**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Оксана ШОВКОПЛЯС
(підпис)

грудня 2024 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня магістр**

зі спеціальності 122 «Комп'ютерні науки»,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна технологія проектування чат-ботів з використанням штучного інтелекту»
здобувача групи ІН.м-34 Савченка Дмитра Сергійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Дмитро САВЧЕНКО
(підпис)

Керівник Альона МОСКАЛЕНКО
доцент, канд. техн. наук, доцент
(підпис)

**Суми – 2024**

**Сумський державний університет**

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Оксана ШОВКОПЛЯС
(підпис)

# ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

**на здобуття освітнього ступеня магістра**

зі спеціальності 122 «Комп'ютерні науки», освітньо-професійної програми «Інформатика» здобувача групи ІН.м–34 Савченка Дмитра Сергійовича

1. Тема роботи: «Інформаційна технологія проектування чат-ботів з використанням штучного інтелекту»
затверджую наказом по СумДУ *№1257-VI від 03.12.2024 року*
2. Термін здачі здобувачем кваліфікаційної роботи *до 06 грудня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд архітектури та методів реалізації. 3) Практична реалізація. 4) Висновок.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів кваліфікаційної роботи | Термін виконання | Примітка |
|---|---|---|---|
| 1 | Аналіз проблеми предметної області, постановка й формування завдань дослідження | 17.10-25.10.2024 | |
| 2 | Огляд архітектури та методів реалізації | 26.10-05.11.2024 | |
| 3 | Практична реалізація | 06.11-18.11.2024 | |
| 4 | Аналіз отриманих результатів | 19.11-21.11.2024 | |
| 5 | Оформлення пояснювальної записки до кваліфікаційної роботи | 22.11-28.11.2024 | |

Здобувач вищої освіти _____ (підпис)

Керівник _____ (підпис)

# АНОТАЦІЯ

**Записка**: 50 стор., 22 рис., 20 літературних джерел, 1 таблиця

**Об’єкт дослідження** — Інформаційна технологія проектування чат-ботів з використанням штучного інтелекту

**Мета роботи** — розробка інформаційної технології проектування чат-ботів, орієнтованих на використання штучного інтелекту.

**Результати** — проведено детальний аналіз сучасних алгоритмів та інструментів для створення чат-ботів з використанням штучного інтелекту. На основі проведеного огляду розроблено інформаційну систему, яка реалізована у вигляді чат-боту. Додаток був протестований та є готовим до загального використання.

ШТУЧНИЙ ІНТЕЛЕКТ, МОВНІ МОДЕЛІ, FASTAPI, BERT, AMAZON WEB SERVICES

**ЗМІСТ**

# ВСТУП

У сучасному світі стрімко зростає потреба в автоматизованих системах, які здатні забезпечувати ефективну комунікацію та оперативну підтримку користувачів. Однією з таких інновацій є чат-боти – програми, що дозволяють автоматизувати взаємодію між людьми та комп'ютерами. Завдяки своїй здатності обробляти та генерувати текстову інформацію в реальному часі, чат-боти значно полегшують процеси обслуговування клієнтів, освітнього супроводу, інформаційного забезпечення, а також виконують функції персональних асистентів. Зустріти чат-боти сьогодні можна в різних галузях: від онлайн-банкінгу до медичних консультантів, від навчальних програм до розважальних сервісів, що вказує на їхню важливу роль у сучасній цифровій екосистемі.

З розвитком штучного інтелекту (ШІ) та машинного навчання, чат-боти стали набагато складнішими й «розумнішими». Вони вже не обмежуються лише відповіді на стандартні запити, а можуть виявляти приховані наміри користувачів, аналізувати попередній досвід взаємодії та адаптувати свої відповіді на основі отриманих даних. Такі можливості забезпечують значно ширше поле для їхнього застосування, оскільки чат-боти можуть навчатися з часом, покращуючи свої комунікаційні здібності та підвищуючи рівень персоналізації.

Серед сучасних прикладів можна згадати голосового асистента Siri, створеного компанією Apple. Siri використовує технології обробки природної мови (NLP) та машинного навчання для розуміння й інтерпретації складних запитів користувачів. Вона здатна підтримувати розмову, відповідати на складні питання, допомагати в організації розкладу, а головне – постійно вдосконалюється через самонавчання. Це дозволяє Siri адаптуватися до потреб і вподобань користувачів, підвищуючи ефективність і зручність взаємодії. Цей приклад демонструє, як штучний

інтелект може значно розширити можливості чат-ботів, дозволяючи їм не тільки реагувати на прості запити, але й активно інтерпретувати наміри користувачів і забезпечувати багатогранний досвід взаємодії.

Метою цієї роботи є створення інформаційної системи, що зосереджена на проєктуванні чат-ботів з використанням інтелектуальних технологій ШІ. У рамках проєкту буде розглянуто різні методи машинного навчання, алгоритми обробки природної мови, проведено аналіз сучасних інструментів для створення та навчання чат-ботів, а також обрано оптимальні рішення для розробки високоякісного програмного продукту. Таким чином, ця система стане основою для створення інтелектуальних чат-ботів, здатних підтримувати інтерактивну та продуктивну комунікацію з користувачами на різних рівнях.

# 1 АНАЛІЗ ПРОБЛЕМИ

## 1.1 Технології обробки природної мови для чат-ботів

Обробка природної мови (Natural Language Processing, NLP) є критичною технологією для створення сучасних чат-ботів, що використовують штучний інтелект для комунікації з користувачами. NLP поєднує підходи комп'ютерних наук, лінгвістики та штучного інтелекту для обробки та аналізу текстових даних [2]. Основні задачі NLP в контексті чат-ботів охоплюють розпізнавання намірів користувача, виділення сутностей із тексту, побудову відповідей та аналіз контексту запиту.

### 1.1.1 Розпізнавання намірів (Intent Detection)

Розпізнавання намірів – це процес, за допомогою якого система визначає ціль користувача, який взаємодіє з чат-ботом. Намір (intent) – це основна причина, через яку користувач ініціює діалог. Наприклад, у банківському чат-боті серед намірів можуть бути «перевірити баланс», «здійснити переказ» або «змінити пароль» [1].

Для виконання задачі розпізнавання намірів застосовуються різні методи, серед яких класифікація тексту (метод опорних векторів (SVM), наївний баєсівський класифікатор або стохастичний градієнтний спуск), нейронні мережі (рекурентні нейронні мережі (RNN), LSTM (Long Short-Term Memory), GRU (Gated Recurrent Units)) та моделі трансформерів (BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pretrained Transformer) від OpenAI)

### 1.1.2 Виділення сутностей (Entity Recognition)

Виділення сутностей або розпізнавання іменованих сутностей (NER, Named Entity Recognition) – це процес виявлення важливих елементів у тексті, таких як дати, імена, адреси, суми грошей, що є необхідними для

виконання певних завдань у чат-боті. Наприклад, у запиті «переказати 500 гривень на рахунок Івана» бот має розпізнати суму та одержувача [3].

Основними підходами до виділення сутностей є методи на основі правил (використання регулярних виразів для пошуку шаблонів, таких як номери телефонів або адреси), статистичні моделі (умовні випадкові поля (Conditional Random Fields, CRF), приховані марковські моделі (HMM)), нейронні мережі та трансформери (архітектури BERT та BiLSTM-CRF).

### 1.1.3 Обробка контексту

Чат-боти, які працюють зі складними запитами, повинні враховувати попередні повідомлення, щоб зберігати контекст розмови. Це дає змогу ботам запам'ятовувати інформацію з попередніх частин діалогу та надавати релевантні відповіді на уточнення чи питання користувачів.

Для реалізації підтримки контексту використовуються різні методи. Наприклад, станова машина дозволяє визначати етапи розмови та змінювати стан бота в залежності від їхнього переходу, що ефективно для простих сценаріїв, але обмежено для динамічних чат-ботів. Використання RNN та LSTM моделей допомагає зберігати інформацію з попередніх кроків діалогу, хоча їхня ефективність падає при обробці довгих текстів [5].

Найсучаснішим підходом є застосування трансформерів, таких як GPT-3, які здатні зберігати та аналізувати великий обсяг тексту, забезпечуючи глибоке розуміння контексту для тривалих розмов. GPT-3 особливо ефективно справляється із завданнями, які вимагають обробки складних текстових запитів.

### 1.1.4 Генерація відповідей (Response Generation)

Для генерації відповідей можуть застосовуватися різні підходи, починаючи від використання фіксованих шаблонів і закінчуючи складними нейронними мережами. Шаблонні відповіді базуються на заздалегідь

підготовлених фразах, які відповідають певним запитам, що дозволяє швидко та ефективно обробляти стандартні ситуації.

Однак для складніших завдань використовуються алгоритми, які обирають найбільш відповідний варіант з набору готових відповідей. Такий підхід заснований на класифікації та враховує подібність між вхідним запитом і можливими варіантами відповідей, використовуючи евристики або векторизацію тексту.

Найбільш сучасним підходом є генеративні моделі, наприклад, Sequence-to-Sequence (Seq2Seq), які аналізують текст запиту та генерують нові відповіді. Використання трансформерів, таких як GPT, забезпечує створення відповідей з високим рівнем природності та адаптацією до стилю й тону діалогу.

### 1.1.5 Алгоритми оцінки та покращення точності NLP-моделей

Щоб забезпечити якість роботи NLP для чат-ботів, важливо правильно налаштувати моделі та оцінювати їх точність. Основні метрики для оцінки включають точність розпізнавання намірів, Точність розпізнавання сутностей, збереження контексту.

Процес оптимізації моделей включає підбір гіперпараметрів, використання розширених навчальних наборів даних та регуляризацію для покращення узагальнення моделей [9]. Часто використовуються ансамблеві методи, що комбінують декілька моделей для покращення кінцевого результату, зокрема їх застосовують для точнішого розпізнавання намірів та сутностей [7][8].

## 1.2 Аналітичний огляд платформ для створення чат-ботів

Розробка сучасних чат-ботів передбачає використання спеціалізованих платформ, що спрощують процес побудови, тестування та розгортання цих систем. Найпопулярнішими платформами, які

забезпечують функціонал для обробки природної мови, інтеграцію з різними каналами комунікації та масштабування, є IBM Watson Assistant, Google Dialogflow, Microsoft Bot Framework, Amazon Lex, а також відкритий фреймворк Rasa. Розглянемо можливості кожної з цих платформ.

### 1.2.1 IBM Watson Assistant

IBM Watson Assistant – це інтелектуальна платформа для створення чат-ботів, яка використовує штучний інтелект для обробки природної мови, розпізнавання намірів та аналізу контексту.

Платформа дозволяє чат-ботам точно ідентифікувати наміри користувачів та витягувати необхідну інформацію завдяки алгоритмам машинного навчання. Завдяки контекстній пам’яті Watson Assistant може підтримувати багатокрокові діалоги, забезпечуючи плавність розмови. Інтеграція з іншими сервісами IBM, такими як IBM Cloud та Watson Discovery, розширює можливості платформи, дозволяючи використовувати її для складних завдань, наприклад, пошуку у великих базах даних.

Платформа підтримує багато мов, що робить її ефективною для роботи з міжнародною аудиторією. Важливою функцією Watson Assistant є можливість вдосконалення діалогів на основі зворотного зв’язку, що сприяє адаптації чат-бота до потреб користувачів. Однак основним недоліком платформи є її висока вартість для масштабних проєктів [14].

### 1.2.2 Google Dialogflow

Dialogflow від Google – це потужний інструмент для створення чат-ботів, який виділяється завдяки інтеграції з екосистемою Google і розвинутій обробці природної мови.

Платформа забезпечує розпізнавання намірів і ключових сутностей у запитах користувачів, що дає змогу створювати ефективні розмовні сценарії. Вона також підтримує контекстне управління діалогами, що

дозволяє будувати багатокрокові послідовні розмови. Інтеграція з Google Assistant і різними сторонніми каналами комунікації значно розширює функціональність платформи, дозволяючи використовувати її для голосових команд.

Крім того, підтримка вебхуків забезпечує обробку складних запитів із доступом до баз даних або сторонніх сервісів. Dialogflow CX, розширена версія платформи, пропонує додаткові можливості для складних розмовних систем. Платформа характеризується зручним інтерфейсом і простою настройкою, однак витрати на обчислювальні ресурси можуть зрости при роботі з великими обсягами даних [12].

### 1.2.3 Microsoft Bot Framework

Microsoft Bot Framework – це платформа, яка надає розробникам інструменти для створення, тестування та розгортання чат-ботів із використанням технологій Microsoft.

Гнучка архітектура дозволяє використовувати кілька мов програмування, таких як C# та JavaScript, що робить платформу універсальною. Інтеграція з Microsoft Azure забезпечує масштабування, обробку даних та взаємодію із сервісами штучного інтелекту, зокрема Azure Language Understanding (LUIS).

Підтримка Omnichannel дозволяє інтегрувати чат-боти з популярними платформами, такими як Teams, Skype та Slack, розширюючи канали комунікації. Збереження контексту розмови забезпечується діалоговими компонентами, які дозволяють створювати інтерактивні сценарії з довготривалим використанням даних [13].

Платформа відзначається високим рівнем кастомізації і широким вибором API для інтеграції з корпоративними системами, але складність освоєння через велику кількість компонентів може стати викликом для новачків.

### 1.2.4 Amazon Lex

Amazon Lex – це інструмент від Amazon, розроблений для створення текстових і голосових чат-ботів, який використовує ті самі технології, що й голосовий помічник Alexa.

Сервіс інтегрується з іншими продуктами AWS, такими як AWS Lambda, Amazon S3 і DynamoDB, що дозволяє створювати складні логічні моделі для обробки запитів [15].

Lex підтримує як текстові, так і голосові інтерфейси, забезпечуючи універсальність застосування. Сервіс автоматично масштабується, адаптуючи свої ресурси до змін у завантаженні, що робить його ідеальним для проєктів із великим обсягом трафіку.

Amazon Lex є оптимальним вибором для розробників, які вже працюють із екосистемою AWS, однак високі витрати на тривале використання можуть обмежити його застосування для складних проєктів.

### 1.2.5 Rasa

Rasa – це платформа з відкритим кодом для створення чат-ботів, яка пропонує високий рівень гнучкості та кастомізації. Вона працює локально, що гарантує безпеку даних і захищеність від зовнішніх загроз. Rasa складається з двох основних компонентів: Rasa NLU для обробки природної мови і Rasa Core для управління діалогами, що дозволяє точно налаштовувати кожен із них.

Платформа підтримує інтеграцію кастомних моделей і алгоритмів, що забезпечує адаптацію для специфічних завдань. Вона дозволяє створювати складні діалоги із збереженням контексту, враховуючи історію користувачів.

Rasa є оптимальним вибором для проєктів, що потребують повного контролю над функціоналом, особливо в умовах обмежених ресурсів або

підвищених вимог до безпеки. Основним недоліком є необхідність технічних навичок для початкового налаштування і розробки власних моделей.

В таблиці 1.1 наведено порівняльну характеристику існуючих мовних асистентів в контексті їх сильних та слабких сторін.

Таблиця 1.1 – Порівняльна характеристика існуючих мовних асистентів

| **Платформа** | **Сильні сторони** | **Слабкі сторони** |
|---|---|---|
| IBM Watson Assistant | Інтеграція з IBM Cloud, гнучка обробка намірів | Висока вартість використання |
| Google Dialogflow | Легка інтеграція, підтримка Google Assistant | Обмеження у складних запитах для великих обсягів |
| Microsoft Bot Framework | Інтеграція з Azure, гнучкість, підтримка Omnichannel | Складність освоєння для новачків |
| Amazon Lex | Голосові інтерфейси, інтеграція з AWS | Висока вартість для тривалих діалогів |
| Rasa | Повний контроль, локальна обробка, підтримка кастомних моделей | Потреба у технічних знаннях для налаштування |

## 1.3 Використання алгоритмів машинного навчання та штучного інтелекту для чат-ботів

Алгоритми машинного навчання та штучного інтелекту є основою для створення інтелектуальних чат-ботів, здатних до глибокого аналізу тексту, адаптивної взаємодії з користувачами та самонавчання.

Використання таких алгоритмів дозволяє чат-ботам виконувати складні функції, від розпізнавання намірів і виділення сутностей до побудови діалогів та аналізу емоційного забарвлення тексту.

### 1.3.1 Алгоритми для розпізнавання намірів і виділення сутностей

Для того, щоб чат-бот міг коректно інтерпретувати запит користувача, необхідно використовувати ефективні алгоритми для визначення намірів і виділення ключових сутностей.

Одним із підходів є застосування методів класифікації тексту, таких як метод опорних векторів, наївний баєсівський класифікатор та логістична регресія. Ці алгоритми добре працюють у випадках простих завдань, але для складніших сценаріїв частіше використовують нейронні мережі. Рекурентні нейронні мережі (RNN) забезпечують збереження інформації про попередні елементи у текстовій послідовності, що робить їх особливо корисними для аналізу тексту.

Проте їхня схильність до втрати контексту при роботі з довгими текстами обмежує ефективність. Для вирішення цієї проблеми використовуються вдосконалені модифікації RNN, такі як LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Units), які завдяки механізмам управління інформацією дозволяють краще зберігати контекст навіть у довгих послідовностях [8].

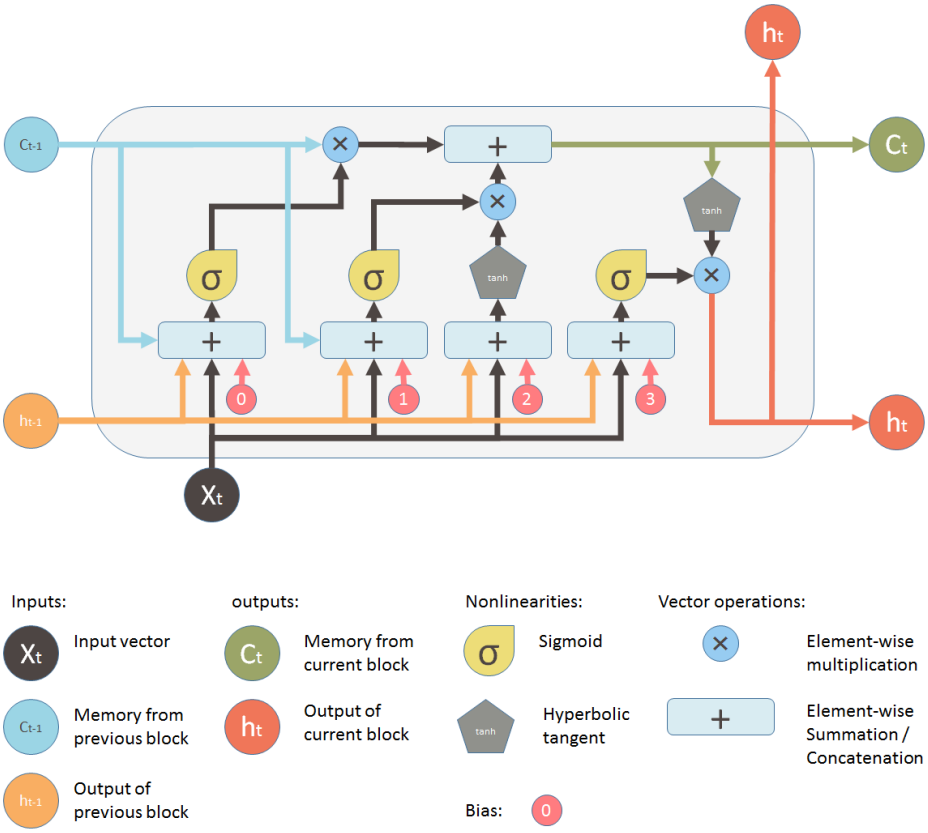


Рисунок 1.1 – Схема алгоритму LSTM

### 1.3.2 Генеративні та класифікаційні моделі для побудови відповідей

Для створення відповідей у чат-ботах використовуються різні підходи, які підбираються залежно від складності завдання та доступних ресурсів системи.

Одним із поширених методів є використання класифікаційних моделей, де бот вибирає найкращу відповідь із заздалегідь підготовленого набору на основі аналізу запитів користувача. Ці моделі застосовують алгоритми, такі як логістична регресія або стохастичний градієнтний спуск, для визначення відповідності запиту конкретній відповіді. Іншим підходом є архітектура Sequence-to-Sequence (Seq2Seq), яка складається з енкодера для обробки вхідного тексту та декодера для генерації відповіді.

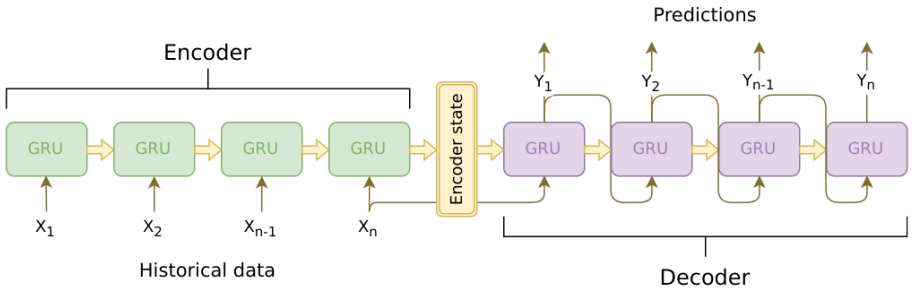


Рисунок 1.2 – Архітектура seq2seq

Цей метод забезпечує природність відповідей, особливо у простих сценаріях, хоча він може виявитися менш ефективним для довгих або складних діалогів.

Такі моделі здатні створювати контекстуалізовані відповіді, враховуючи попередній хід розмови. Наприклад, GPT-3 демонструє високу здатність до підтримки складних діалогів, генеруючи відповіді, адаптовані до конкретних запитів. Проте використання таких моделей потребує значних обчислювальних ресурсів через складність обробки великих обсягів даних.

### 1.3.3 Обробка емоційного забарвлення та тональності тексту

Аналіз емоційного забарвлення тексту дає змогу чат-ботам адаптувати свої відповіді до настрою користувача, підвищуючи ефективність і якість взаємодії. Одним із основних підходів є сентимент-аналіз, який дозволяє класифікувати емоційне забарвлення тексту як позитивне, негативне або нейтральне.

Цей метод часто базується на алгоритмах, таких як метод опорних векторів, наївний баєсівський класифікатор або глибокі нейронні мережі, і використовується для зміни стилю відповідей відповідно до настрою користувача. Інший важливий підхід – тональний аналіз із використанням трансформерів, зокрема моделей BERT і GPT, які завдяки здатності

враховувати контекст можуть аналізувати тональність тексту як загалом, так і окремих його частин.

Це дозволяє ботам адекватно реагувати навіть на складні емоційні запити. Крім того, для визначення конкретних емоцій, таких як гнів, радість, сум чи страх, застосовуються емоційні класифікатори.

Ці моделі часто базуються на нейронних мережах, таких як LSTM та GRU, або трансформерах, і сприяють створенню більш персоналізованої взаємодії з користувачем [10].

### 1.3.4 Алгоритми самонавчання та вдосконалення моделей

Самонавчання є ключовим елементом у створенні чат-ботів на основі штучного інтелекту, що дозволяє їм адаптуватися до нових даних, коригувати свою поведінку і вдосконалювати відповіді в режимі реального часу.

Один із підходів передбачає використання механізму Active Learning, коли бот передає складні або невизначені запити для перевірки експертами, а потім навчається на основі цих даних, підвищуючи точність визначення намірів і сутностей [20].

Інший підхід базується на зборі зворотного зв’язку від користувачів, коли система враховує оцінки або коментарі щодо якості відповідей. Це дозволяє автоматично вдосконалювати поведінку бота відповідно до отриманих оцінок.

Також широко використовується підкріплювальне навчання, яке ґрунтується на механізмі винагород і покарань за певні дії. Завдяки цьому методу боти можуть адаптуватися до нових умов і поступово вдосконалювати свої рішення, що особливо корисно для інтерактивних систем, які працюють у змінних середовищах.

### 1.3.5 Використання ансамблевих методів для підвищення точності

Ансамблеві методи забезпечують можливість комбінування кількох моделей для досягнення більш точних результатів у завданнях, таких як розпізнавання намірів і виділення сутностей.

Для підвищення ефективності в чат-ботах часто використовується підхід голосування, коли кілька моделей працюють над аналізом запиту, а фінальна відповідь обирається на основі більшості голосів. Це забезпечує стабільність системи навіть у разі неточностей однієї з моделей.

Ще одним методом є накладання, де результати первинних моделей обробляються додатковою моделлю для більш глибокого аналізу і формування остаточного рішення, що дозволяє враховувати різні аспекти даних. Також популярними є техніки Bagging і Boosting.

У першому випадку комбінування моделей знижує вплив шуму та випадкових помилок, тоді як Boosting акцентує увагу на складних для розпізнавання запитах, поступово підвищуючи точність роботи системи.

## 1.4 Постановка задачі

Проєктування інформаційної системи для створення інтелектуальних чат-ботів на базі штучного інтелекту зосереджено на вирішенні завдань автоматизованої взаємодії між користувачем і системою. Цей процес охоплює всі етапи розробки, починаючи від створення моделей обробки природної мови і завершуючи інтеграцією з іншими сервісами для забезпечення повноцінної функціональності.

Одним із ключових завдань є розпізнавання намірів користувачів. Це передбачає розробку алгоритмів, здатних точно ідентифікувати наміри, навіть якщо запити є нестандартними або неоднозначними. Для цього застосовуються моделі, такі як LSTM, GRU або трансформери, які

забезпечують адаптивність і високу точність обробки.

Ще одним важливим завданням є виділення сутностей, тобто специфічної інформації в тексті, наприклад, імен, дат, чисел або місць. Для цього використовуються умовні випадкові поля, а також сучасні моделі на основі BERT чи GPT, що забезпечують точність у різних контекстах. Крім того, чат-бот повинен бути здатним зберігати контекст розмови, що дає змогу вести багатокрокові діалоги, адекватно інтерпретуючи запити, які залежать від попередніх повідомлень. Управління контекстом забезпечується за допомогою таких методів, як побудова діалогових ланцюжків із використанням RNN, LSTM або трансформерів.

Система також повинна генерувати адаптивні відповіді, які не тільки надають потрібну інформацію, але й враховують емоційний стан користувача чи контекст розмови. Для цього використовуються генеративні моделі, наприклад, GPT-3, які забезпечують природність і варіативність відповідей. Окреме завдання полягає у визначенні емоційного забарвлення тексту, що дозволяє адаптувати відповіді залежно від настрою користувача.

Інтеграція з різними каналами комунікації, такими як Telegram, Facebook Messenger або вебсайти, забезпечує зручність використання і доступність чат-ботів для широкої аудиторії. Окрім цього, інтеграція включає підтримку вебхуків, що дозволяє підключатися до зовнішніх баз даних і API, забезпечуючи точність і актуальність відповідей. Важливим аспектом є здатність до самонавчання. Це передбачає вдосконалення моделей на основі зворотного зв’язку користувачів через активне навчання або підкріплювальне навчання, що дозволяє системі адаптуватися до нових сценаріїв і запитів.

Серед додаткових вимог є забезпечення масштабованості та продуктивності, що дає змогу обробляти велику кількість запитів із

мінімальними затримками. Для цього необхідно оптимізувати моделі та обирати ефективні обчислювальні ресурси. Особливу увагу слід приділити безпеці та конфіденційності даних, особливо коли йдеться про обробку персональної інформації.

Це включає шифрування даних та впровадження протоколів аутентифікації. Гнучкість налаштувань і можливість масштабування також є важливими, оскільки система може використовуватися в різних галузях, і має бути забезпечена можливість додавання нових функцій або сценаріїв без значних змін у її основній структурі.

Результатом розробки має стати інформаційна система, яка автоматизує створення адаптивних і функціональних чат-ботів, здатних вести діалог на високому рівні. Використання штучного інтелекту та машинного навчання забезпечить гнучкість і персоналізацію, що зробить систему придатною для різних сфер застосування. Інтеграція з сучасними каналами комунікації сприятиме розширенню доступності чат-ботів і їхній ефективній інтеграції в бізнес-процеси.

# 2 ОГЛЯД АРХІТЕКТУРИ ТА МЕТОДІВ РЕАЛІЗАЦІЇ

## 2.1. Вибір архітектури

Для створення інформаційної системи чат-ботів важливо обрати таку архітектуру, яка б відповідала сучасним вимогам до масштабованості, продуктивності та гнучкості. Основні архітектурні підходи, що широко застосовуються у розробці подібних систем, включають монолітну, мікросервісну, івент-орієнтовану та безсерверну архітектури. Кожен із цих підходів має свої переваги та недоліки, які слід враховувати при розробці системи.

### 2.1.1. Монолітна архітектура

Монолітна архітектура передбачає інтеграцію всіх компонентів системи в єдину програмну структуру. Усі функції – від обробки запитів до управління базою даних – виконуються як частина одного додатка [15].

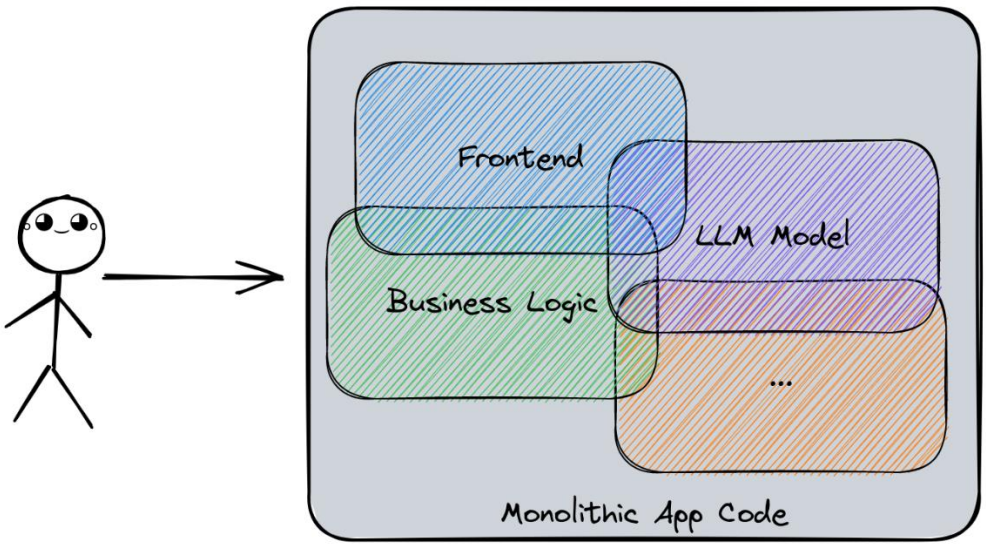


Рисунок 2.1 – Монолітна архітектура в контексті використання Generative AI

Перевагами такої архітектури є проста розробка та розгортання, єдина

точка управління, що спрощує забезпечення цілісності системи.

Водночас, недоліками монолітної архітектури є важкість внесення змін до окремих частин системи, обмежена масштабованість та залежність компонентів один від одного.

Цей підхід добре підходить для створення початкових версій чат-ботів, коли основна мета – швидке розгортання.

### 2.1.2. Мікросервісна архітектура

У мікросервісній архітектурі система складається з невеликих незалежних модулів, які виконують окремі функції. Кожен сервіс розробляється, тестується та масштабується автономно.

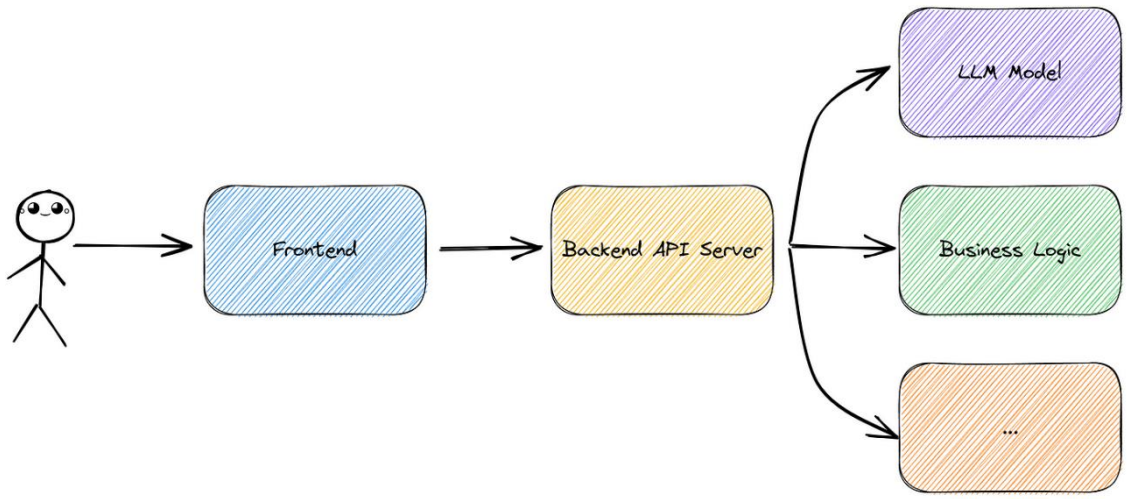


Рисунок 2.2 – Мікросервісна архітектура у контексті Generative AI

Якщо казати про переваги мікросервісної архітектури, то ними є гнучке масштабування окремих компонентів, використання різних технологій для кожного модуля, легке впровадження нових функцій.

З недоліків такої архітектури можна виділити складність координації між сервісами, необхідність ретельного управління їхньою комунікацією,

збільшення витрат на підтримку інфраструктури.

Ця архітектура доцільна для великих та складних систем, де потрібна висока надійність і продуктивність.

### 2.1.3. Подіє-орієнтована архітектура

Подіє-орієнтований підхід (event-driven) орієнтований на обробку подій: система реагує на певні дії або зміни в середовищі.

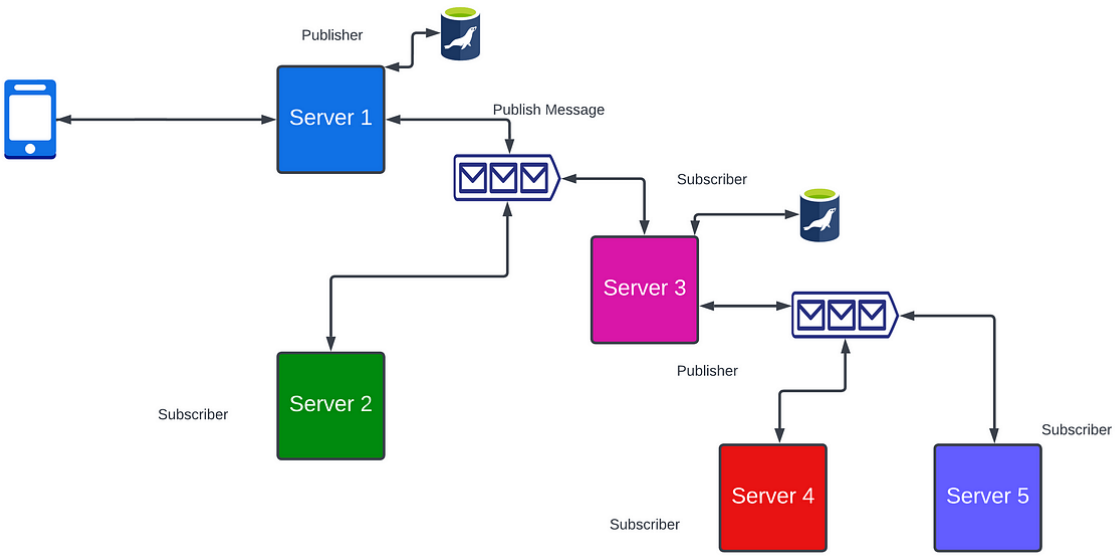


Рисунок 2.3 – Типова діаграма застосунку з використанням event-driven архітектури

Серед переваг такої архітектури можна виділити високу адаптивність, можливість інтеграції з іншими технологіями, зниження навантаження на центральний сервер.

Проте, є і такі недоліки, як складність забезпечення надійності обробки подій, потенційні затримки при великій кількості одночасних запитів.

Цей підхід найкраще підходить для систем, які потребують швидкої реакції на події, наприклад, чат-ботів, що обслуговують велику кількість користувачів у реальному часі.

### 2.1.4. Безсерверна архітектура

Безсерверна архітектура дозволяє запускати функціональні компоненти системи у хмарному середовищі, при цьому ресурси автоматично масштабуються відповідно до потреб [11].

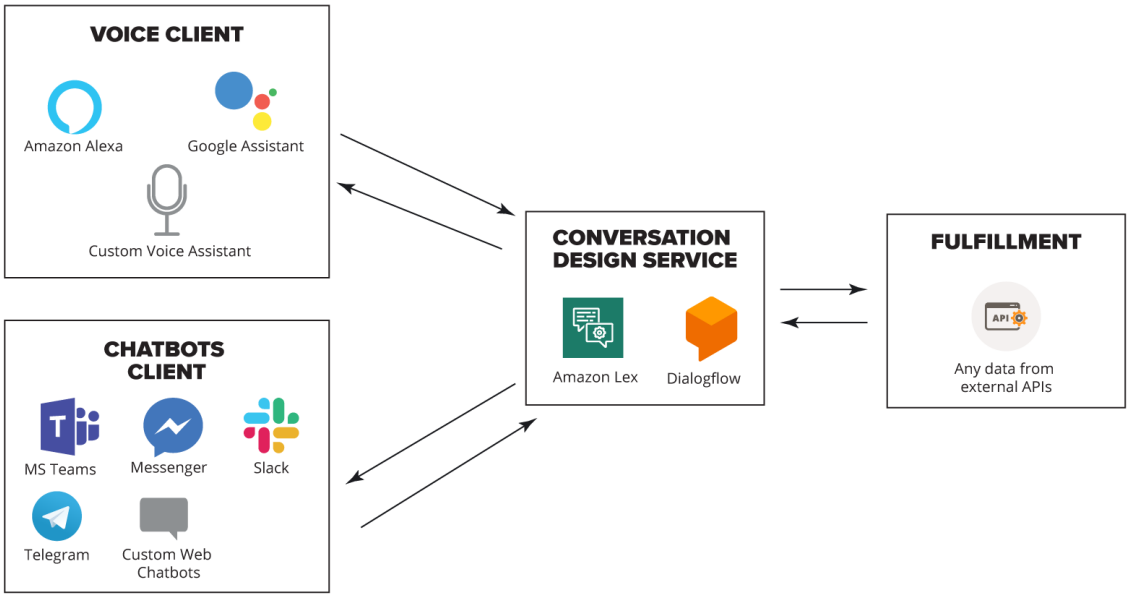


Рисунок 2.4 – Безсерверна архітектура для чат-боту з використанням веб-сервісів Amazon

Перевагами вважаються зниження витрат на інфраструктуру, гнучке масштабування, автоматичне управління ресурсами.

Тим не менш, є і недоліки: залежність від постачальника хмарних послуг, можливі затримки під час запуску функцій, труднощі з локальним тестуванням.

Цей варіант є ефективним для систем із нерівномірним навантаженням та динамічною інфраструктурою.

### 2.1.5. Вибір архітектури

Для реалізації чат-бота найбільш оптимальним є комбінований підхід, який враховує переваги різних архітектур. На початкових етапах розробки доцільно використати монолітну архітектуру, що дозволить швидко створити та протестувати прототип. Після перевірки основної функціональності варто перейти до мікросервісної архітектури, яка забезпечить масштабованість та гнучкість системи. Івент-орієнтовані елементи допоможуть реалізувати обробку запитів у реальному часі, а використання хмарних технологій (serverless) дозволить динамічно управляти навантаженням.

Така архітектурна модель забезпечить стабільність роботи, можливість адаптації до змінних вимог користувачів та високу ефективність обробки запитів.

## 2.2. Аналітичний огляд мови програмування Python та інструментів для розробки

Для реалізації інформаційної системи чат-ботів я обрав мову програмування Python завдяки її гнучкості, універсальності та великій кількості спеціалізованих бібліотек. Python є оптимальним вибором для обробки природної мови, створення API та інтеграції з сучасними інструментами машинного навчання [4].

Причинами вибору саме Python в якості мови програмування є:

1. **Екосистема бібліотек.** Python забезпечує доступ до широкого

спектра інструментів для роботи з NLP, машинним навчанням та API, зокрема Hugging Face Transformers, SpaCy, TensorFlow і PyTorch.

2. **Простота та швидкість розробки.** Завдяки читабельному синтаксису Python дозволяє прискорити процес написання та тестування коду.
3. **Підтримка асинхронності.** Python надає інструменти для асинхронного виконання задач, що є критично важливим для обробки великої кількості запитів у реальному часі.
4. **Поширеність і підтримка.** Розвинена спільнота розробників забезпечує доступ до документації, прикладів коду та готових рішень.

Для успішної розробки даної інформаційної системи було обрано такі основні інструменти:

1. **FastAPI.** FastAPI є сучасним фреймворком для створення високопродуктивних API, орієнтованих на асинхронність і масштабованість. Він здатен автоматично генерувати документацію API за допомогою OpenAPI та використовуючи Swagger, що спрощує тестування та інтеграцію. Окрім цього, він легко інтегрується з Pydantic для перевірки та серіалізації даних, що підвищує надійність і зручність роботи з запитами.
2. **SpaCy.** Дана бібліотека призначена для виконання базових NLP-задач, таких як токенізація, розпізнавання сутностей і синтаксичний аналіз. Також вона підтримує швидку обробку текстів великих обсягів робить її оптимальним вибором для продуктивних застосунків.
3. **Transformers (Hugging Face).** Цей інструмент забезпечує доступ до сучасних моделей трансформерів (GPT, BERT) із попередньо навченими вагами. В основному використовується для задач, пов’язаних із розпізнаванням намірів, генерацією тексту та аналізом

емоційного забарвлення.

4. **SQLAlchemy.** ORM-бібліотека, що дозволяє працювати з базами даних у Python-коді, зменшуючи потребу писати SQL-запити вручну. Вона підтримує складні транзакції та масштабованість системи.
5. **PostgreSQL.** Реляційна база даних із високим рівнем надійності та продуктивності. Використовуватиметься для зберігання історії діалогів, даних користувачів і метаінформації про взаємодію.
6. **Celery.** Інструмент для асинхронного виконання задач, що дозволяє розподіляти навантаження між компонентами системи. Використовуватиметься для виконання обчислювально складних задач у фоновому режимі.
7. **RabbitMQ.** Брокер повідомлень, який забезпечує передачу даних між компонентами системи та управління чергами завдань. Дозволяє досягти високої стійкості та масштабованості системи.

Зпираючись на попередній огляд, було обрано такі інструменти для створення інформаційної системи:

- **FastAPI** для створення RESTful API.
- **Transformers (Hugging Face)** для роботи з NLP-завданнями.
- **SQLAlchemy** та **PostgreSQL** для організації бази даних.
- **Celery** та **RabbitMQ** для асинхронного виконання завдань.

Обраний набір забезпечує швидку розробку, високу продуктивність і гнучкість у масштабуванні системи, що відповідає вимогам до сучасних інформаційних технологій.

# 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

## 3.1. Проєктування бази даних

Для створення інформаційної системи чат-ботів необхідно спроєктувати базу даних, яка забезпечить збереження структурованих даних про користувачів, їхні діалоги, повідомлення, наміри та сутності, розпізнані в тексті. Головна мета такого проєктування – гарантувати зручний доступ до даних, гнучкість у їх обробці, а також можливість масштабування системи відповідно до зростаючих потреб.

Розробка бази даних передбачає врахування таких вимог:

- забезпечення зберігання інформації про користувачів системи;
- збереження інформації про сеанси взаємодії (конверсії) між користувачем і чат-ботом;
- збереження повідомлень разом із метаданими (час, відправник, тип повідомлення);
- зберігання розпізнаних намірів (intents) і сутностей (entities), що були ідентифіковані в повідомленнях;
- встановлення зв’язків між повідомленнями, намірами та сутностями для аналізу даних та вдосконалення моделей.

На основі цих вимог було розроблено логічну модель бази даних. Вона містить такі основні сутності:

1. **User** – таблиця, що зберігає інформацію про користувачів системи.
    - *user_id* – унікальний ідентифікатор користувача.
    - *username* – ім’я користувача.
    - *email* – електронна адреса.
    - *created_at* – дата та час реєстрації.

2. **Conversation** – таблиця для збереження інформації про діалоги користувача з чат-ботом.
    - *conversation_id* – унікальний ідентифікатор сеансу.
    - *user_id* – посилання на користувача, який ініціював діалог.
    - *started_at* – дата та час початку діалогу.
    - *ended_at* – дата та час завершення діалогу.
3. **Message** – таблиця, яка зберігає інформацію про всі повідомлення.
    - *message_id* – унікальний ідентифікатор повідомлення.
    - *conversation_id* – зв’язок із діалогом, до якого належить повідомлення.
    - *sender* – відправник повідомлення (користувач або бот).
    - *content* – текст повідомлення.
    - *sent_at* – дата та час відправлення.
4. **Intent** – таблиця для зберігання інформації про наміри.
    - *intent_id* – унікальний ідентифікатор наміру.
    - *name* – назва наміру.
    - *description* – опис призначення наміру.
5. **Entity** – таблиця для зберігання розпізнаних сутностей.
    - *entity_id* – унікальний ідентифікатор сутності.
    - *name* – назва сутності.
    - *type* – тип сутності (наприклад, дата, ім’я, адреса).
6. **MessageIntent** – таблиця для збереження зв’язків між повідомленнями та намірами.
    - *message_id* – посилання на повідомлення.
    - *intent_id* – посилання на намір.
    - *confidence* – рівень впевненості системи у визначеному намірі.
7. **MessageEntity** – таблиця для збереження зв’язків між повідомленнями та сутностями.
    - *message_id* – посилання на повідомлення.

- *entity_id* – посилання на сутність.
- *value* – значення сутності, визначене у тексті.

Для візуалізації взаємозв’язків між сутностями бази даних було створено діаграму (рисунок 3.2). Ця діаграма демонструє, як різні елементи бази даних пов’язані між собою. Усі ключові атрибути, зв’язки та залежності між таблицями деталізовано для забезпечення прозорості проєктування.

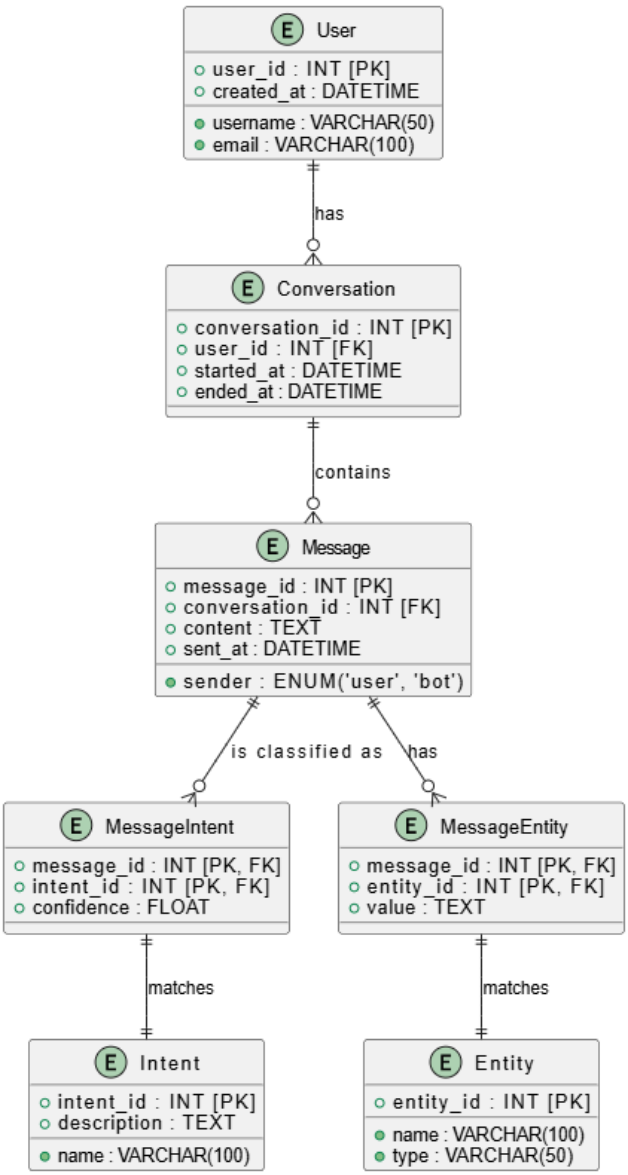


Рисунок 3.1 – Діаграма бази даних

### 3.2. Проєктування інформаційної системи

Проєктування інформаційної системи є одним із ключових етапів розробки, адже саме на цьому етапі визначається архітектура, вибираються основні компоненти, способи їх взаємодії, а також інтеграція із зовнішніми сервісами. Основною метою цього етапу є створення системи, яка забезпечить стабільну обробку даних, інтерактивну взаємодію з користувачами, а також простоту інтеграції з популярними месенджерами, такими як Telegram та Facebook Messenger.

Система спроєктована відповідно до багаторівневої архітектури, яка включає:

1. **Клієнтський рівень** – взаємодія з користувачами через мобільний або веб-додаток.
2. **Серверний рівень** – виконання бізнес-логіки та обробки даних.
3. **Рівень зовнішніх сервісів** – інтеграція із зовнішніми платформами обміну повідомленнями.

Реалізована структура дозволяє розділити функціональність системи між окремими рівнями, що спрощує її підтримку, масштабування та інтеграцію з іншими сервісами. Всі компоненти системи, їх взаємозв’язки та потоки даних відображені на діаграмі компонентів (рисунок 3.2).

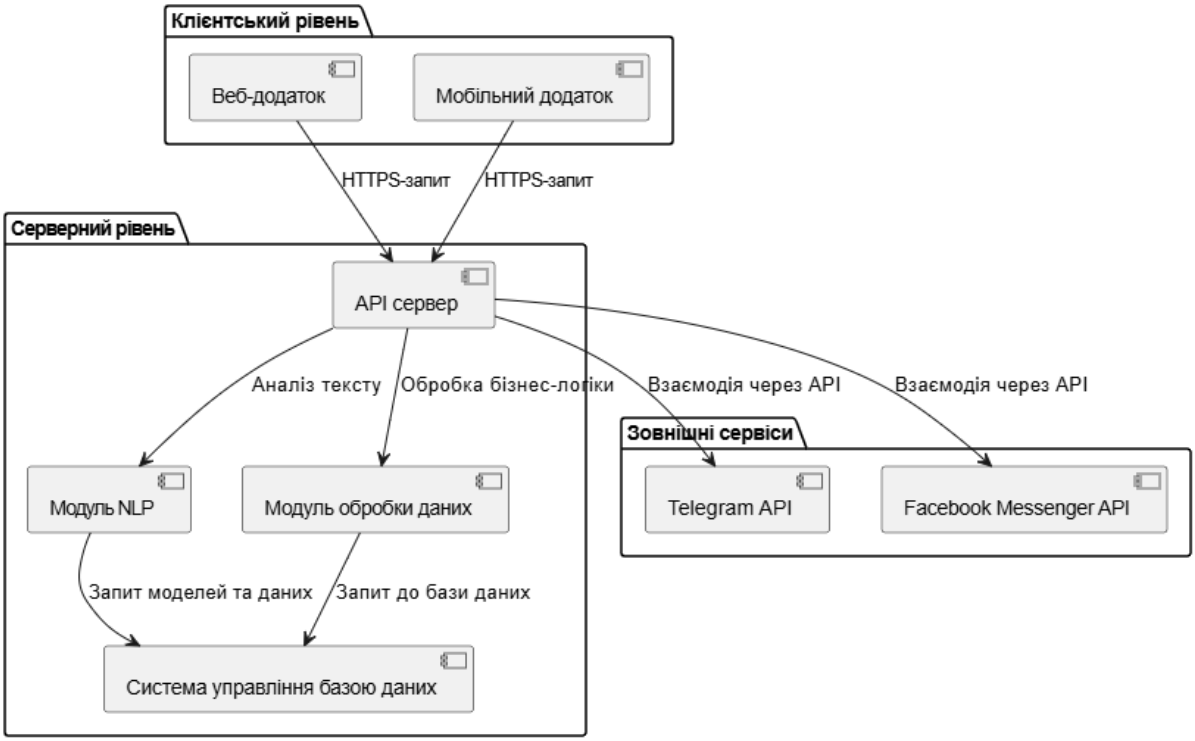


Рисунок 3.2 – Діаграма компонентів системи

Основними компонентами системи є:

1. **Клієнтський рівень.** Веб-додаток є ключовим компонентом системи, який забезпечує доступ користувачів через браузер. Він виконує функцію інтерфейсу для взаємодії з чат-ботом, дозволяючи надсилати повідомлення та отримувати відповіді. Крім того, цей компонент відповідає за зручність користування, відображення результатів обробки та створення позитивного користувацького досвіду. Мобільний додаток, у свою чергу, виступає адаптованою версією інтерфейсу, розробленою спеціально для використання на смартфонах та планшетах. Він забезпечує доступ до функцій чат-бота на мобільних пристроях, враховуючи їхні особливості, та гарантує комфортну взаємодію користувачів із системою у мобільному середовищі.
2. **Серверний рівень.** API-сервер є центральним компонентом

архітектури, який приймає запити від клієнтів, таких як мобільний або веб-додаток, передає їх на обробку до відповідних модулів і повертає результати. Він також виконує функцію формування відповідей для зовнішніх платформ, забезпечуючи інтеграцію з іншими системами. Модуль обробки природної мови (NLP) займається аналізом текстових даних, отриманих від користувачів, і на основі моделей машинного навчання виконує розпізнавання намірів, виділення сутностей та аналіз контексту повідомлень. Обробка даних здійснюється окремим модулем, який реалізує основну бізнес-логіку. Він працює з інформацією, отриманою від NLP-модуля, виконує необхідні операції та обробляє запити до бази даних. Система управління базою даних зберігає всю інформацію, пов’язану з користувачами, діалогами, повідомленнями, намірами та сутностями, забезпечуючи швидкий доступ до даних для подальшої обробки і підтримки функціональності системи.

3. **Рівень зовнішніх сервісів.** Telegram API забезпечує інтеграцію системи з месенджером Telegram, дозволяючи обмінюватися повідомленнями між користувачами та чат-ботом. Завдяки цьому компоненти користувачі можуть взаємодіяти з ботом безпосередньо через інтерфейс Telegram, що значно підвищує зручність і доступність використання. Facebook Messenger API виконує аналогічну функцію, забезпечуючи інтеграцію з платформою Facebook Messenger. Це дозволяє користувачам взаємодіяти з чат-ботом безпосередньо у цьому популярному месенджері, відкриваючи додаткові канали комунікації для системи.

Взаємодія між компонентами системи відбувається за чітко визначеною послідовністю. Алгоритм роботи виглядає так:

1. Користувач надсилає повідомлення через мобільний або веб-додаток, або ж через месенджер (Telegram чи Facebook Messenger).
2. Повідомлення передається API серверу, який обробляє запит і спрямовує текст до NLP-модуля.
3. NLP-модуль аналізує текст, визначає наміри користувача, виділяє сутності та виконує інші операції з текстовими даними.
4. Для отримання додаткових даних або збереження результатів обробки, модулі взаємодіють із системою баз даних.
5. На основі результатів аналізу API сервер формує відповідь, яка повертається користувачеві через мобільний, веб-додаток або месенджер.

Цей процес детально відображений на діаграмі послідовності (рисунок 3.3), яка демонструє взаємодію між усіма компонентами системи.

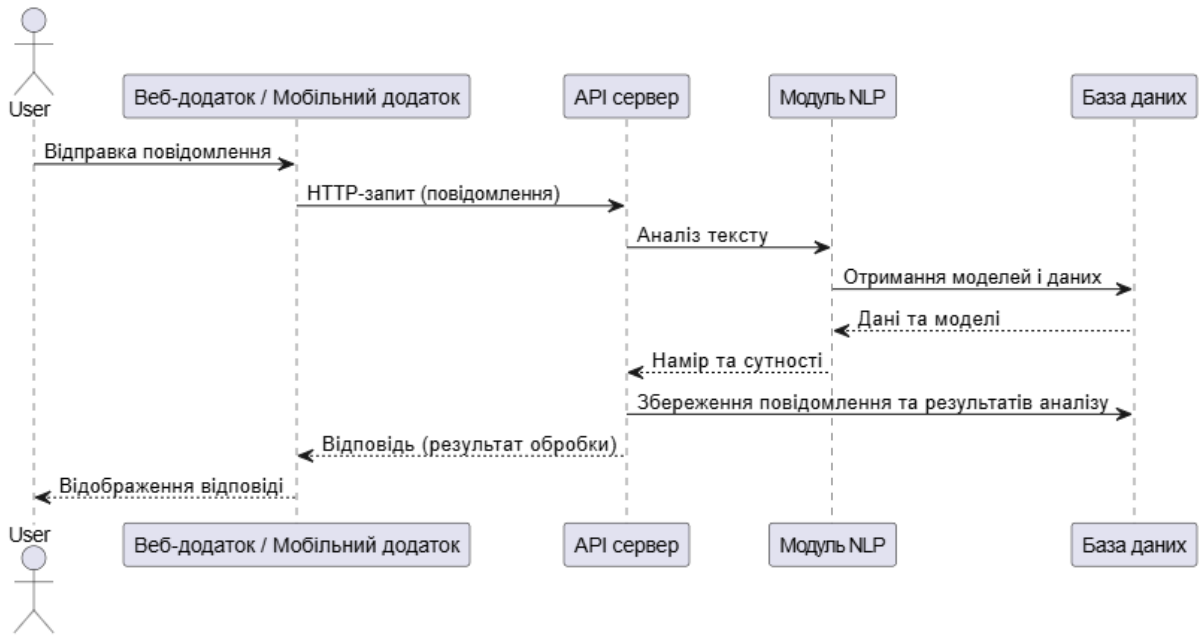


Рисунок 3.3 – Діаграма послідовності інформаційної системи

Інтеграція із зовнішніми платформами обміну повідомленнями дозволяє користувачам взаємодіяти з чат-ботом через знайомі їм

інтерфейси. Для реалізації цієї функції було використано Telegram API та Facebook Messenger API, які дозволяють приймати та надсилати повідомлення через відповідні месенджери. Такий підхід значно розширює доступність чат-бота для користувачів.

### 3.3. Програмна реалізація

Програмна реалізація інформаційної системи охоплює створення API-сервера, модуля обробки природної мови, модуля роботи з базою даних та інтеграцію із зовнішніми сервісами. У цьому розділі детально описано реалізацію всіх основних компонентів системи, а також наведено приклади коду для кожного з них.

API-сервер виконує роль посередника між клієнтом (мобільним або веб-додатком) та іншими модулями системи. Використання FastAPI забезпечує швидкість розробки, асинхронну обробку запитів і автоматичну генерацію документації. На рисунку 3.4 представлено код файлу main.py, який є основним для вдалого запуску та налаштування серверу.

```python
from fastapi import FastAPI
from routes import user_routes, message_routes

app = FastAPI(title="Chatbot API", description="API для взаємодії з чат-ботом")

# Підключення маршрутів
app.include_router(user_routes.router, prefix="/users", tags=["Users"])
app.include_router(message_routes.router, prefix="/messages", tags=["Messages"])

@app.get("/")
async def root():
    return {"message": "Chatbot API is running"}
```

Рисунок 3.4 – Код файлу main.py

Даний код створює основний сервер. Параметри title і description визначають метаінформацію для автоматично згенерованої документації

API. Маршрути підключаються за допомогою include_router, що дозволяє зберігати код окремих частин API у вигляді модулів (user_routes, message_routes). Маршрут root забезпечує перевірку працездатності API. При зверненні до кореневої URL-адреси повертає повідомлення, що сервер працює.

Також нам потрібно розробити базовий маршрут для обробки повідомлень. На рисунку 3.5 представлено код маршруту process_message.

```python
from fastapi import APIRouter, HTTPException, Depends
from sqlalchemy.orm import Session
from database import get_db
from models import Message, NLPResponse
from services.nlp_service import analyze_message
from services.message_service import save_message

router = APIRouter()

@router.post("/")
async def process_message(message: Message, db: Session = Depends(get_db)) -> NLPResponse:
    try:
        # Аналіз повідомлення за допомогою NLP
        analysis_result = analyze_message(message.content)

        # Збереження повідомлення в базі даних
        saved_message = save_message(db, message)

        return {
            "message_id": saved_message.id,
            "analysis_result": analysis_result
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Рисунок 3.5 – Код маршруту process_message

В даному коді маршрут process_message отримує повідомлення у вигляді об'єкта Message, виконує аналіз і повертає результат. Функція з NLP-модуля analyze_message (рисунок 3.6) визначає наміри, сутності та інші характеристики повідомлення. Також присутня функція save_message (рисунок 3.7) для збереження повідомлення в базі даних із використанням ORM. У разі помилки повертається HTTP-відповідь із кодом 500 і деталями помилки.

```python
from transformers import pipeline

# Ініціалізація моделі
ner_pipeline = pipeline("ner", model="dbmdz/bert-large-cased-finetuned-conll03-english")

def analyze_message(text: str):
    """Аналіз тексту для визначення сутностей."""
    entities = ner_pipeline(text)
    intents = detect_intents(text)
    return {
        "entities": entities,
        "intents": intents
    }

def detect_intents(text: str):
    """Простий аналіз намірів (для демонстрації)."""
    if "перевірити баланс" in text.lower():
        return {"intent": "check_balance", "confidence": 0.95}
    return {"intent": "unknown", "confidence": 0.5}

```

Рисунок 3.6 – Функції analyze_message та detect_intents

```python
def save_message(db, message_data):
    new_message = Message(content=message_data.content, user_id=message_data.user_id)
    db.add(new_message)
    db.commit()
    db.refresh(new_message)
    return new_message

```

Рисунок 3.7 – Функція save_message

Повернемось до вже раніше згаданої ORM. В даній інформаційній системі використовується SQLAlchemy. За допомогою неї ми спроєктували моделі бази даних (рисунок 3.8) для того, щоб наша ІС розуміла яким чином варто спілкуватися з базою даних. Також ми прописали ініціалізацію підключення ІС до бази даних (рисунок 3.9).

```python
from sqlalchemy import Column, Integer, String, ForeignKey, DateTime, Text
from sqlalchemy.orm import relationship
from database import Base
from datetime import datetime

class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    username = Column(String, index=True)
    email = Column(String, unique=True, index=True)
    created_at = Column(DateTime, default=datetime.utcnow)
    messages = relationship("Message", back_populates="user")

class Message(Base):
    __tablename__ = "messages"
    id = Column(Integer, primary_key=True, index=True)
    content = Column(Text)
    user_id = Column(Integer, ForeignKey("users.id"))
    sent_at = Column(DateTime, default=datetime.utcnow)
    user = relationship("User", back_populates="messages")
```

Рисунок 3.8 – Деякі моделі бази даних, розроблені за допомогою ORM SQLAlchemy

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

DATABASE_URL = "postgresql://user:password@localhost/chatbot"

Base = declarative_base()
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Рисунок 3.9 – Ініціалізація підключення ІС до бази даних

Розробивши типовий чат-бот, ми можемо перейти до його тестування.

### 3.4. Тестування інформаційної системи

Тестування є важливим етапом розробки, оскільки воно дозволяє перевірити коректність роботи всіх компонентів системи, виявити та усунути можливі помилки. Для цього було розроблено набір автоматизованих тестів, які перевіряють роботу API-сервера, модуля обробки природної мови (NLP), бази даних та інтеграції із зовнішніми сервісами. Усі тести написані з використанням бібліотеки pytest, що забезпечує високу гнучкість та зручність у тестуванні.

Для написання та виконання тестів використовувалися такі інструменти:

- **pytest:** основна бібліотека для створення тестів.
- **FastAPI TestClient:** для перевірки API-запитів.
- **sqlite:** для створення тестової бази даних у пам’яті.
- **pytest-mock:** для імітації (mock) зовнішніх сервісів.

Почнемо з конфігурації тестового середовища. На рисунку 3.10 представлено у вигляді коду те, які дії будуть виконуватись перед запуском тестів.

```python
import pytest
from fastapi.testclient import TestClient
from main import app
from database import Base, engine, get_db
from sqlalchemy.orm import sessionmaker

# Тестова база даних
SQLALCHEMY_DATABASE_URL = "sqlite:///./test.db"
TestSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

@pytest.fixture(scope="module")
def test_db():
    Base.metadata.create_all(bind=engine)  # Створення таблиць
    db = TestSessionLocal()
    try:
        yield db
    finally:
        db.close()
    Base.metadata.drop_all(bind=engine)  # Видалення таблиць після завершення тестування

@pytest.fixture(scope="module")
def client():
    with TestClient(app) as c:
        yield c
```

Рисунок 3.10 – Конфігурація тестового середовища

Ця конфігурація дозволяє створювати тестові таблиці перед початком тестування та видаляти їх після завершення, забезпечуючи ізольованість тестового середовища.

На рисунку 3.11 представлено тестування маршруту messages.

```python
def test_process_message(client, test_db):
    # Вхідні дані
    test_message = {"content": "Привіт, перевірити баланс", "user_id": 1}

    # Відправка POST-запиту
    response = client.post("/messages/", json=test_message)

    # Перевірка статусу відповіді
    assert response.status_code == 200

    # Перевірка вмісту відповіді
    data = response.json()
    assert "message_id" in data
    assert "analysis_result" in data
    assert data["analysis_result"]["intents"]["intent"] == "check_balance"
    assert data["analysis_result"]["intents"]["confidence"] > 0.9
```

Рисунок 3.11 – Тестування маршруту messages

У цьому тесті перевіряється:

1. Коректність обробки API-запиту.
2. Наявність у відповіді ID повідомлення.
3. Точність розпізнавання наміру check_balance з високим рівнем впевненості.

На рисунку 3.12 представлено тестування маршруту /users.

```python
def test_create_user(client):
    # Вхідні дані
    test_user = {"username": "testuser", "email": "testuser@example.com"}

    # Відправка POST-запиту
    response = client.post("/users/", json=test_user)

    # Перевірка статусу відповіді
    assert response.status_code == 201

    # Перевірка вмісту відповіді
    data = response.json()
    assert data["username"] == "testuser"
    assert data["email"] == "testuser@example.com"

```

Рисунок 3.12 – Тестування маршруту users

Даний тест перевіряє:

1. Чи створюється новий користувач у базі даних.
2. Чи збережені введені дані (ім'я користувача, email) відповідають очікуваним.

Перейдемо до тестування NLP. На рисунку 3.13 представлено перевірку того, чи вірним чином NLP-модуль розпізнає наміри та сутності з повідомлення.

```python
from services.nlp_service import analyze_message

def test_analyze_message():
    # Тестовий текст
    test_text = "Привіт, хочу перевірити баланс"

    # Виклик функції аналізу
    result = analyze_message(test_text)

    # Перевірка результатів
    assert "entities" in result
    assert "intents" in result
    assert result["intents"]["intent"] == "check_balance"
    assert result["intents"]["confidence"] > 0.9
```

Рисунок 3.13 – Тестування NLP-модуля

Не менш важливо провести тестування бази даних. На рисунку 3.14 показано код тесту для перевірки функції збереження повідомлень.

```python
from services.message_service import save_message
from models import Message

def test_save_message(test_db):
    # Тестові дані
    message_data = Message(content="Тестове повідомлення", user_id=1)

    # Збереження в базу даних
    saved_message = save_message(test_db, message_data)

    # Перевірка результатів
    assert saved_message.id is not None
    assert saved_message.content == "Тестове повідомлення"
    assert saved_message.user_id == 1
```

Рисунок 3.14 – Тестування функції save_message

Після розробки цих тестів, їх було запущено. Результат представлено на рисунку 3.15.

```
============================================================
                    test session starts
============================================================
platform linux -- Python 3.12, pytest-8.2.1, py-2.1.0, pluggy-0.15.2
rootdir: C:/Users/dmytro.s/PycharmProjects/chatbot-ml/app
collected 4 items

test_api.test_process_message ...... [100%]
test_api.test_create_user ...... [100%]
test_nlp.test_analyze_message ...... [100%]
test_database.test_save_message ...... [100%]

============================================================
                     4 passed in 6.31s
============================================================

Process finished with exit code 0
```

Рисунок 3.15 – Результат виконання тестів

Як ми можемо побачити, наші тести було успішно виконано, а отже це підтверджує працездатність інформаційної системи.

# ВИСНОВКИ

У процесі виконання дипломної роботи на тему «Інформаційна технологія проєктування чат-ботів з використанням штучного інтелекту» було проведено комплексний аналіз як теоретичних основ, так і практичних аспектів розробки адаптивних чат-ботів. Зокрема, дослідження охоплювало сучасні методи обробки природної мови (NLP), алгоритми машинного навчання, а також платформи й інструменти, що використовуються для створення таких систем. Це дозволило глибоко зрозуміти принципи функціонування чат-ботів, їхні можливості та обмеження.

Результати аналізу показали, що розробка сучасних чат-ботів вимагає інтеграції ефективних технологій і підходів. Зокрема, важливим є застосування алгоритмів розпізнавання намірів і виділення сутностей для точного розуміння запитів користувачів, використання передових трансформерних моделей (таких як BERT і GPT) для контекстної генерації відповідей та аналізу складних діалогів. Також необхідно створювати масштабовану інфраструктуру на основі мікросервісної архітектури, а інтеграція з популярними каналами комунікації, такими як Telegram або Facebook Messenger, значно підвищує зручність для користувачів.

Проведений аналіз таких інструментів, як FastAPI, Hugging Face Transformers, PostgreSQL, а також платформ IBM Watson Assistant, Google Dialogflow, Microsoft Bot Framework і Rasa, дав змогу виокремити їхні ключові переваги та недоліки. Це підтвердило, що обрані технології, зокрема Python у поєднанні з FastAPI та NLP-моделями, є оптимальними для реалізації цього проєкту.

У ході роботи було спроєктовано архітектуру інформаційної системи, яка передбачає чітке розмежування клієнтського і серверного рівнів із можливістю інтеграції із зовнішніми платформами. На основі цієї архітектури було розроблено прототип системи, що забезпечує адаптивні

можливості для створення чат-ботів. Автоматизоване тестування підтвердило стабільність і надійність роботи системи.

Запропонована інформаційна система продемонструвала високу продуктивність, гнучкість у налаштуванні та здатність до масштабування. Вона відповідає сучасним вимогам до обробки природної мови, інтеграції з різними системами та забезпечення захисту даних. Отримані результати й практична реалізація заклали основу для подальших досліджень у сфері створення інтелектуальних чат-ботів, які можуть автоматизувати бізнес-процеси, підвищити якість обслуговування користувачів і забезпечити ефективну взаємодію в різних галузях.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛИ


1. Jurafsky D., Martin J. H. Speech and Language Processing [Електронний ресурс] / D. Jurafsky, J. H. Martin. – Режим доступу: https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf (дата звернення: 18.10.2024).

2. Russell S., Norvig P. Artificial Intelligence: A Modern Approach (4th Edition) / S. Russell, P. Norvig. – Pearson, 2020. – 1152 с. (дата звернення: 26.10.2024).

3. Goodfellow I., Bengio Y., Courville A. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville. – MIT Press, 2016. – 800 с. [Електронний ресурс]. – Режим доступу: http://www.deeplearningbook.org/ (дата звернення: 10.11.2024).

4. Chollet F. Deep Learning with Python / F. Chollet. – Manning Publications, 2018. – 384 с. (дата звернення: 23.10.2024).

5. Vaswani A. та ін. Attention is All You Need [Електронний ресурс] / A. Vaswani та ін. – Режим доступу: https://arxiv.org/abs/1706.03762 (дата звернення: 11.11.2024).

6. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Електронний ресурс] / J. Devlin, M.-W. Chang, K. Lee, K. Toutanova. – Режим доступу: https://arxiv.org/abs/1810.04805 (дата звернення: 15.11.2024).

7. Brown T. B. та ін. Language Models are Few-Shot Learners [Електронний ресурс] / T. B. Brown та ін. – Режим доступу: https://arxiv.org/abs/2005.14165 (дата звернення: 30.11.2024).

8. Zhang C., Wallace B. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification [Електронний ресурс] / C. Zhang, B. Wallace. – Режим доступу: https://arxiv.org/abs/1510.03820 (дата звернення: 23.11.2024).

9. Hochreiter S., Schmidhuber J. Long Short-Term Memory / S. Hochreiter, J. Schmidhuber // Neural Computation. – 1997. – Т. 9(8). – С. 1735–1780 (дата звернення: 19.11.2024).

10. Sutskever I., Vinyals O., Le Q. V. Sequence to Sequence Learning with Neural Networks [Електронний ресурс] / I. Sutskever, O. Vinyals, Q. V. Le. – Режим доступу: https://arxiv.org/abs/1409.3215 (дата звернення: 12.11.2024).

11. Rasa Documentation [Електронний ресурс]. – Режим доступу: https://rasa.com/docs/ (дата звернення: 19.11.2024).

12. Google Dialogflow Documentation [Електронний ресурс]. – Режим доступу: https://cloud.google.com/dialogflow/docs (дата звернення: 6.11.2024).

13. Microsoft Bot Framework Documentation [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/uk-ua/azure/bot-service/ (дата звернення: 6.11.2024).

14. IBM Watson Assistant Documentation [Електронний ресурс]. – Режим доступу: https://cloud.ibm.com/docs/assistant (дата звернення: 6.11.2024).

15. Amazon Lex Documentation [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/lex/ (дата звернення: 6.11.2024).

16. Manning C. D., Schütze H. Foundations of Statistical Natural Language Processing / C. D. Manning, H. Schütze. – MIT Press, 1999. – 720 с. (дата звернення: 21.11.2024).

17. OpenAI GPT API Documentation [Електронний ресурс]. – Режим доступу: https://platform.openai.com/docs/introduction (дата звернення: 21.11.2024).

18. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space [Електронний ресурс] / T. Mikolov, K. Chen, G. Corrado, J. Dean. – Режим доступу: https://arxiv.org/abs/1301.3781 (дата звернення: 25.11.2024).

19. Костюченко Н. М., Залевська Н. В. Обробка природної мови в системах штучного інтелекту / Н. М. Костюченко, Н. В. Залевська. – Київ: Наукова думка, 2018. – 256 с. (дата звернення: 27.11.2024).

20. Назаренко О. В. Штучний інтелект і машинне навчання: концепції та

застосування / О. В. Назаренко. – Харків: ХНУРЕ, 2020. – 312 с. (дата звернення: 28.10.2024).