платформа для контейнеризації застосунків, яка дозволяє пакувати програмне забезпечення та всі його залежності в окремі ізольовані контейнери. Такий підхід забезпечує стабільну роботу програм у різних середовищах без необхідності налаштовувати кожне середовище вручну. Контейнер містить усе, що потрібно для запуску застосунку – код, бібліотеки, конфігураційні файли та системні утиліти, що робить його переносимим і відтворюваним на будь-якому сервері або робочій станції, де встановлено Docker.

Однією з ключових переваг Docker є стандартизація середовища. Розробник може бути впевнений, що застосунок працюватиме однаково на локальному комп'ютері, тестовому сервері та в продуктивному середовищі. Це значно зменшує ризик виникнення помилок, пов'язаних із відмінностями конфігурацій і версій програмного забезпечення.

Контейнери Docker забезпечують ізоляцію застосунків один від одного, що дозволяє запускати на одному сервері кілька проєктів без конфліктів залежностей. Кожен контейнер працює незалежно, використовуючи власне файлове середовище, мережеві налаштування та ресурси, що підвищує безпеку та стабільність роботи системи.

Docker дозволяє запускати легкі контейнери, які споживають мінімум оперативної пам'яті та процесорного часу у порівнянні з повноцінними віртуальними машинами. Це робить систему більш продуктивною, зменшує витрати на інфраструктуру та спрощує масштабування застосунків при зростанні навантаження.

Docker спрощує процес розгортання та інтеграції систем. Завдяки інструментам, таким як Docker Compose, можна одночасно запускати декілька контейнерів із налаштованими зв'язками між ними, що дозволяє швидко розгортати складні багатомодульні системи, включаючи веб-сервери, бази даних, кеші та інші сервіси. Це особливо корисно при роботі з мікросервісною архітектурою або великими проєктами, де різні компоненти повинні взаємодіяти один з одним.

Docker має досить прості механізми масштабування та автоматизації. Контейнери можна легко дублювати, запускати на різних серверах або в хмарі, що робить масштабування горизонтальним і майже безболісним. Інструменти оркестрації, такі як Kubernetes, інтегруються з Docker і дозволяють автоматично керувати життєвим циклом контейнерів, балансувати навантаження та забезпечувати високу доступність системи.

Docker також полегшує тестування та розробку. Розробники можуть створювати контейнери для тестових середовищ, де можна проводити автоматизовані тести без ризику впливу на основну систему. Це прискорює цикл розробки та підвищує якість коду, оскільки однакова конфігурація середовища використовується як у розробці, так і у тестуванні та продуктивному середовищі.

У підсумку, Docker поєднує простоту, портативність і контроль над середовищем виконання, що робить його незамінним інструментом для сучасної розробки програмного забезпечення. Він дозволяє забезпечити стабільність, безпеку та ефективність проєктів будь-якого масштабу, від невеликих веб-застосунків до складних систем з мікросервісною архітектурою та високими вимогами до масштабування.

Для контролю версій використовується Git [40], який забезпечує ефективну співпрацю між розробниками, можливість створення окремих гілок для різних модулів і зручне відстеження змін. Репозиторій системи може бути розміщений на GitHub або GitLab, що полегшує командну роботу, розгортання через CI/CD та інтеграцію з Docker– контейнерами.

Застосування такого поєднання технологій дозволяє створити гнучку, модульну та безпечну систему, яку можна легко масштабувати та переносити на інші платформи. Python і Django забезпечують швидку розробку та гнучкість, PostgreSQL – стабільність і ефективне управління даними, а Git – контроль над процесом розробки. У поєднанні з контейнеризацією та REST API це створює надійне технологічне середовище для побудови інтелектуальної системи рекомендацій і безпечного месенджера.

Окрім основних технологій, важливу роль у реалізації системи відіграють бібліотеки Python, які забезпечують реалізацію ключових функцій – від обробки природної мови до взаємодії з мережею та контейнерами. Нижче наведено основні бібліотеки, що використовуватимуться у проєкті:

Natural Language Toolkit (NLTK) – одна з найпопулярніших бібліотек для роботи з текстами та виконання базових операцій обробки природної мови. Вона надає інструменти для токенізації, лематизації, стемінгу, частиномовного розбору та побудови частотних словників. Використання NLTK дозволяє підготувати текстові дані для подальшого аналізу або машинного навчання, що є критично важливим при аналізі історії переписки користувачів [51].

spaCy – орієнтована на високопродуктивну обробку великих обсягів тексту. Вона використовується для більш глибокого синтаксичного аналізу, виявлення іменованих сутностей (Named Entity Recognition, NER) і створення векторних представлень слів. У контексті системи рекомендацій spaCy може бути застосована для виявлення контексту повідомлення, визначення намірів користувача або класифікації типу діалогу [51].

Transformers (Hugging Face) – одна з ключових технологій, яка забезпечує інтелектуальні можливості системи. Бібліотека Transformers надає доступ до сучасних передтренованих моделей глибокого навчання, таких як BERT, GPT-2/GPT-Neo/GPT-J, RoBERTa, DistilBERT, T5 та сотні інших моделей, оптимізованих під різні задачі обробки природної мови. Її використання дозволяє інтегрувати потужні нейронні мережі без необхідності проводити

повноцінне навчання з нуля, що значно скорочує час розробки та обчислювальні витрати [52–56].

У контексті нашої системи Transformers відіграє важливу роль, оскільки саме ця технологія забезпечує інтелектуальну обробку діалогів, генерацію текстових відповідей і формування рекомендаційних повідомлень. Моделі на основі трансформерів здатні аналізувати контекст усього діалогу, визначати наміри користувача, враховувати попередні повідомлення та формувати змістовні відповіді, максимально адаптовані до ситуації. Це дає змогу створити більш «людяний» стиль спілкування, підвищити природність діалогу та покращити взаємодію з користувачем [57].

Важливою властивістю Transformers є можливість донавчання моделей на невеликих спеціалізованих датасетах. Це дозволяє адаптувати загальні мовні моделі під задачі конкретної системи, наприклад:

- генерування пропозицій або рекомендаційних підказок у чаті;
- класифікацію типів повідомлень (інформаційне, питання, уточнення, емоційне тощо);
- виявлення ризикових або небажаних повідомлень у контексті безпеки;
- розпізнавання намірів (Intent Detection) для кращого розуміння інтерфейсу взаємодії.

Бібліотека також надає інструменти для оптимізації моделей, такі як quantization, pruning та ONNX Runtime, що дозволяє запускати моделі на обмежених ресурсах або інтегрувати їх у Docker–контейнери, не жертвуючи продуктивністю. Це особливо корисно для нашої системи яка працює як окремий модуль та може бути розгорнута на різних серверах або хмарних середовищах [58].

Завдяки широким можливостям Transformers стає центральним елементом інтелектуальної підсистеми системи. Він забезпечує:

- глибокий контекстуальний аналіз тексту;
- можливість кастомізації моделей під конкретні задачі;

- інтеграцію з REST API для швидкого отримання рекомендацій;
- масштабованість та адаптивність під різні сценарії використання.

Таке застосування сучасних моделей трансформерів робить систему не просто механізмом для обробки повідомлень, а повноцінною інтелектуальною платформою, здатною виявляти контекст, прогнозувати наміри та підсилювати комунікацію між користувачами. Це суттєво підвищує якість взаємодії та дозволяє системі адаптуватися до складних сценаріїв реального спілкування.

Scikit– learn – це класична бібліотека для реалізації алгоритмів машинного навчання. Вона забезпечує засоби для класифікації, кластеризації, побудови регресійних моделей та оцінювання якості результатів. У системі вона може використовуватись для аналітичних підсистем – наприклад, для групування користувачів за стилем спілкування або створення базових рекомендацій на основі історичних даних [52, 57].

Logging та Rich – для ведення журналів роботи системи та моніторингу подій буде використано модулі logging (вбудований у Python) і Rich для зручного візуального відображення логів. Це дозволить забезпечити відстеження роботи кожного компонента і спростить діагностику помилок [52].

Ще одним важливим інструментом, який використовувався під час розробки системи, є Swagger (OpenAPI). Swagger забезпечує автоматичне документування REST API та створення інтерфейсу для інтерактивної взаємодії з ендпоінтами системи. Завдяки цьому розробники та сторонні інтегратори можуть легко протестувати доступні маршрути, переглянути структуру запитів і відповідей, а також зрозуміти логіку роботи API без необхідності заглиблюватись у вихідний код.

У контексті Django найбільш поширеним рішенням є використання бібліотек drf–yasg або drf–spectacular, які автоматично генерують OpenAPI–специфікацію на основі описаних у коді серіалізаторів, в'юх і моделей. Це дає змогу зменшити кількість потенційних помилок, підвищити прозорість структури API та забезпечити зручність тестування. Swagger UI також дозволяє

виконувати запити прямо з браузера, що значно спрощує відлагодження серверної частини під час розробки.

Оскільки запропонована система являє собою окремий самодостатній модуль із власною логікою, правилами безпеки та функціональністю, наявність детальної та автоматично підтримуваної документації є критично важливою. Swagger забезпечує прозорість архітектури API, спрощує тестування, пришвидшує розробку та полегшує подальше масштабування системи. Це робить технологічний стек повнішим і більш зручним для підтримки в майбутньому.

Висновки до другого розділу

У другому розділі було проведено комплексний аналіз предметної області, сформульовано постановку задачі та визначено архітектурні засади побудови модуля автоматичної генерації відповідей. На основі дослідження встановлено, що зростання обсягів цифрової комунікації, а також потреба у персоналізованих і швидких відповідях, зумовлюють необхідність створення окремого інтелектуального компонента, здатного автономно обробляти текстові повідомлення, аналізувати контекст і формувати релевантні відповіді.

Було обґрунтовано доцільність виокремлення модуля генерації відповідей у незалежний програмний компонент. Такий підхід забезпечує гнучкість, масштабованість та ізоляцію логіки машинного навчання від основного застосунку, що значно полегшує його оновлення, розширення та інтеграцію у сторонні системи. Модульна архітектура дозволяє застосовувати компонент як частину корпоративного месенджера, служби підтримки або іншої платформи обміну повідомленнями.

Аналіз вхідних даних показав, що ключовим фактором якості генерованих відповідей є повнота та структурованість історії переписки. Описано два основних механізми отримання цих даних – прямий доступ до бази даних та отримання через REST API. Обидва підходи забезпечують гнучкість системи та можливість інтеграції з різними джерелами інформації. Особливу увагу

приділено процесу попередньої обробки текстів, який включає нормалізацію, токенізацію, лематизацію, очищення від неінформативних елементів і виділення ключових ознак. Саме якісно підготовлені дані формують основу для коректного контекстного аналізу та подальшої генерації відповідей.

Також було розглянуто архітектурні та проектувальні рішення, що забезпечують ефективність роботи системи. Багаторівнева клієнт-серверна архітектура дозволяє розмежувати відповідальність між рівнем представлення, бізнес-логікою та рівнем даних, спрощує супровід та підвищує безпеку. REST API визначено як оптимальний механізм інтеграції між модулями та зовнішніми сервісами, завдяки його простоті, масштабованості та підтримці стандартів аутентифікації й шифрування.

Узагальнюючи проведений аналіз, можна стверджувати, що розробка окремого модуля автоматичної генерації відповідей на основі машинного навчання є технічно обґрунтованим і ефективним рішенням. Визначена структура системи, механізми обробки даних, а також обрана архітектура створюють надійну основу для подальшої реалізації, навчання моделі та інтеграції модуля в ширші програмні комплекси. Усе це забезпечує можливість побудови гнучкого, масштабованого та безпечного інтелектуального сервісу, здатного адаптуватися до різних сценаріїв використання.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ РЕКОМЕНДАЦІЙ

3.1 Архітектура системи рекомендацій

3.1.1 API інтерфейс

API інтерфейс системи є центральним механізмом взаємодії між клієнтськими застосунками (веб– інтерфейс, мобільний застосунок, адмін–панель) та серверною частиною. У розробленій системі API побудований за принципами архітектурного стилю REST та документований за допомогою платформи Swagger/OpenAPI, що забезпечує прозорість, контрольованість та зручність інтеграції зовнішніх компонентів.

Усі ендпоінти працюють через протокол HTTP та обмінюються даними у форматі JSON, що є галузевим стандартом для сучасних інформаційних систем. API підтримує аутентифікацію на основі токенів, що гарантує захищений доступ до ресурсів.

API структуровано за доменними модулями, кожен із яких відповідає за певну частину функціональної логіки системи. Згідно з наданою Swagger–специфікацією, API включає такі розділи:

Модуль авторизації (api/login):

POST /api/login/ – цендпоінт використовується для автентифікації користувача та виконує наступні функції.

- приймає логін та пароль користувача;
- виконує перевірку облікових даних;
- у разі успіху повертає токен доступу (JWT Token).

Функціональна роль в системі:

Забезпечує захищений доступ до всіх інших ресурсів API. Без успішного логіну доступ до CRUD – операцій та модуля рекомендацій заборонений.

Модуль Learning (CRUD – операції над навчальними даними).

Цей модуль надає повний набір REST – операцій для ресурсу learning. Він включає такі енд поінти:

GET /learning/ – отримання списку всіх записів.

POST /learning/create/ – створення нового запису.

GET /learning/{learn_id}/ – отримання конкретного запису за унікальним ідентифікатором.

PUT /learning/{learn_id}/ – повне оновлення ресурсу.

PATCH /learning/{learn_id}/ – часткове оновлення (оновлюються лише передані поля).

DELETE /learning/{learn_id}/ – видалення запису.

Модуль Learning є точкою входу для користувача щоб виконанти донавчання моделі (тюнінг), спеціально переданими форматами даних. Він забезпечує повний життєвий цикл даних для навчання починаючи від створення набору навчальних даних закінчуючи його видаленням із системи.

Модуль Messages:

GET /messages/ – повертає перелік усіх збережених повідомлень.

GET /messages/{message_id}/ повертає конкретне повідмлення.

GET /messages/{chat_id}/ повертає всі повідомлення для певного чату.

Модуль Messages є частиною програми направлений на зберігання історії спілкування, а його ентітя Message є основним об’єктом який використовується при взаємодії із API.

Модуль Recommendations:

POST /recommendations/ – генерує рекомендації на основі надісланих даних.

Модуль Recommendations це один з найважливіших модулів у системі який відповідає за отримання історії спілкування конкретного чату від користувача та повернення ймовіних варіантів відповіді враховуючи контекст спілкування. В міру своєї важливості він потребує додаткової докуменлації згідно джерелам [59, 60]. Це потрібно щоб користувач міг точно розуміти які дані

ми очікуємо в ньому для цього було використано декоратор а саме «swagger_auto_schema» розгляньмо його:

```
@swagger_auto_schema(
        operation_description="Отримує список повідомлень і
повертає рекомендації.",
        request_body=openapi.Schema(
            type=openapi.TYPE_ARRAY,
            items=openapi.Items(
                type=openapi.TYPE_OBJECT,
                properties={
  "chat_id": openapi.Schema(type=openapi.TYPE_INTEGER, example=1),
  "message_text": openapi.Schema(type=openapi.TYPE_STRING,
example="Hello! How are you?"),
  "sent_by": openapi.Schema(type=openapi.TYPE_STRING,
example="Alice"),
  "timestamp": openapi.Schema(type=openapi.TYPE_STRING,
format=openapi.FORMAT_DATETIME,
                                                example="2025- 11-
06 12:00:00"),
                }
            )
        ),
        responses={200: "OK"},
    )
```

Розгляньмо її більш детально:

operation_description – це текстове пояснення, яке додається до Swagger–документації, щоб описати призначення цього ендпоінта. По суті – людське пояснення, що користувач API має отримати від цього методу.

request_body – це декларація того, яку структуру даних запит повинен містити. Тут визначено формат тіла запиту: Swagger завдяки цьому знає, яких полів очікувати, як їх валідувати та як відображати у документації.

Schema(type=ARRAY, items=OBJECT) – цей фрагмент описує, що запит складається зі списку елементів, де кожен елемент – це структурований об’єкт.

properties у Schema – описують зміст кожного об’єкта.

Кожне поле виконує свою роль:

- chat_id – ідентифікатор діалогу, потрібний, щоб зрозуміти, до якого чату належить повідомлення.
- message_text – текст самого повідомлення; основні дані, які будуть оброблятися.

- sent_by – вказує, хто відправив повідомлення; важливо для логіки рекомендацій чи аналізу.
- timestamp – момент відправлення; дозволяє моделі враховувати послідовність і час.

Це не просто перелік параметрів – це структурована модель того, як виглядає одне повідомлення, яке надходить до системи.

example – показує реальний приклад того, як повинні виглядати поля при виклику API. Це допомагає і людині, і Swagger UI правильно формувати запит.

format=openapi.FORMAT_DATETIME – Вказує, що поле повинно бути у форматі дати/часу. Swagger на цій основі робить автоматичну валідацію та підказки.

responses={200: "OK"} – Визначає, яку відповідь Swagger має відобразити як результат успішного виконання запиту.

Моджемо візуально розглянути як виглядає описаний API зо допомогою Swagger на прикладі модуля Recommendations дивитися рис. 3.1:

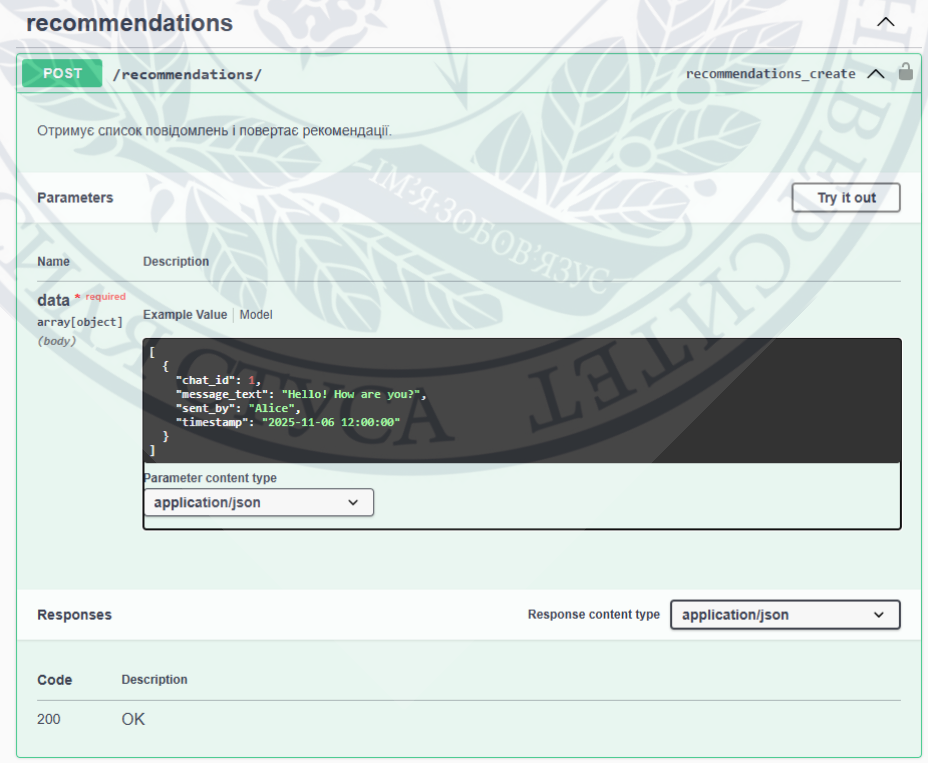

Рисунок 3.1 Модуль рекомендацій в Swagger UI

Усі ендпоінти які відповідають за функціонал системи (Learning , Messages, Recommendations) вимагають наявності авторизаційного токена. Без нього доступ відхиляється.

3.1.2 Механізми безпеки та валідації API

Архітектура авторезації:

1) Користувач передає свої облікові дані через ендпоінт.
2) Сервер успішно автентифікує користувача та генерує унікальний авторизаційний токен.
3) Токен передається клієнту і надалі включається у заголовок кожного запиту: `Authorization: Token <значення_токена>`
4) API блокує будь-яку спробу доступу до захищених ресурсів без цього токена.

Таким чином, система гарантує, що доступ до ендпоінтів для роботи з повідомленнями, навчальними даними та рекомендаціями отримують лише автентифіковані користувачі.

Усі ендпоінти, окрім шляху входу (/login), позначені як захищені, що означає примусову авторизацію.

Такий підхід запобігає:

- несанкціонованому отриманню внутрішніх даних,
- модифікації або видаленню інформації,
- надсиланню фальшивих або шкідливих даних у модуль рекомендацій.

Захист від маніпуляцій і помилок клієнта

Система включає декілька рівнів контролю, спрямованих на виявлення та блокування некоректних, небезпечних або навмисно спотворених запитів. Будь-який запит, що не відповідає визначеним правилам, не допускається до обробки, що гарантує стабільність і безпечність внутрішніх компонентів.

API блокує виконання операцій у випадках, коли:

- Авторизаційний токен відсутній, прострочений або сфальсифікований, що унеможливлює доступ анонімних або підроблених клієнтів;
- Структура тіла запиту не відповідає формальному опису, тобто клієнт намагається передати поля не передбачені схемою, або у неправильному форматі;
- Типи даних несумісні з описом в OpenAPI, наприклад передано текст замість числового ідентифікатора;
- Обов'язкові поля відсутні, що вказує на спробу обходу логіки або на технічну помилку клієнта.

Такі механізми дозволяють захищати систему від широкого спектра атак і помилкових сценаріїв взаємодії.

Зокрема, вони забезпечують:

- Захист від SQL-ін'єкцій через неконтрольовані поля.

Оскільки API приймає лише чітко визначений набір параметрів, сторонні або шкідливі поля не потрапляють до внутрішніх компонентів. Валідаційна схема відсікає будь-які дані, які за структурою або типом не відповідають вимогам. Навіть якщо зловмисник спробує передати SQL-команди у вигляді строк, вони не будуть інтерпретовані сервером, адже такі поля не передбачені моделлю даних.

- Запобігання JSON-маніпуляціям через додавання зайвих параметрів.

Декларативна схема (swagger_auto_schema) не дозволяє клієнту вносити додаткові або вкладені об'єкти, не описані у моделі. Це унеможливлює підміни параметрів, спроби передати конфіденційні дані через приховані поля або обійти серверні обмеження через «маніпулятивні» JSON-структури. Усі зайві поля ігноруються або блокуються на рівні валідації, не доходячи до бізнес-логіки.

- Запобігання помилкам клієнтського коду.

Часто помилки виникають не від зловмисників, а від некоректної реалізації на стороні клієнта. Жорстка валідація дозволяє виявляти і блокувати такі запити на ранньому етапі, забезпечуючи стабільність і правильність роботи API.

Завдяки багаторівневому контролю структури даних, авторизації та валідації API успішно захищений від найпоширеніших атак та непередбачуваних помилок. Система реагує виключно на коректні та авторизовані запити, що забезпечує надійність, стабільність та безпечність роботи модулів рекомендацій і всієї серверної частини.

3.1.3 Обробка вхідних повідомлень

Обробка вхідних повідомлень складається з двох ключових компонентів:

- Видалення чутливої інформації, який видаляє або маскує приватні дані задля забезпечення безпеки користувачів, та
- Аналітичний підмодуль, відповідальний за нормалізацію, токенізацію, вилучення ключових слів та визначення намірів користувача.

Обидва підмодулі працюють послідовно, формуючи конвеєр попередньої обробки тексту, що використовується в подальшому системою для генерації відповідей. Тому розгляньмо їх більш детально.

Видалення чутливої інформації (MessageSanitizer)

Клас MessageSanitizer відповідає за автоматичне очищення текстових повідомлень від конфіденційної інформації. Такий підхід забезпечує виконання вимог GDPR, етичних принципів обробки персональних даних та внутрішніх політик безпеки. Очищення виконується за допомогою набору регулярних виразів, що описують різні види чутливої інформації.

До таких типів даних належать:

- номери банківських карток;
- email-адреси;
- телефонні номери;
- IP-адреси (IPv4, IPv6);
- ідентифікатори UUID;
- IBAN;
- українські номери паспортів, ІПН;

- адресні конструкції;
- автомобільні номери;
- дати народження;
- логіни користувачів у форматі @username.

Санітайзер працює у три етапи:

Етап перший компіляція правил – при ініціалізації класу всі регулярні вирази переводяться у скомпільовані патерни для пришвидшення подальшої обробки. Неправильно складені правила відсікаються, що захищає систему від помилок.

Етап другий покрокова заміна чутливих блоків для кожного повідомлення:

- текст пропускається через послідовність регулярних виразів;
- усі знайдені чутливі фрагменти замінюються маркерами, наприклад: 4111 1111 1111 1111 → [REDACTED_CARD].

Етап третій заміна вього отриманого масиву повідомлень на очищений текст із його занесенням в базу даних.

Модуль виконує критично важливу функцію фільтрації даних, спрямовану на забезпечення інформаційної безпеки системи. Він запобігає потраплянню персональної або конфіденційної інформації до журналів, бази даних чи моделей машинного навчання, не допускає передачу ін'єкційних та шкідливих структур, які можуть вплинути на подальші етапи обробки, а також зменшує ризик обходу бізнес-логіки за допомогою маніпулятивних форматів даних. Завдяки цьому MessageSanitizer фактично виступає першою лінією захисту всієї системи, забезпечуючи коректність, безпечність і контрольованість вхідних повідомлень.

Аналітичний модуль препроцесингу (AnalysingModule)

Другим етапом після очищення є лінгвістична обробка тексту.

Клас AnalysingModule реалізує повний цикл препроцесингу діалогу:

- нормалізацію тексту;
- очищення від зайвих символів;
- токенізацію;

- виділення ключових слів;
- визначення інтенту користувача;
- побудову агрегованого представлення діалогу.

Загальна схема роботи цих модулів зображенна на рисунку 3.2

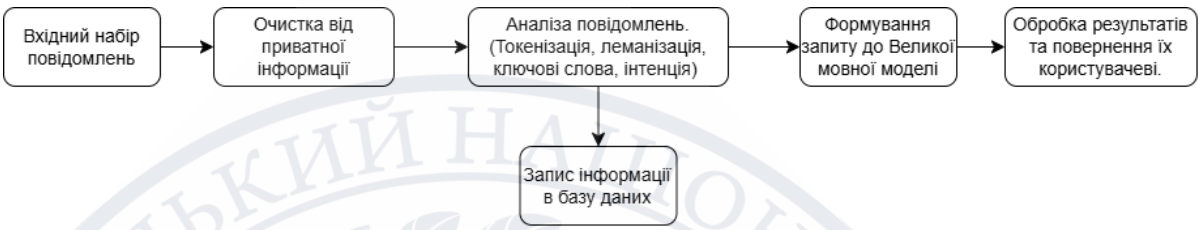

Рисунок 3.2 Схема обробки вхідних повідомлень.

Цей модуль є основним джерелом структурованої інформації, яку згодом використовують логічні моделі та алгоритми генерації відповідей. Він оцінює повідомлення не лише синтаксично, а й семантично, виявляючи приховані ризики, такі як спроби маніпуляції контекстом, нетипові структури запитів, підозрілі послідовності команд або поведінкові розбіжності, що можуть свідчити про загрозу. Завдяки такому багаторівневому підходу система здатна вчасно фіксувати небажані впливи, попереджати помилки на етапах прийняття рішень та мінімізувати можливість експлуатації логіки обробки даних. Аналіз стає ключовим елементом захисту, який забезпечує глибоке розуміння реального змісту кожної взаємодії та підтримує стабільність роботи всієї платформи.

3.2 Генерація рекомендацій за допомогою готової (LLM) моделі.

У рамках даної роботи для автоматичної генерації відповідей використовується готова велика мовна модель (Large Language Model, LLM) із бібліотеки Hugging Face Transformers. Для забезпечення ефективності та повторного використання ресурсоємної моделі реалізовано сервіс LLMService, що інкапсулює логіку завантаження та генерації тексту.

Ключові особливості реалізації:

Ініціалізація моделі – модель та відповідний токенайзер завантажуються один раз при першому виклику сервісу. Це дозволяє уникнути повторного створення важких об’єктів і економить ресурси під час роботи системи. Параметри завантаження включають:

- model_name – назва моделі з Hugging Face Hub (у прикладі використано "google/gemma-3-1b-it"),
- device_map – визначає, на якому пристрої виконуються обчислення (CPU, GPU або авто),
- num_responses – кількість альтернативних варіантів відповіді.

Генерація відповідей – метод generate приймає на вхід останнє повідомлення або історію переписки та формує відповідний промпт для моделі.

Генерація відбувається з параметрами:

- do_sample=True – використовується стохастичний вибір токенів для підвищення різноманітності,
- top_k=30 – обмеження на розгляд найбільш ймовірних токенів,
- temperature=0.7 – контроль креативності відповідей.

Після генерації текст розділяється на кілька варіантів відповіді за символом "|", що дозволяє системі пропонувати користувачу альтернативні формулювання.

Переваги підходу:

Універсальність: Можна замінити модель на будь-яку сумісну з Hugging Face без зміни логіки генерації.

Масштабованість: Завдяки одноразовому завантаженню моделі зменшується час реакції та навантаження на систему.

Гнучкість: Кількість відповідей, ступінь креативності та інші параметри генерації легко змінюються через аргументи сервісу.

Таким чином, реалізований модуль забезпечує інтеграцію сучасної LLM для автоматичного формування рекомендацій і відповідей у чаті, що є критичною частиною системи інтелектуальної взаємодії з користувачем.

Приклад використання модуля генерації відповідей:

Для демонстрації практичної роботи модуля наведемо типовий сценарій його використання у системі. Сервіс LLMService ініціалізується через фабричну функцію get_llm(), яка гарантує створення лише одного екземпляра моделі протягом роботи застосунку. Це забезпечує стабільну продуктивність та оптимальне використання обчислювальних ресурсів.

Вхідні дані:

```
[
  {
    "chat_id": 1,
    "message_text": "Hello! How are you?",
    "sent_by": "Alice",
    "timestamp": "2025-11-06 12:00:00"
  },
  {
    "chat_id": 1,
    "message_text": "Hi Alice! I'm doing well, thanks for asking.
How about you?",
    "sent_by": "Bob",
    "timestamp": "2025-11-06 12:01:30"
  },
  {
    "chat_id": 1,
    "message_text": "I'm good too! Are you free this weekend to
catch up?",
    "sent_by": "Alice",
    "timestamp": "2025-11-06 12:02:10"
  }
]
```

Розглянемо відповіді, згенеровані системою:

```
{
  "status": "success",
  "analyzed_messages": [
    "Yes, I am. ",
    "What are your plans for the weekend?",
    "I was thinking of going for a hike, and maybe some relaxing
time. ",
    "That sounds lovely. ",
    "I'll be there! "
  ]
}
```

Аналіз показав, що LLM здатна формувати загалом логічні та контекстно релевантні відповіді, проте не всі згенеровані варіанти є ідеально точними або природними. Модель правильно зрозуміла основну ідею діалогу – пропозицію зустрічі на вихідні – і частково забезпечила послідовність бесіди, проте деякі

відповіді могли бути більш плавними або детальними. Це можна пояснити використанням відносно невеликої моделі, яка має обмежену здатність підтримувати довгі або складні контексти. Водночас навіть така модель дозволяє отримати декілька альтернативних варіантів відповіді, що робить її придатною для демонстраційних або тестових сценаріїв генерації тексту та інтерактивних чатів. Загалом результати можна оцінити як задовільні: модель виконує свою функцію на базовому рівні і забезпечує основу для подальшого поліпшення та інтеграції більш потужних моделей у систему.В сі згенеровані відповіді мають ввічливу та дружню тональність, що важливо для чат-ботів соціального типу. Модель демонструє здатність формулювати фрази, що відповідають соціальним нормам та очікуванням користувача.

Висновки до третього розділу

У третьому розділі було розглянуто архітектуру системи рекомендацій, включно з побудовою REST API, механізмами безпеки та обробкою вхідних повідомлень, а також інтеграцію готової LLM для генерації відповідей. Проведений аналіз показав, що реалізовані модулі забезпечують структуровану та контрольовану взаємодію між клієнтськими застосунками та серверною частиною, гарантують захищений доступ до ресурсів і ефективну валідацію вхідних даних. Механізми видалення чутливої інформації та аналітична обробка повідомлень забезпечують безпеку і стабільність роботи системи, формуючи коректну основу для подальшої генерації рекомендацій. Інтегрована LLM показала здатність формувати логічні та контекстно релевантні відповіді, проте через використання відносно невеликої моделі не всі варіанти були ідеально точними або природними. Незважаючи на це, модель дозволяє генерувати кілька альтернативних варіантів відповіді з дружньою та ввічливою тональністю, що робить її придатною для демонстраційних або тестових сценаріїв. Загалом система демонструє задовільний рівень функціональності, забезпечуючи базову ефективність генерації рекомендацій та створюючи основу для подальшого масштабування і підключення більш потужних мовних моделей.

ВИСНОВКИ

У кваліфікаційній роботі було проведено комплексне дослідження проблематики автоматизації комунікаційних процесів і розроблено прототип системи генерації відповідей на повідомлення на основі історії переписки з використанням сучасних технологій обробки природної мови та машинного навчання.

У першому розділі встановлено актуальність автоматизації комунікаційних систем, зумовлену стрімким зростанням обсягів цифрових комунікацій, потребою у швидкому реагуванні на повідомлення та розвитком технологій штучного інтелекту. Проведений аналіз показав, що традиційні підходи до обробки користувацьких запитів втрачають ефективність у масштабних системах, що зумовлює впровадження інтелектуальних рішень на основі NLP та великих мовних моделей.

Розглянуто еволюцію чат-ботів – від простих систем із фіксованими сценаріями до інтелектуальних моделей на основі архітектури Transformer, які забезпечують новий рівень взаємодії з можливістю підтримки зв'язного діалогу, розуміння намірів користувача та персоналізації стилю комунікації.

У другому розділі сформульовано постановку задачі та обґрунтовано доцільність виокремлення механізму генерації відповідей у самостійний програмний модуль. Визначено архітектурні рішення, що базуються на багаторівневій клієнт-серверній архітектурі з REST API, контейнеризації через Docker та мікросервісному підході. Такий підхід забезпечує гнучкість, масштабованість та можливість інтеграції модуля в різні системи без необхідності повної перебудови їхньої архітектури.

Проаналізовано механізми отримання та обробки вхідних даних – як через прямий доступ до бази даних, так і через REST API. Визначено етапи попередньої обробки тексту, що включають нормалізацію, токенізацію, лематизацію та виділення ключових ознак, які формують основу для контекстного аналізу.

Для реалізації системи обрано технологічний стек на базі Python, Django, PostgreSQL та Git, який забезпечує швидку розробку, стабільність управління даними та ефективний контроль версій. Особливу увагу приділено бібліотеці Transformers від Hugging Face, яка надає доступ до сучасних передтренованих моделей глибокого навчання.

У третьому розділі детально описано реалізацію системи рекомендацій. Розроблено REST API з модулями авторизації, управління навчальними даними, повідомленнями та генерації рекомендацій, документованими за допомогою Swagger/OpenAPI. Реалізовано багаторівневі механізми безпеки, що включають токенову автентифікацію, валідацію структури запитів та захист від поширених типів атак.

Створено підсистему обробки вхідних повідомлень, яка складається з компонента видалення чутливої інформації (MessageSanitizer) та аналітичного модуля препроцесингу (AnalysingModule). Ці компоненти забезпечують відповідність вимогам безпеки даних і формують структуровану основу для генерації відповідей.

Інтегровано готову мовну модель через сервіс LLMService, який реалізує завантаження моделі та генерацію текстових відповідей з параметрами, що контролюють креативність та різноманітність результатів. Експериментальна перевірка продемонструвала здатність системи формувати контекстно релевантні відповіді з дружньою тональністю, хоча використання відносно невеликої моделі обмежує природність деяких варіантів.

Практична значущість роботи полягає у створенні функціонального прототипу, який може бути інтегрований у різні комунікаційні платформи – від корпоративних месенджерів до служб підтримки клієнтів. Модульна архітектура та контейнеризація забезпечують простоту розгортання, масштабування та підтримки системи.

Результати дослідження підтверджують ефективність застосування сучасних технологій обробки природної мови та великих мовних моделей для автоматизації комунікаційних процесів. Розроблена система демонструє

задовільний рівень функціональності і створює основу для подальшого вдосконалення шляхом інтеграції більш потужних моделей, розширення можливостей персоналізації та впровадження додаткових механізмів контекстного аналізу.

Напрямками подальших досліджень можуть бути: підключення більш потужних мовних моделей, реалізація механізмів донавчання на спеціалізованих датасетах, розробка системи оцінювання якості згенерованих відповідей, впровадження багатомовної підтримки та створення аналітичної підсистеми для моніторингу ефективності комунікацій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Комп'ютерно математичний огляд "The effects of information overload on online conversation dynamics" URL: https://link.springer.com/article/10.1007/s10588– 020– 09314– 9 (дата звернення: 05.09.2025)
2. Shailendra Kadre, Shailesh Kadre, Subhendu Dey Mastering Text Analytics : A Hands– on Guide to NLP Using Python, Apress, 2025. 513 с.
3. Bruno Goncalves Natural Language Processing (NLP) Fundamentals, 3rd Edition, Apress, 2019. 625 с.
4. Статистика обміну повідомленнями станом на 2024 рік URL: https://texting.io/sms– users– statistics/ (дата звернення: 06.09.2025)
5. Romilla Ready Kate Burton Neuro-linguistic Programming For Dummies, For Dummies, 2015. 448 с.
6. Tom Dotz, Tom Hoobyar «NLP: The Essential Guide to Neuro– Linguistic Programming», NLP Comprehensive, 2018. 480 с.
7. Charles Duhigg Supercommunicators: How to Unlock the Secret Language of Connection, Random House Trade Paperbacks, 2025. 320 с.
8. Impact of Modern Communication Channels on Business Processes (Žunac, Kocijan, Martinčević) URL: https://ojs.srce.hr/index.php/entrenova/article/view/20184 (дата звернення: 07.09.2025)
9. The Effect of WhatsApp Usage on Employee Innovative Performance at the Workplace: Perspective from the Stressor–Strain–Outcome Model URL: https://ojs.srce.hr/index.php/entrenova/article/view/20184 (дата звернення: 07.09.2025)
10. Impact of Modern Communication Channels on Business Processes (Žunac, Kocijan, Martinčević) URL: https://www.mdpi.com/2076– 328X/12/11/456 (дата звернення: 07.09.2025)
11. Consumers' Perceptions About Different Communication Channels in Bangladesh: A Comparative Study (Mamun, 2025) URL: https://www.sciencepublishinggroup.com/article/10.11648/j.innov.20250602.11 (дата звернення: 09.09.2025)
12. Sumit Raj Building Chatbots with Python: Using Natural Language Processing and Machine Learning, Apress, 2018. 211 с.
13. Amir Shevat Designing Bots: Creating Conversational Experiences 1st Edition, O'Reilly Media, 2017. 345 с.
14. Chat bot ELIZA URL: https://www.chatbots.org/paper/eliza_– _a_computer_program_for_the_study_of_natural_language_communication_ (дата звернення: 09.09.2025)
15. Andriy Burkov The Hundred–Page Machine Learning Book (The Hundred–Page Books), O'Reilly Media, 2019. 160 с.
16. Andreas C. Müller, Sarah Guido Introduction to Machine Learning with Python: A Guide for Data Scientists), O'Reilly Media, 2016. 398 с.

17. Anil Ananthaswamy Why Machines Learn: The Elegant Math Behind Modern AI, Dutton, 2024. 480 c.
18. Chip Huyen AI Engineering: Building Applications with Foundation Models, O'Reilly Media, 2025. 532 c.
19. Andriy Burkov Machine Learning Engineering, O'Reilly Media, 2020. 310 c.
20. John C. Lennox God, AI and the End of History: Understanding the Book of Revelation in an Age of Intelligent, SPCK Publishing, 2025. 608 c.
21. Kyle Faber NLP 2.0 - The Ultimate Guide to Neuro Linguistic Programming: How to Rewire Your Brain and Create the Life You Want and Become the Person You Were Meant to Be, O'Reilly Media, 2017. 89 c.
22. Duygu Altinok Mastering spaCy: An end–to–end practical guide to implementing NLP applications using the Python ecosystem, Packt Publishing, 2021. 372 c.
23. Hobson Lane, Maria Dyshel Natural Language Processing in Action, Second Edition, Manning, 2025. 688 c.
24. Benoît PRIEUR Traitement automatique du langage naturel avec Python – Le NLP avec spaCy et NLTK: Le NLP avec spaCy et NLTK, ENI, 2024. 277 c.
25. Aurélien Géron Hands–On Machine Learning with Scikit–Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 2022. 861 c.
26. Prem Timsina Building Transformer Models with PyTorch 2.0: NLP, computer vision, and speech processing with PyTorch and Hugging Face (English Edition), O'Reilly Media, 2024. 247 c.
27. Debu Sinha Practical Machine Learning on Databricks: Seamlessly transition ML models and MLOps on Databricks, Packt Publishing, 2023. 244 c.
28. James Gough, Daniel Bryant Mastering API Architecture: Design, Operate, and Evolve API–Based Systems, O'Reilly Media, 2022. 286 c.
29. JJ Geewax, Sarah D. API Design Patterns, O'Reilly Media, 2022. 153 c.
30. James Higginbotham Principles of Web API Design: Delivering Value with APIs and Microservices (Addison–Wesley Signature Series (Vernon)), Addison-Wesley Professional, 2021. 368 c.
31. Chris Richardson Microservices Patterns: With examples in Java, Manning, 2018. 520 c.
32. Sean P. Kane ,Karl Matthias Docker: Up & Running, O'Reilly Media, 2023. 327 c.
33. Jaime Buelta Hands–On Docker for Microservices with Python: Design, deploy, and operate a complex system with multiple microservices using Docker and Kubernetes, Packt Publishing, 2019. 158 c.
34. RICHES CECILIA The REST API Handbook : How to Build, Test, Consume, and Document REST APIs, O'Reilly Media, 2025. 574 c.
35. Matthias Biehl RESTful API Design: Best Practices in API Design with REST (API–University Series Book 3), O'Reilly Media, 2016. 327 c.

36. Bartosz Konieczny Data Engineering Design Patterns: Recipes for Solving the Most Common Data Engineering Problems, O'Reilly Media, 2025. 372 c.
37. Neal Ford, Mark Richards Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architecture,O'Reilly Media, 2021. 459 c.
38. StudioD21 Smart Tech Content 40 PYTHON LIBRARIES: An Essential Guide for Students and Professionals (Quick Learn Series Book 17), O'Reilly Media, 2025. 389 c.
39. Regina O. Obe, Leo S. Hsu PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database, O'Reilly Media, 2017. 312 c.
40. C. J. Date. Database Design and Relational Theory: Normal Forms and All That Jazz., O'Reilly Media, 2019. 451 c.
41. Toby J. Teorey, Sam S. Lightstone, Tom Nadeau, H.V. Jagadish. Database Modeling and Design, 4th Edition., O'Reilly Media, 2016. 384 c.
42. Jan L. Harrington. Relational Database Design and Implementation, 4th Edition., Morgan Kaufmann, 2016. 712 c.
43. Gerardus Blokdyk Database Normalization A Complete Guide - 2019 Edition, 5STARCooks, 2018. 307 c.
44. Michael J. Hernandez Database Design for Mere Mortals: 25th Anniversary Edition, Addison-Wesley Professional, 2020. 640 c.
45. Carlos Coronel, Steven Morris Database Systems: Design, Implementation, & Management, Cengage Learning, 2018. 816 c.
46. Qiang Hao, Michail Tsikerdekis Grokking Relational Database Design, Manning, 2025. 280 c.
47. Ambiorix Mora Database Normalization and Design: From Theory to the Efficient Model, O'Reilly Media, 2025. 66 c.
48. Gerardus Blokdyk Database normalization Second Edition, O'Reilly Media, 2018. 88 c.
49. Bill Karwin SQL Antipatterns: Avoiding the Pitfalls of Database Programming (Pragmatic Programmers), O'Reilly Media, 2017. 334 c.
50. Anna Skoulikari Learning Git: A Hands– On and Visual Guide to the Basics of Git, O'Reilly Media, 2023. 317 c.
51. Doug Hellmann Python 3 Standard Library by Example, The (Developer's Library) 2017. 1456 c.
52. David B. Python Distilled (Developer's Library), Pearson, 2021. 352 c.
53. RICHES CECILIA The REST API Handbook: How to Build, Test, Consume, and Document REST APIs, O'Reilly Media, 2025. 359 c.
54. Wei-Meng Lee Hugging Face in Action, Manning, 2025. 368 c.
55. Lewis Tunstall Natural Language Processing with Transformers: Building Language Applications with Hugging Face, O'Reilly Media, 2022. 406 c.
56. Calissa Corinne Hugging Face Transformers: A Step-by-Step Guide to Building NLP Applications with Python, O'Reilly Media, 2025. 317 c.
57. Henry Habib OpenAI API Cookbook: Build intelligent applications including chatbots, virtual assistants, and content generators, O'Reilly Media, 2024. 192 c.

58. Anne Gentle, Eric Holscher, Diane Skwish, Kelly Holcomb Docs Like Code: Collaborate and Automate to Improve Technical Documentation, O'Reilly Media, 2022. 132 с.

59. OpenAPI Initiative. OpenAPI Specification Documentation. 2024. URL: https://spec.openapis.org/oas/latest.html (дата звернення: 15.11.2025).

60. Swagger. Swagger Documentation – Designing and Documenting RESTful APIs. 2024. URL: https://swagger.io/docs/specification/v3_0/about/ (дата звернення: 15.11.2025).