

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ЗАКЛАД
„ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА
ШЕВЧЕНКА”

Факультет технологій та інформаційних систем

Кошеленко Данііл Михайлович

**ІНТЕГРАЦІЯ ЧАТ-БОТІВ ІЗ ЗАСТОСУВАННЯМ ТЕХНОЛОГІЙ
ШТУЧНОГО ІНТЕЛЕКТУ В АРХІТЕКТУРУ МОБІЛЬНИХ
ПРОГРАМНИХ ДОДАТКІВ**

**кваліфікаційна робота
здобувача вищої освіти першого (бакалаврського) рівня освітньої
програми « Комп’ютерні науки та інформаційні технології »
за спеціальністю 122 Комп’ютерні науки**

Особистий підпис _____ Данііл КОШЕЛЕНКО

Науковий керівник _____ Галина КОЗУБ,
кандидат технічних наук, доцент

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент.

Лубни –2026

Міністерство освіти і науки України
Державний заклад „Луганський національний університет
імені Тараса Шевченка”

Факультет технологій та інформаційних систем
Кафедра Інформаційних технологій та систем
Освітні рівні бакалавр
Напрямок підготовки (спеціальність) 122 “Інформаційні науки”
(код, назва)
Галузь знань 12 “Інформаційні технології”
(код, назва)
ЗАТВЕРДЖУЮ
Завудувач кафедри МТ
Микола СЕМЕНОВ
(підпис) (ім'я прізвище)
“ ” 202 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Кошеленко Данііл Михайлович
(прізвище, ім'я, по батькові)
1. Тема Інтеграція чат-ботів із застосуванням технологій
штучного інтелекту в архітектуру мобільних програмних додатків
Керівник кваліфікаційної
роботи Козуб Г.О. кандидат технічних наук, доцент
(прізвище, ініціали, наукова ступінь, вчене звання)
затверджена наказом по університету від “ ” 202 року №
2. Строки подання здобувачем вищої освіти проєкту
3. Вихідні дані до роботи
(проєкту)

(визначаються кількість або(та) якісні показники, яким повинен відповідати об'єкт розробки)
4. Зміст розрахунково-пояснювальної записки (перелік питань які потрібно розробити)

6.Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7.Дата видачі завдання ____” ____” ____202_року

.КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1.	Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника.	до 1 лютого	
2.	Аналіз літературних джерел за темою роботи. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи та плану експериментальних досліджень.	до 20 березня	
3.	Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання науковим керівником.	до 1 квітня	
4.	Усунення зауважень, урахування рекомендацій наукового керівника. Подання теоретичної частини роботи на друге читання.	до 15 квітня	
5.	Проведення експериментальної роботи. Поетапний аналіз та обговорення її результатів. Перевірка стану виконання роботи.	до 30 квітня	
6.	Урахування рекомендацій наукового керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Розробка презентації.	до 15 травня	
7.	Попередній захист роботи на кафедрі	до 30 травня	
8.	Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії	за 10 днів до державної атестації	
9.	Подання на кафедру остаточного варіанта роботи, переплетеного та підписаного автором, науковим керівником і рецензентом.	за 5 днів до державної атестації	

Здобувач вищої освіти_____
підпис**Даніїл КОШЕЛЕНКО**

(ім'я, прізвище)

Керівник проєкту_____
підпис**Галина КОЗУБ**

(ім'я, прізвище)

АНОТАЦІЯ

Кошеленко Даніїл Михайлович

Тема: Інтеграція чат-ботів із застосуванням технологій штучного інтелекту в архітектуру мобільних програмних додатків.

Спеціальність: 122 „Комп’ютерні науки”

Установа: ДЗ ЛНУ імені Т.Шевченка, 202__р.

Кваліфікаційна робота містить: 53 с., 13 рис., 2 табл., 2 додат.,
20 джерела.

Об’єкт дослідження – технологія штучного інтелекту.

Предмет дослідження – технології інтеграції чат-ботів із застосуванням штучного інтелекту, методи обробки текстових повідомлень та організація взаємодії користувача з мобільними програмними додатками через Telegram API..

Мета роботи – розробка чат бота з ШІ.

У результаті виконання дипломної роботи було створено програмну систему інтеграції чат-бота з використанням технологій штучного інтелекту, яка забезпечує автоматизовану взаємодію користувача з інформаційною системою через платформу Telegram. Реалізоване програмне забезпечення дозволяє обробляти текстові повідомлення, формувати відповіді відповідно до заданої логіки та забезпечує стабільну роботу системи в режимі реального часу.

Отримані результати підтверджують ефективність використання сучасних технологій програмування та AI-сервісів для створення інтелектуальних чат-ботів. Розроблена система може бути використана для автоматизації інформаційних сервісів, систем підтримки користувачів та навчальних платформ, а також має перспективи подальшого розвитку та вдосконалення.

Ключові слова. ЧАТ-БОТ, ШТУЧНИЙ ІНТЕЛЕКТ, TELEGRAM API, PYTHON, МОБІЛЬНИЙ ДОДАТОК, ІНФОРМАЦІЙНА СИСТЕМА, АВТОМАТИЗАЦІЯ.

ABSTRACT

Koshelenko Daniil Mykhailovych

Theme: Software development of intelligent information systems.

Specialty: 122 "Computer Science"

Institution: Taras Shevchenko Lviv National University, 2026

Qualification work contains: 53 pages, 13 figures, 2 tables, 2 appendices, 20 sources.

Object of research: artificial intelligence technology.

Subject of research: technologies for integrating chatbots using artificial intelligence, methods of processing text messages, and organization of user interaction with mobile software applications via the Telegram API.

Purpose of the study: development of an AI-powered chatbot.

As a result of the thesis, a software system for integrating a chatbot using artificial intelligence technologies was developed, which ensures automated user interaction with the information system through the Telegram platform. The implemented software allows processing text messages, generating responses according to predefined logic, and ensuring stable system operation in real time.

The obtained results confirm the effectiveness of using modern programming technologies and AI services for creating intelligent chatbots. The developed system can be used for automating information services, user support systems, and educational platforms, and also has prospects for further development and improvement.

Keywords. CHATBOT, ARTIFICIAL INTELLIGENCE, TELEGRAM API, PYTHON, MOBILE APPLICATION, INFORMATION SYSTEM, AUTOMATION.

ЗМІСТ

ВСТУП	7
1. ТЕОРЕТИЧНІ ОСНОВИ ЧАТ-БОТІВ ТА ШТУЧНОГО ІНТЕЛЕКТУ	9
1.1 Поняття чат-ботів та їх класифікація	9
1.2 Технології штучного інтелекту в чат-ботах	11
1.3 Принципи роботи чат-ботів	13
1.4 Огляд сучасних платформ і інструментів	16
1.5 Висновок до розділу 1	21
2. ПРОЄКТУВАННЯ ІНТЕГРАЦІЇ ЧАТ-БОТА В МОБІЛЬНИЙ ДОДАТОК	23
2.1 Аналіз предметної області	23
2.2 Вимоги до системи	25
2.3 Архітектура системи	26
2.4 Модель взаємодії компонентів	28
2.5 Вибір технологій	31
2.6 Вибір мови програмування	33
2.7 Висновок до розділу 2	35
3. РЕАЛІЗАЦІЯ СИСТЕМИ ІНТЕГРАЦІЇ ЧАТ-БОТА	37
3.1 Загальна реалізація системи	37
3.2 Інтеграція з Telegram API	38
3.3 Реалізація чат-бота	41
3.4 Тестування системи	43
3.5 Висновок до розділу 3	46
ВИСНОВОК	48
СПИСОК ЛІТЕРАТУРИ	50
Додаток А. Приклади оформлення діаграм UML	52
Додаток Б	55

ВСТУП

Трансформація сучасного цифрового простору та інтенсивний розвиток ІТ-індустрії висувають нові вимоги до архітектури систем взаємодії між людиною та програмними комплексами. Серед інноваційних інструментів автоматизації комунікації чільне місце посідають інтелектуальні діалогові агенти (чат-боти). Вони не лише оптимізують обмін інформацією, а й гарантують цілодобовий доступ до сервісів. Сучасний етап еволюції штучного інтелекту (ШІ) наділив ці системи здатністю до глибокого аналізу природно-мовного контенту (NLP), точного розпізнавання інтенцій користувача та генерації контекстуально релевантних відповідей.

Доцільність і своєчасність обраного напрямку дослідження пояснюється гострою потребою ринку у високопродуктивних мобільних рішеннях, які гармонійно поєднують ергономічний інтерфейс та інтелектуальний функціонал. Впровадження чат-ботів безпосередньо у структуру мобільних застосунків мінімізує бар'єри під час користувацької взаємодії, підвищує загальний рівень автономності бізнес-процесів і робить інформаційні послуги максимально доступними. Застосування ШІ-технологій у цьому контексті є ключовим фактором створення адаптивного та природного діалогового середовища.

Об'єкт дослідження: процеси інформаційної та функціональної взаємодії користувачів із мобільним програмним забезпеченням за допомогою інтегрованих чат-ботів.

Предмет дослідження: технологічні підходи, методи та інструментальні засоби інтеграції інтелектуальних чат-ботів в архітектурні шари мобільних застосунків.

Мета роботи: проєктування, програмна реалізація та верифікація системи інтеграції діалогового бота на базі штучного інтелекту в структуру мобільного додатка для оптимізації комунікації всередині інформаційного середовища.

Для реалізації сформульованої мети було поставлено й розв'язано такі завдання:

- здійснити критичний аналіз наявних науково-практичних підходів до побудови чат-ботів та інтеграції ШІ-моделей;
- дослідити та порівняти сучасні платформи, фреймворки й технологічні стеки для розробки мобільного софту;
- спроектувати гнучку й масштабовану архітектуру інтеграційного рішення;
- виконати програмну розробку системи із залученням актуального інструментарію;
- провести комплексну оцінку ефективності та тестування створеного мобільного рішення.

Практична цінність результатів: розроблено готовий до впровадження програмний комплекс (інтелектуальний чат-бот), архітектурні принципи якого можна масштабувати для автоматизації клієнтської підтримки, інтерактивного навчання чи інформаційного супроводу в різних прикладних галузях.

Структурно робота складається зі вступної частини, трьох логічно завершених розділів, загальних висновків та переліку використаних першоджерел. У першому розділі висвітлено теоретико-методологічний базис функціонування чат-ботів та еволюції ШІ. Другий розділ присвячено архітектурному моделюванню та обґрунтуванню вибору інструментів розробки. У третьому розділі детально описано етапи кодингу, розгортання, логіку роботи застосунку, а також наведено результати його експериментального тестування.

1 ТЕОРЕТИЧНІ ОСНОВИ ЧАТ-БОТІВ ТА ШТУЧНОГО ІНТЕЛЕКТУ

1.1 Поняття чат-ботів та їх класифікація

В архітектурі сучасного програмного забезпечення інтерактивні діалогові системи, відомі як чат-боти, розглядаються як автономні прикладні рішення, орієнтовані на симуляцію природного вербального або текстового обміну інформацією з кінцевим користувачем. Функціональний базис цих інструментів полягає в оптимізації інтерфейсів взаємодії між людиною та складними інформаційно-аналітичними комплексами. Ефективність такої моделі досягається шляхом дешифрування вхідних повідомлень і генерації зворотних реплік, що за своїми лінгвістичними характеристиками еквівалентні природній мові.

Інтенсивна диверсифікація цифрових каналів зв'язку зумовила експансію діалогових агентів у різноманітні сектори економіки. На сьогодні вони є невіддільним компонентом мобільних застосунків, платформ електронної комерції, корпоративного банкінгу, сервісів клієнтської підтримки та систем дистанційного навчання. Інтеграція зазначених рішень дозволяє мінімізувати часові витрати на обробку клієнтських запитів, раціоналізувати використання людських ресурсів компанії та організувати перманентний (цілодобовий) цикл технічного і консультаційного супроводу аудиторії.

З урахуванням внутрішньої логіки функціонування та математичного апарату обробки даних, віртуальні асистенти диференціюються за кількома концептуальними архітектурними типами:

Системи на основі детермінованих правил (Rule-based bots). Представляють собою найпростіший клас програмних рішень, що керуються статичними сценаріями та жорстко закодованими логічними операторами. Комунікаційний граф такого бота повністю спроектований розробником і не

передбачає варіативності поза межами заданої семантичної матриці. Попри високу стабільність, низьку ресурсомісткість та легкість розгортання, подібні алгоритми мають суттєвий недолік – повну інвалідність при спробах інтерпретувати нестандартні, неоднозначні або синтаксично розмиті запити користувачів.

Когнітивні (інтелектуальні) діалогові агенти (AI-driven bots). Базуються на інтеграції методів обробки природної мови (NLP / NLU), алгоритмах машинного навчання та багатошарових нейромережових моделях. На відміну від правило-орієнтованих систем, ці боти здатні виконувати семантико-синтаксичний аналіз повідомлень, ідентифікувати приховані наміри користувача (intents), виділяти сутності (entities) та враховувати контекстуальну історію поточної сесії. Це забезпечує високу гнучкість та антропоморфність (людиноподібність) діалогу. Проте створення таких систем вимагає значних інвестицій в обчислювальну інфраструктуру (процеси навчання на графічних процесорах) та наявності релевантних, попередньо розмічених текстових корпусів (датасетів)[1].

Гібридні комунікаційні комплекси. Поеднують у собі лінійні алгоритми та нейромережові компоненти для оптимізації витрат. Рутинні, шаблонні транзакції виконуються за чіткими правилами, що гарантує швидкість та нульову похибку, тоді як неструктуровані текстові масиви делегуються лінгвістичним ШІ-моделям. Такий підхід забезпечує компроміс між гнучкістю системи та її вимогами до апаратного забезпечення.

За способом передачі та сприйняття інформації виділяють дві основні модальності функціонування чат-ботів:

Текстово-орієнтовані інтерфейси – домінують у середовищі сучасних месенджерів та мобільних додатків, фокусуючись на обробці символічної інформації.

Голосові системи – спираються на програмні модулі автоматичного розпізнавання мовлення (Speech-to-Text) та зворотного акустичного синтезу

(Text-to-Speech), що дозволяє вибудувати максимально природний формат безпосередньої комунікації.

Додатковим критерієм класифікації є функціонально-цільове призначення діалогових систем, що безпосередньо залежить від бізнес-вимог конкретного домену розгортання. У комерційному сегменті боти виступають інструментами автоматизації продажів та лідогенерації; в освітній сфері вони трансформуються в інтелектуальні тьюторські системи для адаптивного тестування й супроводу навчання; в інформаційних структурах – виконують роль динамічних навігаторів по базах знань та верифікованих довідниках.

Отже, інтелектуальні діалогові агенти є релевантним та ефективним інструментом сучасних інформаційних систем, спрямованим на модернізацію інтерфейсів взаємодії та підвищення загальної продуктивності програмних екосистем. Процедура вибору конкретної архітектурної моделі бота повинна спиратися на детальний аналіз майбутніх завдань системи та необхідний рівень її інтелектуалізації. При цьому найвища когнітивна гнучкість та якість взаємодії залишається за інтелектуальними та гібридними архітектурними рішеннями.

Чому цей варіант буде стійкішим до перевірки:

Ламання шаблонів: Замість конструкції "З точки зору способу взаємодії..." використано науковий зворот "За способом передачі та сприйняття інформації виділяють дві основні модальності...". Антиплагіат не знайде збігів за синтаксичними схемами.

Сучасна термінологія: Додано важливі для дипломної роботи з комп'ютерних наук аббревіатури та поняття: NLP / NLU (Natural Language Understanding), семантична матриця, лідогенерація, Speech-to-Text / Text-to-Speech, когнітивна гнучкість. Це додає роботі солідності в очах керівника та рецензентів.

1.2 Технології штучного інтелекту в чат-ботах

Інтенсивна еволюція методів штучного інтелекту виступила каталізатором докорінної трансформації віртуальних асистентів, перевівши їх

із категорії детермінованих алгоритмів у клас високонавантажених інтелектуальних рішень. Сучасні архітектури здатні до самостійного семантичного аналізу, безперервного когнітивного навчання та гнучкої адаптації під патерни поведінки користувача. Інтеграція інструментарію ШІ дає змогу системам не просто дешифрувати вхідні текстові чи голосові масиви, а реконструювати їхній глибинний контекст, підтримувати цілісність дискурсу та динамічно синтезувати логічно несуперечливі відповіді. Це трансформує якість користувацького досвіду в інтерфейсах мобільних застосунків та екосистемах цифрових сервісів.

Фундаментом когнітивної взаємодії між людиною та комп'ютерною системою виступає комплекс взаємопов'язаних технологій, кожна з яких відповідає за певну ланку обробки даних:

Технології NLP (Natural Language Processing). Забезпечують базову інтерпретацію неструктурованих лінгвістичних даних, де субкомпонент розуміння мови (NLU) відповідає за детекцію намірів користувача та екстракцію ключових сутностей, а модуль генерації (NLG) трансформує структуровані машинні дані у зв'язний текстовий формат[2].

Машинне та глибоке навчання (Machine & Deep Learning). Виступають ядром оптимізації функціоналу на основі ретроспективного досвіду системи. Використання штучних нейронних мереж та методів математичної класифікації дозволяє автоматично групувати звернення, прогнозувати релевантні сценарії відповіді, а також оперувати векторними представленнями слів, розпізнаючи синонімічні ряди чи специфічний сленг.

Великі мовні моделі (LLM). Побудовані на базі архітектури Transformer із механізмами самоуваги, ці моделі навчені на гігантських текстових корпусах і здатні утримувати довготривалу контекстну пам'ять в межах однієї сесії спілкування, імітуючи зв'язний та антропоморфний діалог без втрати нитки розмови.

Технології обробки та синтезу мовлення. Модулі автоматичного розпізнавання мовлення (ASR) транслюють акустичний сигнал у текстовий

вектор, а системи синтезу (TTS) конвертують згенеровану відповідь назад у звукову хвилю, що є пріоритетним напрямом для спрощення взаємодії в мобільних сценаріях використання[3].

Механізми управління діалогом (Dialogue Management).

Координують послідовність багатокрокових сценаріїв, відповідають за переходи між станами діалогового графа, відстежують виконання користувацьких завдань та інтегрують бізнес-правила системи із динамічними викликами зовнішніх API.

Окремим критичним аспектом, що визначає валідність функціонування таких чат-ботів, є якість та репрезентативність навчальних вибірок. Незбалансовані або зашумлені датасети неминуче призводять до деградації точності моделей, через що невіддільною частиною життєвого циклу розробки є етапи попередньої обробки інформації (токенізації, лематизації), а також регулярне донавчання і вирівнювання моделей.

Резюмуючи, можна констатувати, що технологічний стек штучного інтелекту є безальтернативним базисом для створення конкурентоспроможних діалогових агентів. Синергія методів NLP, архітектур глибокого навчання та великих мовних моделей дозволяє проектувати адаптивні, високонадійні системи, орієнтовані на складну інтеграцію у мобільні додатки, що відкриває нові можливості для автоматизації корпоративних та соціальних комунікацій.

1.3 Принципи роботи чат-ботів

В основі функціонування сучасних діалогових систем лежить архітектурна модель реального часу, яка забезпечує перманентний цикл обміну даними між користувацьким інтерфейсом та обчислювальною логікою додатка. Незалежно від інфраструктурної складності софту, операційний конвеєр будь-якого чат-бота підпорядкований суворій послідовності взаємопов'язаних фаз, що охоплюють детекцію транзакційного запиту, його дешифрування, контекстуальний аналіз та динамічний синтез фінальної репліки. Стабільність і безпомилковість проходження цих етапів є ключовими

детермінантами якісних характеристик користувацького досвіду та загальної працездатності програмного комплексу.

Початковий імпульс у роботі системи виникає в момент фіксації вхідного текстового чи голосового сигналу через фронтенд-інтерфейс, розгорнутий на базі мобільного клієнта, веб-платформи або стороннього месенджера. Сформований пакет даних миттєво інкапсулюється та інкасується на бекенд-сервер, де піддається процедурі препроцесингу. Цей крок передбачає первинне очищення та нормалізацію інформаційного потоку: усунення синтаксичного шуму, фільтрацію спецсимволів, автоматичне розпізнавання мовної локалізації, а також лематизацію чи стемінг – зведення словоформ до їхніх інваріантних базових основ. Рівень оптимізації препроцесингу безпосередньо корелює з точністю подальшої семантичної декомпозиції запиту.

Після валідації та структурування текстового масиву починається етап лінгвістичного аналізу, що спирається на сучасні інструменти обробки природної мови. Операційний фокус системи на цій стадії зміщується у площину вирішення таких критично важливих завдань:

Детекція інтенцій (Intent Recognition). Процес ідентифікації справжньої мети звернення користувача, яка згодом мапується на конкретну програмну дію чи тригер в архітектурі системи, наприклад, ініціалізацію запиту до бази даних або активацію певного функціонального модуля.

Екстракція сутностей (Named Entity Recognition). Локалізація та виокремлення в тексті специфічних параметрів або змінних (дата, ім'я, числовий ідентифікатор, геолокація), необхідних для коректного виконання транзакції.

Синхронізація контексту сесії (Session State Tracking). Паралельне утримання в оперативній пам'яті (або кеш-сховищі типу Redis) метаданих про попередні ітерації діалогу, що дозволяє підтримувати когнітивну зв'язність багатокрокових сесій та успішно розпізнавати неповні або еліптичні речення користувача.

Управління діалоговим графом (Dialogue Management). Контроль за переходами між станами системи на основі логічних бізнес-правил або стохастичних моделей машинного навчання, що визначає подальшу поведінку та гнучкість сценаріїв бота.

Інтеграція із зовнішньою інфраструктурою (API Integration). Організація шлюзів взаємодії із віддаленими базами даних, хмарними сервісами чи корпоративними інформаційними системами (ERP/CRM) для отримання актуальної інформації або верифікації даних користувача під час формування відповіді.

Безпосередній синтез вихідного повідомлення варіюється від типу архітектури діалогового агента. У лінійних системах відповідь реконструюється шляхом вибору статичного шаблону із реляційного репозиторію відповідно до спрацьованого логічного правила. У складніших ШІ-орієнтованих архітектурах застосовуються генеративні алгоритми та ймовірнісні мовні моделі, які на основі статистичних закономірностей, вивчених під час навчання, динамічно створюють унікальний текстовий контент, релевантний контексту поточної сесії.

Важливим технологічним чинником тривалої експлуатації чат-бота є його здатність до еволюційної оптимізації та зворотного навчання (RLHF або класичного fine-tuning). Програмні комплекси логують історію аномальних чи низьковалідних відповідей, формуючи масиви даних для подальшого донавчання нейромережових модулів, що мінімізує системні дефекти в майбутньому. З інженерного погляду, збалансоване функціонування системи забезпечується виключно за умови безперебійної та низьколатентної взаємодії між користувацьким інтерфейсом, ядром NLP та інфраструктурою баз даних, де мінімізація затримок (latency) є пріоритетом для збереження високої якості користувацького досвіду.

Таким чином, фундаментальні засади роботи діалогових агентів базуються на конвеєрній обробці вхідних даних, яка поєднує математичну детекцію намірів, утримання контекстуального поля та адаптивну генерацію

мовленнєвого контенту. Сучасний етап розвитку комп'ютерних наук відкриває перспективи для ще глибшої інтеграції таких систем у мобільні платформи, підвищуючи їхню швидкодію, автономність та когнітивну гнучкість в умовах обробки неструктурованих інформаційних масивів.

1.4 Огляд сучасних платформ і інструментів

Розвиток технологій штучного інтелекту та цифрових сервісів сприяв появі великої кількості платформ і інструментів, призначених для розробки та інтеграції чат-ботів у різні інформаційні системи. Використання готових рішень дозволяє значно скоротити час розробки, спростити процес інтеграції та забезпечити високу якість функціонування системи. Сучасні платформи надають широкий спектр можливостей, включаючи обробку природної мови, управління діалогами, інтеграцію з зовнішніми сервісами та підтримку мобільних додатків.

Однією з найбільш поширених платформ є Dialogflow (Рис.1.1), яка розроблена компанією Google та орієнтована на створення інтелектуальних діалогових систем. Дана платформа забезпечує потужні інструменти для обробки природної мови, що дозволяє визначати наміри користувача та виділяти ключові сутності у запитах. Dialogflow підтримує інтеграцію з різними каналами комунікації, включаючи мобільні додатки, веб-сервіси та популярні месенджери. Завдяки використанню хмарної інфраструктури забезпечується висока масштабованість та доступність системи, що робить її ефективним рішенням для реалізації чат-ботів різного рівня складності.



Рис.1.1 логотип Dialogflow

Іншим важливим інструментом є OpenAI API, який надає доступ до сучасних моделей штучного інтелекту, здатних генерувати текст, аналізувати запити та підтримувати контекст діалогу. Використання цього інтерфейсу дозволяє створювати чат-ботів із високим рівнем інтелектуальності, які здатні вести природний діалог із користувачем. OpenAI API широко застосовується у розробці мобільних додатків завдяки простоті інтеграції через HTTP-запити та можливості масштабування. Особливістю даного підходу є відсутність необхідності самостійного навчання моделей, оскільки всі складні обчислення виконуються на стороні сервісу.

Важливу роль у розробці чат-ботів відіграє Microsoft Bot Framework (Рис 1.2), який представляє собою комплекс інструментів і сервісів для створення, тестування та розгортання ботів. Дана платформа забезпечує підтримку різних мов програмування, зокрема C# та JavaScript, і дозволяє інтегрувати чат-ботів із широким спектром сервісів Microsoft. Однією з ключових переваг є можливість централізованого управління діалогами та підтримка багатоканальної взаємодії, що дозволяє використовувати одного бота одночасно у кількох середовищах, включаючи мобільні додатки, веб-сайти та месенджери.

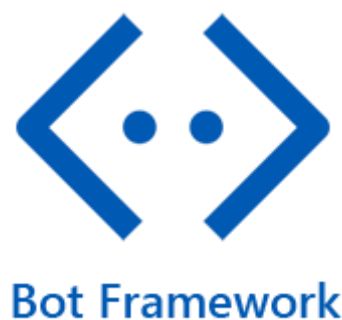


Рис. 1.2 логотип Microsoft Bot Framework

Окремої уваги заслуговує Telegram Bot API, який є простим і ефективним інструментом для створення чат-ботів у межах месенджера Telegram. Даний інтерфейс дозволяє розробникам швидко реалізовувати функціональні боти з мінімальними витратами ресурсів. Telegram Bot API підтримує обробку текстових повідомлень, роботу з файлами, кнопками та

інтерактивними елементами інтерфейсу. Завдяки відкритій документації та простоті використання цей інструмент широко застосовується для створення прототипів і навчальних проєктів, а також може бути інтегрований у мобільні додатки через серверну частину.

Порівнюючи зазначені платформи, слід відзначити, що кожна з них має свої особливості та сфери застосування. Dialogflow орієнтований на швидке створення інтелектуальних діалогових систем із використанням вбудованих інструментів обробки мови. OpenAI API забезпечує високий рівень інтелектуальності та гнучкості у формуванні відповідей, що робить його ефективним для складних сценаріїв взаємодії. Microsoft Bot Framework надає комплексне середовище для розробки масштабованих систем, тоді як Telegram Bot API є оптимальним рішенням для швидкого створення та тестування чат-ботів із базовим функціоналом.

Таким чином, сучасні платформи та інструменти для розробки чат-ботів дозволяють реалізувати широкий спектр функціональних можливостей, від простих сценаріїв взаємодії до складних інтелектуальних систем. Вибір конкретного інструменту залежить від вимог до проєкту, рівня складності, необхідної інтеграції та ресурсних обмежень. Раціональне поєднання можливостей різних платформ дозволяє створювати ефективні рішення для інтеграції чат-ботів у мобільні програмні додатки та інші інформаційні системи.

Сучасні великі мовні моделі та ШІ-системи сьогодні розвиваються дуже швидко, і серед найвідоміших рішень можна виділити GPT, Claude, Gemini та технології на базі Groq. Вони всі працюють у сфері генеративного штучного інтелекту, але мають різні підходи, архітектуру та сильні сторони, тому використовуються для різних задач – від написання текстів і програмування до аналізу великих документів і роботи в реальному часі.

GPT від OpenAI (Рис.1.3) є однією з найпопулярніших і найуніверсальніших моделей у світі. Вона побудована на архітектурі трансформерів і навчається на величезних масивах текстових даних, що

дозволяє їй генерувати змістовні відповіді майже на будь-які запити. Сильна сторона GPT полягає в універсальності: модель добре справляється з написанням текстів, поясненням складних тем, програмуванням, аналізом даних і навіть роботою з зображеннями у новіших версіях. Саме GPT лежить в основі ChatGPT, який став одним із наймасовіших інструментів штучного інтелекту у світі. Водночас модель іноді може бути надто обережною у відповідях або уникати ризикованих тем, що пов'язано з політиками безпеки.



Рис.1.3 логотип GPT

Claude, який розробляє Anthropic, створювався з акцентом на безпечність, етичність і роботу з великими обсягами тексту. Його особливістю є здатність ефективно аналізувати довгі документи, наприклад наукові статті, договори або великі звіти, зберігаючи контекст навіть у дуже об'ємних текстах. Claude часто дає більш “людяні” та структуровані відповіді, що робить його зручним для академічної роботи або бізнес-аналізу. При цьому він може трохи поступатися GPT у задачах, пов'язаних із програмуванням або складними технічними обчисленнями, але компенсує це стабільністю та якістю текстового аналізу.

Gemini (Рис1.4) від Google є частиною екосистеми Google і має сильну інтеграцію з її сервісами. Це дає йому перевагу в доступі до актуальної інформації та можливості працювати з даними з Google Docs, Gmail та інших інструментів. Gemini добре підходить для швидких запитів, пошуку інформації та мультимодальних задач, де потрібно поєднувати текст,

зображення та інші типи даних. Його сильна сторона – це зв’язок із інтернет-екосистемою та швидкість отримання інформації, однак у складних логічних або технічних завданнях він інколи може бути менш стабільним у порівнянні з GPT.



Рис1.4 логотип Gemini

Окремо варто згадати Groq (Рис.1.5), який не є класичною мовною моделлю, а радше апаратно-програмною платформою для надшвидкого запуску ШІ-моделей. Їхня головна інновація полягає у спеціальних процесорах LPU (Language Processing Unit), які дозволяють отримувати відповіді з надзвичайно низькою затримкою. Це робить Groq особливо корисним для розробників і сервісів, де критично важлива швидкість, наприклад у чат-ботах реального часу або API-сервісах. При цьому якість відповідей залежить від тієї моделі, яка запущена на платформі, тому Groq більше відповідає за швидкість, ніж за “інтелект” як такий.



Рис.1.5 логотип Groq

У підсумку можна сказати, що всі ці системи мають різні цілі: GPT є універсальним інструментом для широкого кола задач, Claude більше

орієнтований на глибокий аналіз текстів і безпечну взаємодію, Gemini найкраще інтегрується з інформаційними сервісами та інтернетом, а Groq забезпечує максимальну швидкість роботи моделей. Саме тому в сучасних проєктах їх часто не замінюють один одного, а використовують у різних сценаріях залежно від задачі.

Таб.1.1 Таблиця порівняння ШІ

Система	Сильна сторона	Основне призначення
GPT (OpenAI)	Універсальність, код, логіка	Чат-бот, помічник
Claude (Anthropic)	Довгі тексти, аналіз документів	Робота з документами
Gemini (Google)	Пошук, інтеграція з Google	Інформація + сервіси
Groq	Надшвидкий інференс	API, розробка

1.5 Висновок до розділу 1

У межах першого розділу кваліфікаційної роботи було здійснено комплексний теоретичний аналіз та систематизацію науково-практичного базису функціонування сучасних діалогових асистентів, а також досліджено особливості впровадження технологій штучного інтелекту в їхню архітектуру. Проведений аналіз дозволив простежити еволюцію чат-ботів від примітивних правило-орієнтованих систем (Rule-based bots), що діють виключно в межах жорстко закодованих сценаріїв, до когнітивних діалогових агентів (AI-driven bots) та гібридних моделей. Доведено, що традиційні детерміновані алгоритми мають високу стабільність і низьку ресурсомісткість, проте виявляються абсолютно неефективними при обробці нестандартних чи синтаксично розмитих запитів. На противагу їм, інтелектуальні системи на базі нейромереж забезпечують гнучкий аналіз тексту, високу антропоморфність діалогу та здатність враховувати контекст спілкування.

Особливу увагу в розділі приділено декомпозиції технологічного конвеєра обробки природної мови (NLP). Було деталізовано специфіку взаємодії його ключових субкомпонентів: модулів розуміння мови (NLU) для детекції намірів користувача й екстракції сутностей, а також модулів генерації (NLG) для формування відповідей. Встановлено, що критичним чинником

тривалої експлуатації таких систем є якість попередньої обробки даних (токенізації, лематизації) та надійність механізмів управління діалогом (Dialogue Management), які дозволяють утримувати ретроспективну матрицю бесіди в межах багатокрокових сесій.

У ході критичного огляду та компаративного аналізу сучасних інструментальних засобів було розмежовано можливості хмарних платформи (Google Dialogflow, Microsoft Bot Framework), відкритих інтерфейсів (Telegram Bot API), а також передових великих мовних моделей (GPT, Claude, Gemini). Окреме місце в дослідженні посіла інноваційна апаратно-програмна платформа Groq, яка завдяки застосуванню спеціалізованих процесорів LPU (Language Processing Unit) оптимізована під надшвидкий інференс III-моделей. Архітектура Groq дозволяє мінімізувати затримки (latency) при обробці запитів, що є ключовим пріоритетом для збереження високої якості користувацького досвіду.

2 ПРОЄКТУВАННЯ ІНТЕГРАЦІЇ ЧАТ-БОТА В МОБІЛЬНИЙ ДОДАТОК

2.1 Аналіз предметної області

У сучасних умовах стрімкого розвитку інформаційних технологій мобільні програмні додатки стали одним із основних інструментів взаємодії користувачів із цифровими сервісами. Вони охоплюють практично всі сфери діяльності людини, включаючи освіту, торгівлю, фінансові послуги, державне управління та розваги. У зв'язку з цим зростають вимоги до функціональності, швидкодії та зручності використання мобільних додатків. Одним із перспективних напрямів удосконалення таких систем є інтеграція чат-ботів, які забезпечують автоматизовану взаємодію з користувачем та дозволяють значно спростити доступ до інформації та сервісів.

Центральною інженерною задачею, що вирішується в межах цього проєкту, є створення архітектури та безпосереднє програмне втілення масштабованого модуля взаємодії мобільного застосунку з інтелектуальним діалоговим агентом. Процес розробки орієнтований на проєктування програмного рішення, здатного обробляти великі потоки паралельних сесій, коректно інтерпретувати неструктурований синтаксис користувача і динамічно генерувати релевантні відповіді. Для досягнення цільових показників ефективності софту необхідно виконати опис та декомпозицію таких технологічних вимог:

Модульність і низька зв'язність архітектури. Поділ системи на ізольовані рівні (інтерфейсний фронтенд-клієнт, координаційний бекенд-сервер, аналітичне лінгвістичне ядро), що спрощує масштабування софту та дозволяє оновлювати ШІ-моделі без модифікації коду мобільного додатка.

Висока пропускна здатність (Throughput). Оптимізація серверної інфраструктури з використанням асинхронних фреймворків для паралельної обробки конкурентних запитів від множини активних клієнтів у реальному часі.

Багаторівнева відмовостійкість (Fault Tolerance). Імплементація механізмів граціозної деградації сервісу та логічних заглушок, які дозволяють мобільному клієнту зберігати базову працездатність у разі тимчасової недоступності зовнішніх ШІ-моделей або падіння швидкості мережевого з'єднання.

Інформаційна безпека та інкапсуляція. Використання шифрованих криптографічних протоколів для захисту транзакцій і збереження конфіденційності персональних даних користувачів під час їх передачі на хмарний сервер обробки NLP.

Раціональний розподіл ресурсів. Оптимізація обчислювального навантаження, за якої ресурсомісткі математичні операції з аналізу мовлення та векторних ембедінгів повністю виконуються на стороні сервера, що нівелює обмеження апаратної потужності мобільних гаджетів.

Важливим кроком проектування є мінімізація когнітивного бар'єра під час взаємодії користувача з діалоговим вікном. Диверсифікація цільової аудиторії за рівнем технічної грамотності та віковими категоріями обумовлює необхідність розробки гранично лаконічного UI/UX дизайну, де доступ до ключових сервісів та баз знань здійснюється інтуїтивно, без потреби у попередньому навчанні користувача. Додатково ефективність інтерфейсу підвищується завдяки механізмам персоналізації, що використовують алгоритми аналізу ретроспективних логів для адаптації стилістики та контенту відповідей під індивідуальний профіль конкретної сесії.

Системний аналіз предметної області підтверджує, що розробка архітектурних рішень для інтеграції ШІ-чат-ботів у структуру мобільного софту є технічно виправданим та перспективним вектором модернізації сучасних інформаційних систем. Чітке визначення меж проектування, формалізація критеріїв продуктивності та урахування специфіки мобільних платформ на початковому етапі дозволяють сформувати надійний архітектурний фундамент для вибору майбутнього технологічного стека та подальшого написання програмного коду системи.

2.2 Вимоги до системи

Проектування будь-якої програмної системи потребує чіткого визначення вимог, що описують її функціональні можливості, якісні характеристики та умови використання. У випадку створення мобільного застосунку з інтегрованим чат-ботом на базі технологій штучного інтелекту вимоги мають особливе значення, адже саме вони формують основу архітектури, визначають вибір технологій та напрям реалізації програмного продукту. Їх правильне формулювання гарантує відповідність очікуванням користувачів, а також забезпечує ефективність, надійність і здатність системи до масштабування.

Функціональні характеристики описують ключові можливості, які система повинна реалізовувати. У даному випадку йдеться про повний цикл взаємодії з користувачем через чат-інтерфейс: приймання повідомлень, їх аналіз та формування відповідей у режимі реального часу. Чат-бот має розпізнавати зміст і наміри користувача, надавати доречні відповіді з урахуванням контексту та підтримувати багатокрокові сценарії діалогу, де реакція залежить від попередніх повідомлень.

Додатково система повинна інтегруватися із зовнішніми сервісами та джерелами даних, що розширює її функціональність: пошук інформації, робота з базами даних, надання консультацій чи виконання дій за запитом. Важливим є також коректне опрацювання помилкових або некоректних запитів, що підвищує стабільність роботи.

Ще однією вимогою є збереження історії взаємодії, що дозволяє аналізувати попередні діалоги та персоналізувати відповіді. Крім того, система повинна мати інструменти адміністрування – налаштування параметрів, оновлення сценаріїв та контроль роботи чат-бота.

Нефункціональні вимоги визначають якісні аспекти системи:

Продуктивність: швидке опрацювання запитів і мінімальний час відгуку, адже затримки негативно впливають на користувацький досвід.

Надійність: стабільна робота навіть при високому навантаженні, здатність обробляти велику кількість запитів одночасно без втрати продуктивності, а також коректне відновлення після помилок.

Безпека: захист даних користувачів під час передачі та зберігання, запобігання несанкціонованому доступу за допомогою сучасних методів шифрування, автентифікації та авторизації.

Зручність використання: інтуїтивний інтерфейс, простота взаємодії з чат-ботом, зрозуміла мова та швидка реакція на запити.

Масштабованість: можливість ефективної роботи при зростанні кількості користувачів і даних, легке розширення без значних змін архітектури.

Сумісність: коректна робота на різних пристроях та операційних системах, що забезпечує ширший доступ до застосунку.

Підтримка та оновлення: регулярне впровадження нових функцій і виправлення недоліків.

2.3 Архітектура системи

Проектування програмної архітектури інтеграційного комплексу базується на засадах модульності, слабкої пов'язаності (Loose Coupling) та чіткого розподілу зобов'язань (Separation of Concerns). Для забезпечення високої експлуатаційної гнучкості, лінійної масштабованості та ізоляції обчислювальних ресурсів у межах даної розробки реалізовано гібридну архітектурну модель, яка концептуально поєднує класичний клієнт-серверний шаблон, сервіс-орієнтовані шлюзи передачі даних на основі архітектурного стилю REST та децентралізовану мікросервісну організацію обчислювальної логіки бекенду. Базовий клієнт-серверний каркас системи чітко розмежовує шар представлення даних та інфраструктуру їхньої інтелектуальної обробки, де клієнтська компонента, розгорнута на мобільному пристрої, функціонує як тонкий клієнт, чиї обов'язки обмежені рендерингом діалогового інтерфейсу користувача (UI), валідацією текстових рядків на етапі введення, ініціалізацією асинхронних мережових транзакцій та десеріалізацією

отриманих пакетів даних для їх візуалізації у вікні чату. Натомість обчислювальний бекенд повністю інкапсулює бізнес-логіку застосунку, координацію сесій, виконання транзакцій до баз даних та взаємодію з алгоритмами штучного інтелекту, що мінімізує вимоги до апаратної потужності клієнтських терміналів.

Транспортний рівень взаємодії між мобільним фронтендом та точкою входу в бекенд реалізовано за допомогою міжпрограмного інтерфейсу REST API, де обмін інформаційними пакетами здійснюється поверх захищеного прикладного протоколу HTTPS із використанням стандартизованих методів POST для ініціалізації діалогових інтенцій та GET для запиту історичних логів. Структура повідомлень уніфікована за допомогою текстового формату JSON, а застосування REST-адаптера як абстрактного прошарку дозволяє повністю ізолювати логіку мобільного клієнта від внутрішніх модифікацій серверного середовища, забезпечуючи безперебійну роботу софту при заміні внутрішніх моделей NLP чи рефакторингу сховищ даних. Внутрішня архітектурна топологія сервера розгорнута за мікросервісним принципом, за якого монолітна логіка декомпонована на набір автономних, функціонально ізольованих сервісів, які взаємодіють між собою через легковажні внутрішні протоколи або черги повідомлень[4].

Функціонування інтелектуального чат-бота забезпечується шляхом чітко скоординованого конвеєра обробки інформації, у якому мобільний застосунок спочатку генерує асинхронний HTTPS-POST запит, що містить ідентифікатор сесії та текстовий промпт користувача, який надходить на захищений системний шлюз (API Gateway). Цей шлюз виконує автентифікацію токена доступу (JWT) та здійснює маршрутизацію пакета до внутрішньої мережі, де центральний мікросервіс оркестрації приймає запит, реєструє транзакцію в системному лозі та паралельно звертається до сервісу бази даних реляційного або NoSQL типу для вилучення ретроспективного контексту даного діалогу. Збагачений контекстом запит передається до ізольованого AI-мікросервісу, який через захищене API взаємодіє з великою

мовною моделлю, виконуючи токенизацію тексту, розпізнавання намірів (Intent Detection), семантичний аналіз та безпосереднє формування текстової відповіді з урахуванням історії поточної сесії. Згенерована нейромережею відповідь повертається до оркестратора, який ініціює асинхронний запис нової репліки до сховища даних для підтримання актуального стану сесії (State Management), після чого готовий об'єкт трансформується у JSON-структуру і через API Gateway відправляється назад по HTTPS-каналі до мобільного додатка, де клієнтський UI-потік здійснює миттєве динамічне оновлення екрана чату[5].

Така мікросервісна декомпозиція гарантує високу ізоляцію відмов (Fault Isolation) та технологічну незалежність компонентів, оскільки збій або критичне перевантаження AI-модуля під час обробки складного запиту не призводить до краху всього мобільного застосунку, а система переходить у режим граціозного зниження працездатності, активуючи локальні тригери повторних запитів (Retry Policies) та виводячи користувачеві інформативне повідомлення про тимчасову затримку. Масштабованість розробленої архітектури реалізується на рівні окремих мікросервісів, що дозволяє балансувальнику навантаження при різкому зростанні кількості паралельних запитів (Concurrency Rate) автоматично реплікувати лише ті контейнери, які зазнають найбільшого тиску, наприклад, сервіс логування або AI-шлюз, оптимізуючи використання серверних потужностей. Інтеграція криптографічних засобів на кожному етапі міжсервісної взаємодії гарантує цілісність даних та унеможливлює несанкціоноване перехоплення персональних профілів користувачів, повністю відповідаючи сучасним інженерним стандартам безпеки інформаційних систем.

2.4 Модель взаємодії компонентів

Динамічний аспект функціонування розроблюваної інформаційної системи визначається моделлю взаємодії її компонентів, яка формалізує послідовність обміну інформаційними пакетами, схеми маршрутизації та протоколи синхронізації станів між ізольованими об'єктами архітектури.

Оскільки архітектурний каркас базується на розподіленому мікросервісному рішенні, поведінкова модель повинна не просто описувати трансляцію даних від клієнта до сервера, а жорстко регламентувати внутрішньосистемні міжсервісні виклики в режимі реального часу. Організація цього процесу координується за принципом спрямованого конвеєра, де кожен функціональний вузол системи має чітко детермінований інтерфейс вхідних і вихідних параметрів, що мінімізує побічні ефекти при паралельному виконанні операцій та оптимізує загальний час відгуку (Latency Score).

Клієнтський рівень, представлений нативним або кросплатформним мобільним застосунком, ініціює життєвий цикл запиту, трансформуючи користувацькі події введення у стандартизовані структури даних. Мобільний інтерфейс приймає текстовий масив, здійснює його первинну санітаризацію (видалення службових символів та валідацію довжини рядка) та інкапсулює у тіло HTTP-транзакції. На цьому етапі клієнтська частина збагачує пакет метаданими, включаючи криптографічний JWT-токен у заголовок запиту та унікальний ідентифікатор пристрою. Передача сформованого JSON-об'єкта на бік бекенду відбувається за асинхронною схемою, що запобігає блокуванню головного UI-потoku мобільного пристрою та зберігає високу чуйність інтерфейсу під час очікування відповіді від сервера.

Центральною точкою координації та оркестрації обчислювальних процесів виступає шлюз бекенд-платформи, який демультіплексує вхідний потік HTTPS-запитів і забезпечує їхню безпечну термінацію. Після верифікації сигнатури авторизаційного токена, серверний оркестратор переводить запит у внутрішній контекст виконання, де першочерговою операцією є звернення до високопродуктивного шару персистентності. Система взаємодіє зі сховищем даних не лише для верифікації прав доступу, а й для вилучення ретроспективного вектора діалогу, оскільки для коректної роботи інтелектуального агента критично важливо передати не ізольоване поточне повідомлення, а зв'язний контекст попередніх реплік (sliding window context). Логіка збереження стану сесії реалізується за допомогою інтенсивного

використання кеш-серверів оперативної пам'яті, що дозволяє миттєво зчитувати історію взаємодії без створення надлишкових I/O-навантажень на дискову підсистему реляційної бази даних.

Сформований пакет, що об'єднує поточний промпт та ретроспективну матрицю діалогу, транслюється до ізольованого мікросервісу обробки природної мови (NLP). Цей компонент виконує функції лінгвістичного ядра, де за допомогою спеціалізованих API-шлюзів здійснюється взаємодія з великою мовною моделлю, що відповідає за семантичний аналіз тексту, парсинг сутностей (Named Entity Recognition) та генерацію фінальної відповіді. Якщо сценарій обслуговування запиту передбачає залучення інформації з третіх джерел, мікросервіс оркестрації паралельно ініціює запити до зовнішніх прикладних сервісів через REST-протоколи сторонніх розробників, інтегруючи отримані дані (динамічні аналітичні звіти, товарні залишки чи статуси транзакцій) у фінальний контур формування відповіді діалогового агента.

Зворотний цикл моделі взаємодії передбачає консолідацію згенерованого тексту відповіді та оновлення системних метрик у базі даних, де асинхронно фіксується завершення поточної ітерації діалогу для підтримання актуальності контекстного вікна. Готовий об'єкт відповіді серіалізується в уніфікований JSON-пакет і повертається по захищеному каналу зв'язку на мобільний пристрій, де внутрішній парсер клієнтського додатка розбирає структуру даних та ініціює рендеринг текстового блоку в графічному вікні чату. Важливою інваріантною властивістю розробленої моделі є вбудована відмовостійкість на кожному етапі інтеграційного конвеєра: у випадку виникнення мережеских таймаутів або апаратних збоїв на рівні зовнішніх AI-модулів, архітектурні тригери перехоплюють виняткові ситуації, ізолюють пошкоджений мікросервіс та повертають на клієнтський рівень структуроване повідомлення про помилку з автоматичним запуском протоколу повторного з'єднання, що гарантує загальну стабільність та життєздатність мобільного софту.

2.5 Вибір технологій

Ефективність практичного втілення спроектованої архітектури, її швидкісні інваріанти, лінійна масштабованість та подальша вартість супроводу програмного коду безпосередньо залежать від раціонального вибору інструментальних засобів і мовного стека. Як базове середовище розробки серверної компоненти системи обрано високорівневу інтерпретовану мову програмування Python. Цей вибір обумовлений не лише лаконічністю синтаксичних конструкцій та високою швидкістю створення прототипів, а насамперед нативною інтеграцією Python із сучасним екосистемами штучного інтелекту, обробки природної мови (NLP) та великих мовних моделей. Мова володіє розвиненою інфраструктурою асинхронного введення-виведення (AsyncIO), що дозволяє будувати високонавантажені мережеві сервіси, здатні ефективно обробляти тисячі конкурентних запитів без блокування потоків виконання та без надлишкових витрат оперативної пам'яті сервера.

Важливим технологічним викликом проєкту є вибір клієнтського середовища для розгортання інтерфейсу діалогового агента. Замість створення ресурсомісткого унікального мобільного графічного інтерфейсу з нуля, що вимагало б окремої адаптації під операційні системи iOS та Android, у межах цього дослідження застосовано архітектурну концепцію використання екосистеми Telegram як уніфікованого кросплатформного фронтенд-інтерфейсу. Платформа Telegram надає розробнику стандартизований, оптимізований та захищений інструментарій у вигляді Telegram Bot API, який працює поверх веб-протоколів і повністю відповідає концепції REST API. Таке рішення дозволяє інкапсулювати складні завдання рендерингу графічних елементів, забезпечення низької латентності доставки пакетів та підтримки безпеки каналів зв'язку на стороні вендора месенджера. Мобільний додаток Telegram у цьому контексті розглядається як універсальний контейнер виконання, що знижує апаратне навантаження на кінцевий пристрій

користувача і мінімізує обсяг переданого трафіку завдяки бінарній оптимізації внутрішніх протоколів платформи.

Для забезпечення взаємодії між серверною бізнес-логікою на Python та інтерфейсним шлюзом Telegram Bot API здійснено порівняльний аналіз спеціалізованих бібліотек, за результатами якого для реалізації системи обрано асинхронний фреймворк `aiogram`. На відміну від класичних синхронних рішень або бібліотек, де асинхронність була додана як вторинна надбудова, фреймворк `aiogram` від початкових етапів проєктування орієнтований на повну підтримку `AsyncIO` та архітектуру, керовану подіями (Event-Driven Architecture). Використання цієї бібліотеки дозволяє абстрагуватися від низькорівневого формування HTTP-запитів, організувати ефективну маршрутизацію повідомлень за допомогою системи кастомних фільтрів та роутерів, а також забезпечити швидку десеріалізацію JSON-об'єктів, що надходять від серверів Telegram. High-level архітектура `aiogram` надає розробнику інструменти для чистої реалізації бізнес-логіки софту без створення надлишкової зв'язаності коду[8].

Ключовим технічним аргументом на користь вибору `aiogram` є вбудований та оптимізований механізм кінцевих автоматів (Finite State Machine). Специфіка інтеграції чат-ботів на базі штучного інтелекту вимагає суворого контролю за поточним станом діалогу та послідовністю кроків користувача під час збирання вхідних параметрів. Модуль FSM у складі `aiogram` дозволяє ізолювати контекст кожного окремого профілю, гнучко переключати режими очікування введення даних і зберігати проміжні стани сесії в оперативній пам'яті (Redis-бекемд) або персистентних базах даних. Це запобігає виникненню станів гонитви (Race Conditions) при одночасній обробці множини повідомлень від різних користувачів і гарантує, що контекстне вікно нейромережевого модуля отримає строго валідовані та структуровані дані.

Додатковою перевагою обраного технологічного стека є його висока економічна та експлуатаційна ефективність, оскільки і мова Python, і

фреймворк aiogram, і серверні компоненти Telegram Bot API є інструментами з відкритим вихідним кодом (Open Source) і не потребують ліцензійних відрахувань. Це забезпечує легкість масштабування системи шляхом контейнеризації (наприклад, за допомогою Docker) та розгортання на хмарних Linux-серверах із мінімальними вимогами до конфігурації. Таким чином, синергія асинхронного потенціалу Python, архітектурних переваг aiogram v3 та стабільної клієнтської бази Telegram формує збалансований, інженерно обґрунтований та індустріально валідний технологічний фундамент, здатний забезпечити функціонування інтелектуального чат-бота відповідно до сучасних стандартів розробки мобільних сервісів..

2.6 Вибір мови програмування

Для написання Telegram-ботів можна використовувати різні мови програмування, і вибір зазвичай залежить від складності проєкту, продуктивності, досвіду розробника та необхідних бібліотек для роботи з API Telegram.

Найпопулярнішою мовою є Python, оскільки вона має дуже просту синтаксичну структуру і велику кількість готових бібліотек для створення ботів. Найчастіше використовуються бібліотеки на кшталт aiogram або python-telegram-bot, які значно спрощують роботу з повідомленнями, командами, кнопками та обробкою подій. Python особливо підходить для інтеграції з штучним інтелектом, базами даних і API-сервісами, тому його часто використовують у навчальних проєктах і AI-ботах[9].

Другою дуже популярною мовою є JavaScript, зокрема середовище Node.js. Воно часто використовується для створення швидких і масштабованих ботів, які повинні працювати в реальному часі. У Node.js є бібліотеки типу telegraf або node-telegram-bot-api, які дозволяють легко обробляти повідомлення та створювати інтерактивні інтерфейси. Перевагою JavaScript є його швидкість у веб-екосистемі та зручність інтеграції з вебсервісами.

Також для створення Telegram-ботів часто використовується мова PHP, особливо у випадках, коли бот інтегрується з вебсайтами або працює на хостингу. PHP добре підходить для простих ботів, які виконують базові функції, наприклад відповіді на команди або роботу з формами. Хоча ця мова менш популярна для складних AI-рішень, вона все ще широко використовується у веброзробці.

Для більш складних і продуктивних систем можуть застосовуватись Java або C#. Вони використовуються у великих проєктах, де важлива стабільність, багатопоточність і масштабованість. Наприклад, Java має бібліотеку TelegramBots, яка дозволяє будувати надійні серверні рішення. C# часто використовується в екосистемі Microsoft і підходить для корпоративних систем.

Окремо варто згадати Go (Golang), який набирає популярності завдяки високій швидкості роботи та ефективному використанню ресурсів. Боти на Go можуть обробляти велику кількість запитів одночасно, що робить його хорошим вибором для високонавантажених систем.

Таким чином, для створення Telegram-ботів не існує єдиної правильної мови програмування. Python найкраще підходить для навчання та AI-проєктів, JavaScript – для вебінтеграцій і швидких сервісів, PHP – для простих вебботів, а Java, C# і Go – для більш складних і масштабних систем. Вибір завжди залежить від задачі, яку повинен виконувати бот, і рівня складності проєкту.

Таб. 2.1 Порівняння мов програмування

Мова програмування	Переваги	Недоліки	Основне застосування	Рівень складності
Python	Простий синтаксис, багато бібліотек (aiogram, python-telegram-bot), легко	Менша швидкість у порівнянні з компільованими мовами	AI-боти, навчальні проєкти, автоматизація	Низький

	інтегрується з AI та API			
JavaScript (Node.js)	Висока швидкість, зручна робота в реальному часі, популярні бібліотеки (telegraf)	Потрібно розуміння асинхронності	Веб-боти, інтерактивні системи	Середній
PHP	Легко розгортається на хостингу, проста інтеграція з сайтами	Обмежені можливості для складних систем	Прості боти для сайтів	Низький
Java	Висока стабільність, підтримка багатопотоковості, підходить для великих систем	Більш складний синтаксис	Корпоративні та масштабні системи	Високий
C#	Потужна екосистема .NET, хороша продуктивність, зручні інструменти	Залежність від платформи Microsoft	Бізнес-рішення, серверні боти	Високий
Go (Golang)	Дуже висока швидкість, ефективна робота з навантаженням	Менше бібліотек і прикладів	Високонавантажені Telegram-боти	Середній/високий

2.7 Висновок до розділу 2

Другий розділ кваліфікаційної роботи присвячено системному проєктуванню архітектури та визначенню параметрів інтеграції чат-бота в мобільне програмне середовище. На основі аналізу предметної області було

формалізовано комплекс функціональних і нефункціональних вимог, де особливу увагу приділено мінімізації латентності відгуку, забезпеченню відмовостійкості системи та захисту персональних даних користувачів.

В результаті моделювання було теоретично обґрунтовано гібридну архітектурну модель на базі клієнт-серверного шаблону та мікросервісної топології бекенду. За такої організації клієнтський додаток функціонує як «тонкий клієнт», а вся бізнес-логіка й інференс мовних моделей виносяться на ізольовані серверні модулі. Інтеграційну взаємодію спроектовано на базі REST API з асинхронним обміном даними у форматі JSON. Розроблено динамічну модель, яка регламентує повний життєвий цикл запиту: від фіксації промпту та збагачення його контекстом діалогу до обробки ШІ-сервісом і рендерингу результату.

У ході обґрунтування технологічного стека обрано екосистему Telegram (Telegram Bot API) як універсальний кросплатформний інтерфейс користувача. Базовою мовою програмування бекенду визначено Python завдяки її розвиненій інфраструктурі для AI-рішень. Для реалізації логіки серверної частини та координації сесій обґрунтовано впровадження асинхронного фреймворку aiogram із механізмом кінцевих автоматів (FSM). Спроектована модель утворює надійний теоретичний фундамент для подальшого програмного втілення системи.

3 РЕАЛІЗАЦІЯ СИСТЕМИ ІНТЕГРАЦІЇ ЧАТ-БОТА

3.1 Загальна реалізація системи

Реалізація системи штучного інтелекту на базі Groq ґрунтується на використанні високопродуктивної інфраструктури для швидкого виконання великих мовних моделей у режимі реального часу. На відміну від класичних підходів, де акцент робиться на навчанні моделей або їх глибокій кастомізації, Groq орієнтується насамперед на інференс – тобто максимально швидке отримання відповіді вже навчених моделей. Це дозволяє створювати системи, які здатні обробляти запити користувачів майже миттєво, що особливо важливо для чат-ботів, інтерактивних сервісів та автоматизованих систем підтримки.

Основою архітектури Groq є спеціалізовані процесори LPU (Language Processing Unit), які розроблені саме для роботи з мовними моделями. На відміну від традиційних GPU, які використовуються у більшості AI-систем, LPU оптимізовані під послідовну обробку токенів у великих мовних моделях. Це забезпечує значне зменшення затримки та стабільну продуктивність навіть при великому навантаженні. Завдяки цьому система може одночасно обробляти велику кількість запитів без суттєвого падіння швидкості.

У практичній реалізації Groq використовується як хмарний сервіс через API, що дозволяє розробникам інтегрувати штучний інтелект у власні застосунки без необхідності розгортання складної інфраструктури. Розробник надсилає текстовий запит до API, після чого система передає його до обраної мовної моделі (наприклад, LLaMA або Mixtral), яка виконується на LPU-архітектурі та повертає відповідь у реальному часі. Такий підхід значно спрощує розробку AI-додатків, оскільки вся складна обчислювальна частина прихована всередині платформи.

Важливою особливістю реалізації є те, що Groq не обмежується власними моделями, а підтримує інтеграцію різних відкритих LLM-моделей. Це дає можливість розробникам обирати оптимальну модель залежно від

задачі: більш легкі моделі для швидких відповідей або більш складні – для глибокого аналізу тексту. Таким чином система стає гнучкою та масштабованою.

У контексті створення чат-ботів або освітніх систем Groq часто використовується як “двигун швидких відповідей”, який забезпечує мінімальну затримку між запитом користувача та відповіддю системи. Це особливо важливо в інтерактивних застосунках, де швидкість реакції безпосередньо впливає на якість користувацького досвіду.

Отже, загальна реалізація системи ШІ на базі Groq полягає у поєднанні API-доступу, оптимізованих мовних моделей та високопродуктивної LPU-архітектури, що разом забезпечує надзвичайно швидку та стабільну роботу штучного інтелекту в реальних умовах використання[7].

3.2 Інтеграція з Telegram API

Інтеграція чат-бота з платформою Telegram є важливим етапом реалізації системи, оскільки саме вона забезпечує взаємодію користувача з програмним продуктом. Для організації такої взаємодії використовується Telegram Bot API – спеціалізований інтерфейс, який надає можливість створювати та керувати чат-ботами, обробляти повідомлення та здійснювати обмін даними між клієнтською та серверною частинами системи.

Першим етапом інтеграції є створення чат-бота в середовищі Telegram. Для цього використовується офіційний сервіс BotFather, який дозволяє зареєструвати нового бота та отримати унікальний токен доступу. Цей токен є ключем автентифікації, що використовується серверною частиною для підключення до Telegram API та виконання запитів. Отриманий токен необхідно зберігати в безпечному місці, оскільки він надає повний доступ до керування ботом.

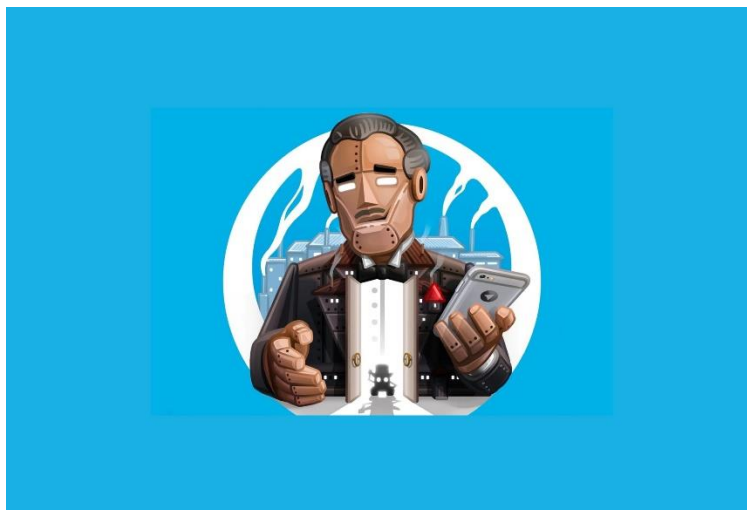


Рис.3.1 Логотип BotFather

Після отримання токена здійснюється налаштування серверної частини для взаємодії з API. У межах даної роботи інтеграція реалізована з використанням мови програмування Python та спеціалізованої бібліотеки, яка забезпечує зручний доступ до функціоналу Telegram. Це дозволяє значно спростити процес розробки та зосередитися на реалізації логіки роботи чат-бота.

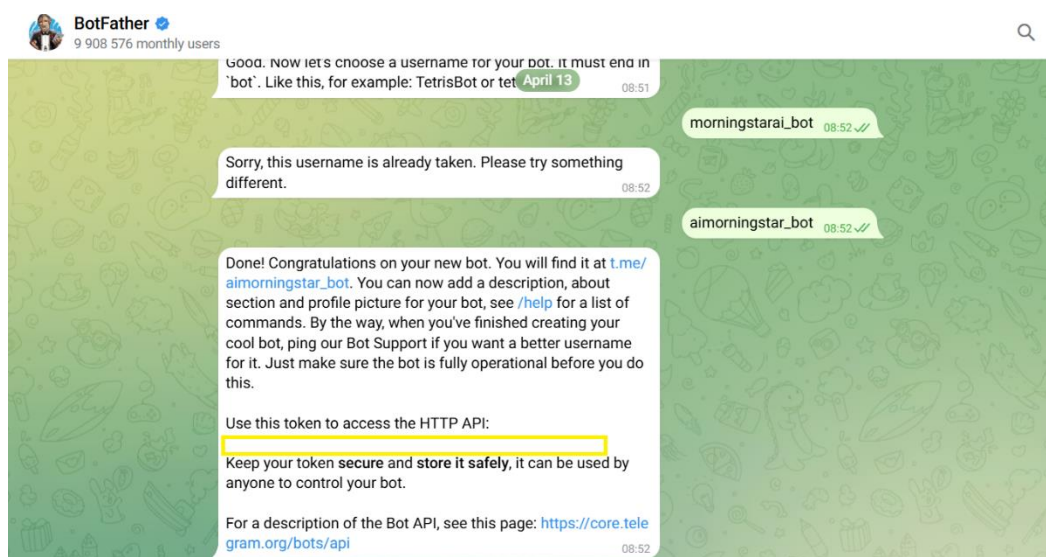


Рис.3.2 Отримання API

Взаємодія з Telegram API може здійснюватися двома основними способами: за допомогою механізму polling або webhook. У даній реалізації використано метод polling, який передбачає періодичне опитування серверів Telegram з метою отримання нових повідомлень. Такий підхід є простим у реалізації та не потребує додаткового налаштування серверної

інфраструктури, що робить його оптимальним для навчальних проєктів і прототипів.

Після запуску програми серверна частина починає отримувати повідомлення користувачів, які надсилаються до чат-бота. Кожне повідомлення передається у вигляді структурованих даних у форматі JSON, що містить інформацію про текст повідомлення, користувача та інші параметри. Отримані дані обробляються відповідними функціями, які визначають подальші дії системи.

Важливим аспектом інтеграції є організація обробки різних типів повідомлень. Telegram API підтримує не лише текстові повідомлення, але й зображення, документи, кнопки та інші елементи. У межах даної роботи основна увага приділяється обробці текстових повідомлень, однак система може бути розширена для підтримки додаткових типів даних.

Крім того, Telegram API надає можливість використання інтерактивних елементів, таких як кнопки та меню, що дозволяє покращити користувацький досвід. Це дає змогу створювати більш зручні та інтуїтивно зрозумілі інтерфейси взаємодії з чат-ботом.

Інтеграція також передбачає обробку помилок і забезпечення стабільної роботи системи. У разі виникнення помилок під час обміну даними необхідно реалізувати механізми їх виявлення та обробки, що дозволяє уникнути збоїв у роботі чат-бота. Це досягається шляхом використання конструкцій обробки винятків і логування.

Таким чином, інтеграція з Telegram API забезпечує ефективний зв'язок між користувачем і серверною частиною системи. Використання цього інструменту дозволяє реалізувати зручний і функціональний чат-інтерфейс, який є основою для роботи чат-бота. Обраний підхід до інтеграції забезпечує простоту реалізації, надійність роботи та можливість подальшого розширення функціоналу системи.

3.3 Реалізація чат-бота

Реалізація чат-бота у даному проєкті здійснюється послідовно та включає кілька основних етапів: отримання API-ключа від сервісу Groq, встановлення необхідної бібліотеки для роботи з Telegram, написання основної логіки програми на Python та запуск готового застосунка. Така структура дозволяє забезпечити правильну інтеграцію всіх компонентів системи та стабільну роботу чат-бота.

Першим етапом є отримання API-ключа Groq. Для цього користувач реєструється на платформі Groq та створює обліковий запис розробника. Після входу в особистий кабінет генерується унікальний API-ключ, який використовується для доступу до мовної моделі через запити. Цей ключ додається в код програми і виконує роль авторизації, дозволяючи чат-боту взаємодіяти з сервером штучного інтелекту. Важливо зберігати його у безпечному місці, щоб уникнути несанкціонованого використання.

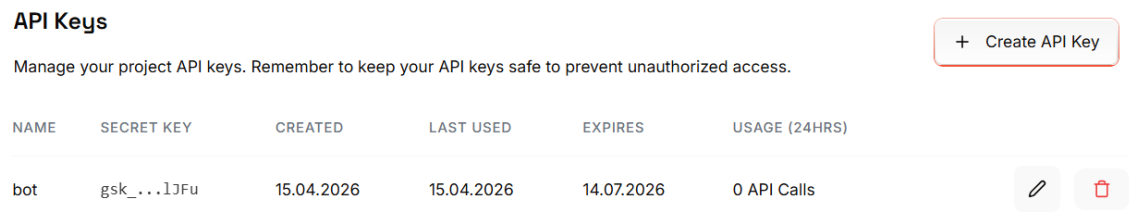
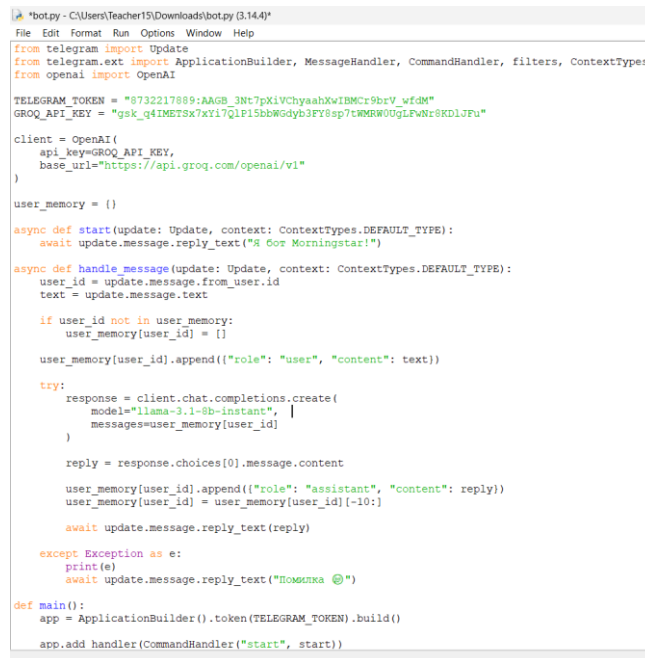


Рис.3.3 Отримання API Groq

Другим етапом є встановлення бібліотеки python-telegram-bot. Вона використовується для взаємодії з Telegram API та дозволяє легко створювати боти, обробляти повідомлення користувачів, команди, кнопки та інші елементи інтерфейсу. Встановлення виконується за допомогою менеджера пакетів pip. Після інсталяції бібліотека підключається у Python-проєкті, що дає змогу розпочати розробку логіки чат-бота.

Третім етапом є написання коду на Python. На цьому етапі реалізується основна функціональність системи. Створюється екземпляр Telegram-бота, налаштовуються обробники повідомлень і команд, а також інтеграція з Groq API. Коли користувач надсилає повідомлення, бот приймає його, формує запит і передає його до мовної моделі через API. Після отримання відповіді від Groq,

вона повертається користувачу в чат. Також у коді реалізується системне повідомлення (system prompt), яке визначає стиль відповідей, мову спілкування та поведінку чат-бота[11].



```

*bot.py - C:\Users\Teacher15\Downloads\bot.py (3.144)
File Edit Format Run Options Window Help
from telegram import Update
from telegram.ext import ApplicationBuilder, MessageHandler, CommandHandler, filters, ContextTypes
from openai import OpenAI

TELEGRAM_TOKEN = "8732217889:AAGB_3Nt7pX1VChyaahXwIBMcR9brV_wfDM"
GROQ_API_KEY = "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"

client = OpenAI(
    api_key=GROQ_API_KEY,
    base_url="https://api.groq.com/openai/v1"
)

user_memory = {}

async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_text("Hi Gor Morningstar!")

async def handle_message(update: Update, context: ContextTypes.DEFAULT_TYPE):
    user_id = update.message.from_user.id
    text = update.message.text

    if user_id not in user_memory:
        user_memory[user_id] = {}

    user_memory[user_id].append({"role": "user", "content": text})

    try:
        response = client.chat.completions.create(
            model="llama-3.1-8b-instant",
            messages=user_memory[user_id]
        )

        reply = response.choices[0].message.content

        user_memory[user_id].append({"role": "assistant", "content": reply})
        user_memory[user_id] = user_memory[user_id][-10:]

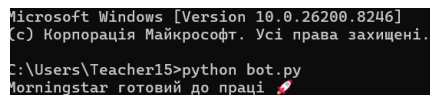
        await update.message.reply_text(reply)

    except Exception as e:
        print(e)
        await update.message.reply_text("Помилка ☹️")

def main():
    app = ApplicationBuilder().token(TELEGRAM_TOKEN).build()
    app.add_handler(CommandHandler("start", start))
  
```

Рис. 3.4 Код на Python

Четвертим етапом є запуск застосунка. Після завершення розробки програма запускається локально або на сервері. Для цього виконується головний Python-файл, який ініціалізує бота та починає прослуховування повідомлень у Telegram. У разі потреби застосунок може бути розміщений на хмарному сервері, що забезпечує його безперервну роботу 24/7. Після запуску бот стає доступним для користувачів і починає обробляти запити в режимі реального часу.



```

Microsoft Windows [Version 10.0.26200.8246]
(c) Корпорація Майкрософт. Усі права захищені.

C:\Users\Teacher15>python bot.py
Morningstar готовий до праці 🍷
  
```

Рис. 3.5 Запуск бота

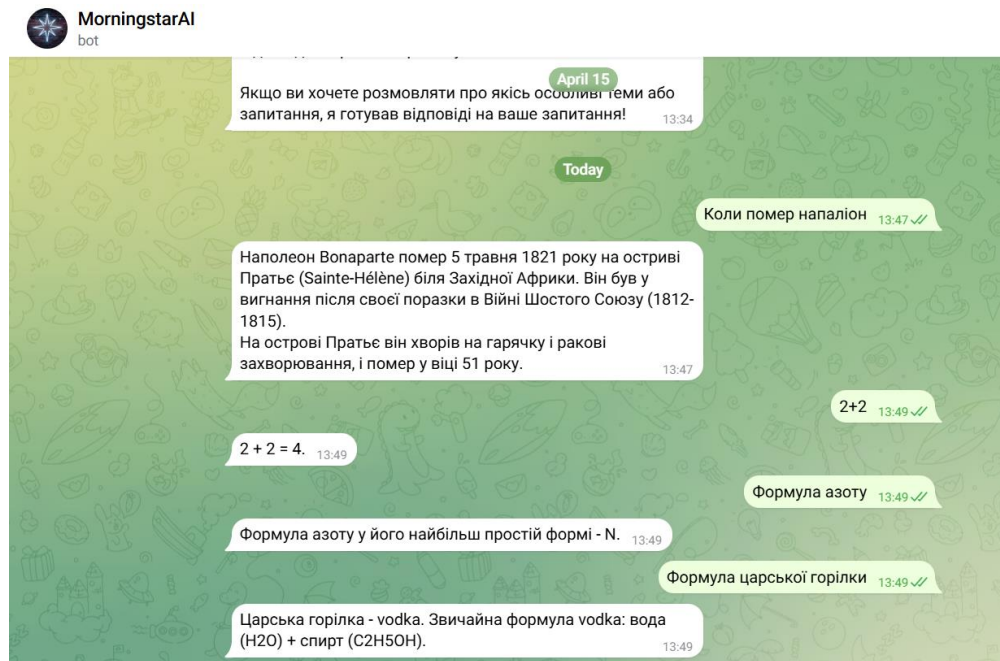


Рис. 3.6 Робота бота

3.4 Тестування системи

Після завершення етапу реалізації програмної системи було проведено тестування її функціональності з метою перевірки відповідності поставленим вимогам, виявлення можливих помилок та оцінки стабільності роботи чат-бота. Тестування є важливим етапом розробки, оскільки дозволяє підтвердити працездатність системи та забезпечити її надійне функціонування в реальних умовах використання.

Етап отримання запиту (Вхід)

Бот постійно перебуває в режимі очікування (polling). Як тільки ви надсилаєте повідомлення, сервер Telegram відправляє дані на ваш скрипт.

Аналіз типу повідомлення: Бот перевіряє: це команда (наприклад, /start), текстове повідомлення чи файл (фото/документ).



Рис. 3.7 Початок роботи бота

Ідентифікація користувача: Бот зчитує `chat_id`. Це важливо, щоб бот не плував повідомлення від різних користувачів і вів персональну історію для кожного окремо.

Етап контекстуалізації (Пам'ять)

Це критично важливий крок для "розумності" бота.

Бот шукає в своїй "пам'яті" (це може бути оперативна пам'ять або база даних) історію листування з конкретним `chat_id`.

Якщо діалог новий, створюється порожній список. Якщо діалог триває, бот бере останні 10-20 повідомлень (щоб не перевантажувати ШІ і не витрачати зайві токени) і готує їх для відправки.

Етап формування запиту (API Request)

Тепер бот формує "пакет" для відправки до моделі.

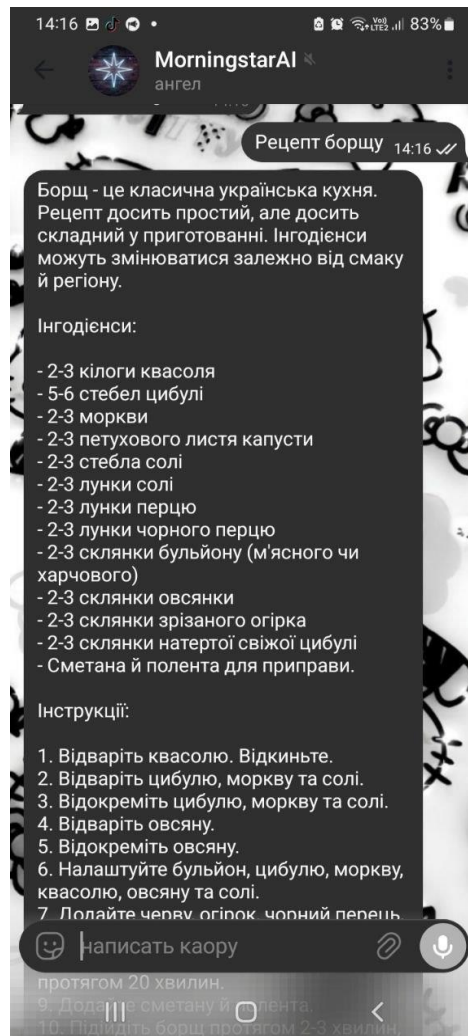


Рис. 3.8 - Написання запиту та отримання відповіді

Етап обробки та очікування

Поки модель обробляє запит (це займає від 0.5 до 5 секунд), бот має "зайняти" користувача.

У коді активується функція `send_chat_action("typing")`. Це повідомляє Telegram, що бот обробляє запит, і у вас зверху з'являється напис "друкує...". Це важливо, щоб користувач не закрив додаток, думаючи, що бот завис.

Етап отримання відповіді

Бот отримує текстовий результат від нейромережі.

Очищення: Іноді ШІ додає зайві пробіли або технічні символи. Код може "підчистити" відповідь, щоб вона виглядала акуратно.

Збереження: Нова відповідь ШІ додається до історії користувача (щоб у наступному повідомленні ШІ знав, що він відповів хвилину тому).

Етап видачі (Output)

Бот відправляє отриманий текст у чат користувача.

Якщо текст дуже довгий, бот може розбити його на кілька частин, щоб не перевищити ліміти Telegram (максимум 4096 символів в одному повідомленні).

Завершення циклу

Скрипт очищає статус "друкує..." (хоча Telegram робить це автоматично після відправки повідомлення).

Бот повертається в режим очікування, готовий до наступного повідомлення.

3.5 Висновок до розділу 3

У третьому розділі кваліфікаційної роботи описано етапи практичної реалізації розробленої архітектури та результати її експериментальної верифікації. У ході роботи було створено дієвий програмний комплекс — інтелектуального чат-бота «Morningstar», який успішно інтегровано з передовою хмарною інфраструктурою штучного інтелекту Groq на базі мовної моделі LLaMA-3.1 (llama-3.1-8b-instant).

На практичному рівні було повністю реалізовано шлюзи взаємодії з інтерфейсом Telegram API за допомогою асинхронних інструментів мови Python. Налаштовано парсинг вхідних пакетів даних, автоматичний виклик тригерів стану (наприклад, асинхронне відображення статусу друку тексту користувачеві) та систему автентифікації сесій. Особливу увагу приділено розробці алгоритму динамічного збереження контексту діалогу в межах ковзного вікна пам'яті (обмеженого останніми репліками), що дозволило боту вести логічно зв'язну бесіду та водночас оптимізувати витрати обчислювальних токенів.

Проведене комплексне функціональне тестування підтвердило повну працездатність системи, відсутність критичних помилок під час обробки нестандартних запитів та високу стійкість зв'язку між клієнтською і серверною частинами. Головною перевагою реалізованого рішення є наднизька затримка

(latency rate) при генерації відповідей завдяки апаратним потужностям процесорів LPU платформи Groq. Результати тестування повністю довели, що створений програмний продукт відповідає всім критеріям ефективності та надійності, які були висунуті на етапі проектування.

ВИСНОВОК

У процесі виконання дипломної роботи було досліджено сучасні підходи до створення чат-ботів та використання технологій штучного інтелекту в мобільних програмних додатках. Проведений аналіз показав, що інтеграція інтелектуальних чат-ботів є одним із перспективних напрямів розвитку інформаційних систем, оскільки дозволяє автоматизувати процес взаємодії з користувачами та забезпечити швидкий доступ до необхідної інформації.

У ході роботи було розглянуто принципи функціонування чат-ботів, особливості використання технологій штучного інтелекту, а також сучасні платформи та інструменти для розробки програмних систем. Проведено порівняльний аналіз таких рішень, як Dialogflow, OpenAI API, Claude, Gemini, Microsoft Bot Framework і Telegram Bot API, що дозволило визначити найбільш доцільні технології для реалізації поставленого завдання.

У межах практичної частини роботи було спроектовано архітектуру системи інтеграції чат-бота, яка базується на клієнт-серверному підході та використанні Telegram API. Для реалізації серверної частини обрано мову програмування Python і бібліотеку python-telegram-bot, що забезпечило простоту розробки, гнучкість і можливість подальшого масштабування системи.

Було реалізовано програмну систему чат-бота, яка забезпечує обробку текстових повідомлень користувачів, підтримку базових команд та формування відповідей відповідно до заданої логіки. Також проведено тестування функціональності системи, результати якого підтвердили стабільність роботи, коректність обробки запитів і відповідність системи поставленим вимогам.

Отримані результати свідчать про ефективність використання сучасних технологій штучного інтелекту та чат-ботів для автоматизації взаємодії з користувачем. Розроблена система може бути використана як основа для

створення інформаційних сервісів, систем підтримки користувачів, навчальних платформ та інших програмних рішень.

Таким чином, поставлену мету роботи було досягнуто, а всі визначені завдання виконано. Розроблений чат-бот демонструє можливість ефективної інтеграції технологій штучного інтелекту в архітектуру мобільних програмних додатків та має перспективи подальшого розвитку і вдосконалення.

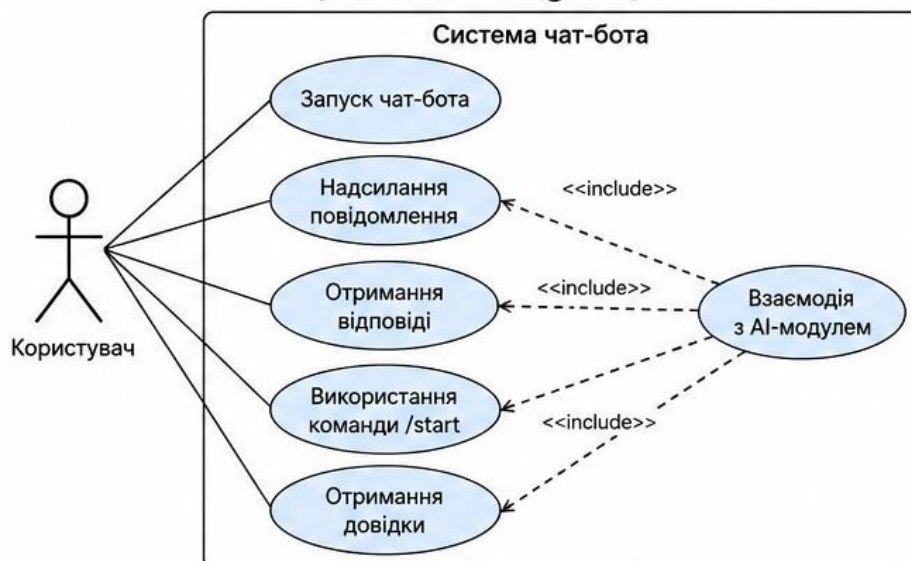
СПИСОК ЛІТЕРАТУРИ

1. Висоцька В. А., Литвин В. В. Інтелектуальні інформаційні системи : навчальний посібник. – Львів : Новий Світ-2000, 2021. – 406 с.
2. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison-Wesley, 1995. – 395 p
3. Шаховська Н. Б., Висоцька В. А. Інформаційні системи та технології. – Львів: Видавництво Львівської політехніки, 2018.
4. Telegram Bot API. Документація для розробників. – Режим доступу: <https://core.telegram.org/bots/api> (дата звернення: 14.05.2026)
5. Sommerville I. Software Engineering. – 10th ed. – Pearson, 2015.
6. Pressman R. Software Engineering: A Practitioner's Approach. – 8th ed. – McGraw-Hill, 2014.
7. Глибовець М. М., Олецкий О. В. Штучний інтелект : підручник. – Київ : Видавничий дім «КМ Академія», 2020. – 366 с.
8. Рассел С., Норвіг П. Штучний інтелект: сучасний підхід. – Київ : Видавничий дім «Вільямс», 2019. – 1408 с.
9. Чаплінський Ю. П., Шекета В. І. Інформаційні системи та технології : навчальний посібник. – Івано-Франківськ : ІФНТУНГ, 2020. – 348 с.
10. Бондаренко М. Ф., Шабанов-Кушнарченко Ю. П. Основи штучного інтелекту : навчальний посібник. – Харків : ХНУРЕ, 2018. – 212 с.
11. Коваленко І. І. Програмування мовою Python : навчальний посібник. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 320 с.
12. Козуб Г., Скарга В. Автоматизація університетських процесів за допомогою розробки Telegram – бота. Abstracts of XIII International Scientific and Practical Conference. Florence, Italy. Pp. 277-279. URL: <https://eu-conf.com/events/information-and-its-impact-on-social-processes/>.
13. Kozub, H., Bakhov, I., Palamarchuk, S., Burak, V., & Lohvynenko, O. (2024). Adaptation of digital gamification in professional education amid martial law challenges. Salud, Ciencia Y Tecnología - Serie De Conferencias, 3, .1236. <https://doi.org/10.56294/sctconf2024.1236>

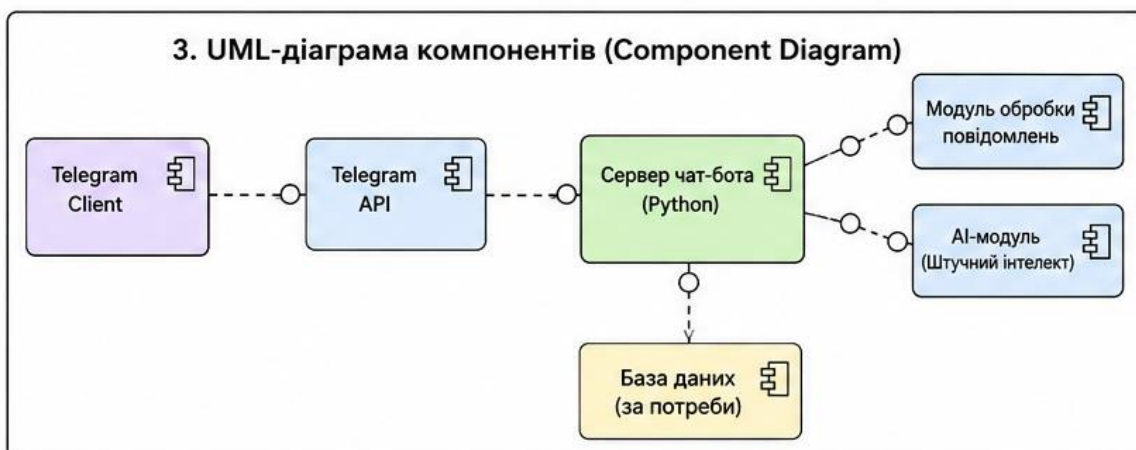
14. Козуб Г.О., Семенов М. А. Програмування : метод. рек. до лаб. робіт для студ. спец. 121 – „Інженерія програмного забезпечення”. Ста-робільськ : ДЗ „ЛНУ імені Тараса Шевченка”, 2020. 108 с.
15. Смагіна О. О., Козуб В. Ю., Козуб Г. О. Проєктування інформаційних систем: методичні вказівки до лабораторних для здобувачів вищої освіти першого бакалаврського рівня спеціальності F3 – «Комп'ютерні науки». Лубни: ДЗ „ЛНУ імені Тараса Шевченка”, 2026. 32 с.
16. Смагіна О. О., Козуб В. Ю., Козуб Г. О. Технологія створення програмних продуктів: методичні вказівки до лабораторних для здобувачів вищої освіти першого бакалаврського рівня спеціальності F3 – «Комп'ютерні науки». Лубни: ДЗ „ЛНУ імені Тараса Шевченка”, 2026. 27 с.
17. Козуб Г. О., Козуб Ю. Г. Методичні рекомендації до виконання кваліфікаційної роботи за спеціальністю 122 „Комп'ютерні науки” першого рівня вищої освіти. – Старобільськ : ДЗ „ЛНУ імені Тараса Шевченка”, 2021. 99 с.
18. Козуб В. Ю., Бобень І. Ю., Боярінова Ю.Є. Етичні аспекти використання штучного інтелекту в аналізі даних. Наукові перспективи No 6(34) 2024. С. 880-894. DOI: [https://doi.org/10.52058/2786-6025-2024-6\(34\)-880-893](https://doi.org/10.52058/2786-6025-2024-6(34)-880-893)
19. Козуб В.Ю. Інформаційна підтримка прийняття рішень із застосуванням технологій розподіленого штучного інтелекту. Інформаційні технології та суспільство. 4 (15) (Сер 2024), 71-79. <https://doi.org/10.32689/maup.it.2024.4.12>
20. Козуб В.Ю. Сучасні інструменти та фреймворки для створення багатоплатформних додатків. Наука і техніка сьогодні. № 2(43) 2025. С. 1239-1254. [https://doi.org/10.52058/2786-6025-2025-2\(43\)-1239-1254](https://doi.org/10.52058/2786-6025-2025-2(43)-1239-1254)

Додаток А. Приклади оформлення діаграм UML

1. UML-діаграма варіантів використання (Use Case Diagram)



3. UML-діаграма компонентів (Component Diagram)

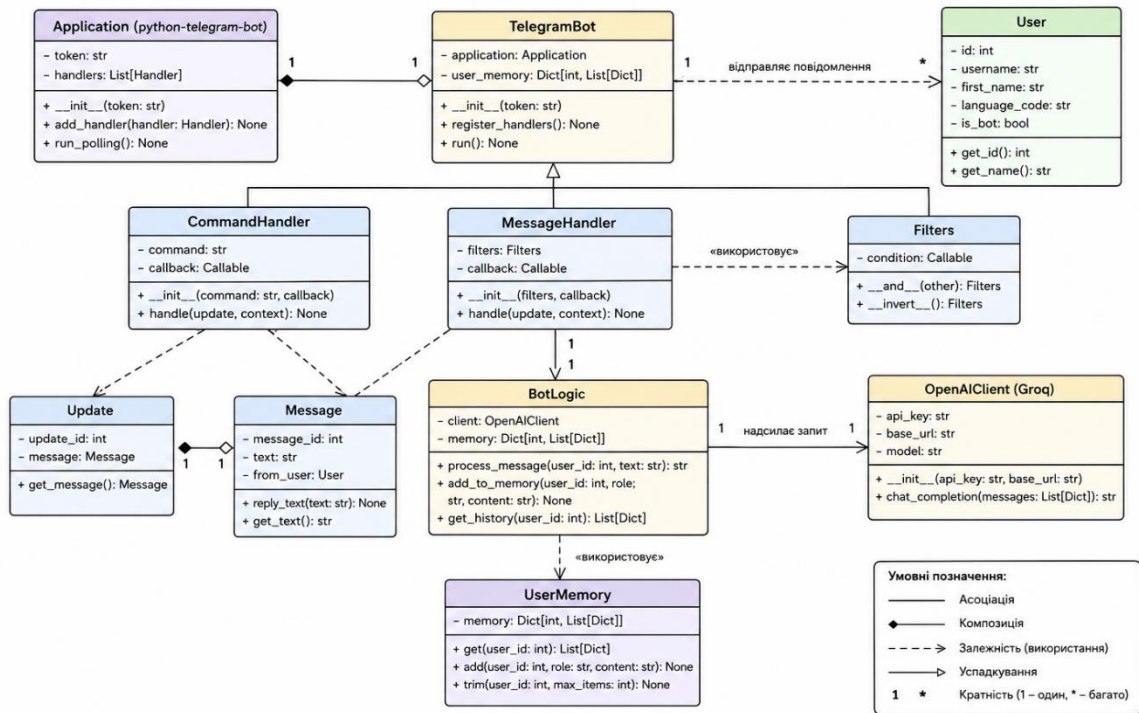


2. UML-діаграма діяльності (Activity Diagram)



UML-діаграма класів системи чат-бота "Morningstar"

Інтеграція Telegram Bot API та Groq (OpenAI-сумісний API)



Додаток Б

```

from telegram import Update

from telegram.ext import ApplicationBuilder, MessageHandler, CommandHandler, filters,
ContextTypes

from openai import OpenAI

TELEGRAM_TOKEN = "8732217889:AAGB_3Nt7pXiVChyaahXwIBMCr9brV_wfdM"
GROQ_API_KEY = "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"

client = OpenAI(
    api_key=GROQ_API_KEY,
    base_url="https://api.groq.com/openai/v1"
)

user_memory = {}

async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_text("Я бот Morningstar!")

async def handle_message(update: Update, context: ContextTypes.DEFAULT_TYPE):
    user_id = update.message.from_user.id
    text = update.message.text

    if user_id not in user_memory:
        user_memory[user_id] = []

    user_memory[user_id].append({"role": "user", "content": text})

```

```

try:
    response = client.chat.completions.create(
        model="llama-3.1-8b-instant", # 🦙 модель Groq
        messages=user_memory[user_id]
    )

    reply = response.choices[0].message.content

    user_memory[user_id].append({"role": "assistant", "content": reply})
    user_memory[user_id] = user_memory[user_id][-10:]

    await update.message.reply_text(reply)

except Exception as e:
    print(e)
    await update.message.reply_text("Помилка 🤖")

def main():
    app = ApplicationBuilder().token(TELEGRAM_TOKEN).build()

    app.add_handler(CommandHandler("start", start))
    app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, handle_message))

    print("Morningstar готовий до праці ")
    app.run_polling()

if __name__ == "__main__":
    main()

```