**МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ**
**ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ**
**КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ**

# КВАЛІФІКАЦІЙНА РОБОТА

**на тему: РОЗРОБКА ПЕРСОНАЛЬНОГО ГОЛОСОВОГО АСИСТЕНТА ДЛЯ ЕФЕКТИВНОЇ ІНТЕРАКЦІЇ З КОРИСТУВАЧЕМ**

Здобувача вищої освіти другого (магістерського) рівня,
ІІ курсу, групи КМІ-23-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»

_____ Усачик Денис Костянтинович
(підпис)

Науковий керівник _____ Редич Олександр Володимирович, к.е.н., доцент
(підпис)

Керівник освітньо-професійної програми _____ Горбовий Артур Юліанович, д.т.н, професор
(підпис)

Допущено до захисту за рішенням кафедри протокол від «10» січня 2025 №11

Завідувач кафедри _____ Філоненко Михайло Миколайович, к.ф.-м.н., доцент
(підпис)

Ірпінь – 2025

Реєстрація № 3 27.01.2025 _____
(дата) (підпис відповідальної особи)

Результати перевірки до захисту 7.01.25 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 5 (відмінно) 04.02.2025 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ В.М. Франчук
(підпис)

Члени комісії:

_____ А.Ю. Горбовий
(підпис)

_____ М.М. Філоненко
(підпис)

_____ І.Д. Погорєловська
(підпис)

# БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

| Об'єкти контролю | Діапазон оцінки в балах | Фактична кількість балів | Примітки | Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря, (підписи) |
|---|---|---|---|---|
| **ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ** | | | | |
| 1. Зміст кваліфікаційної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій) | від 0 до 30 | 27 | | 04.02.2025<br>Франчук В.М.<br>Горбовий А.Ю.<br>Філоненко М.М.<br>Погорєловська І.Д.<br>Суховецька Л.П. |
| 2. Дотримання вимог щодо оформлення | від 0 до 10 | 9 | | |
| 3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни | від 0 до 5 | 5 | | |
| 4. Рецензент | від 0 до 5 | 5 | | |
| РАЗОМ БАЛІВ | від 0 до 50 | 46 | | |

**Оцінювання кваліфікаційної роботи науковим керівником - 47 балів.**

**ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС**

| Оцінка за 100-бальною шкалою | Оцінка за шкалоюЄКТС | Оцінка за національною шкалою | Дата | П.І.Б.: голови, членів комісії, секретаря, (підпис) |
|---|---|---|---|---|
| 93 | A | 5 (відмінно) | 04.02.2025 | Франчук В.М.<br>Горбовий А.Ю.<br>Філоненко М.М.<br>Погорєловська І.Д.<br>Суховецька Л.П. |

**ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС**

| Оцінка за 100-бальною шкалою | Оцінка за шкалою ЄКТС | Оцінка за національною шкалою |
|---|---|---|
| 90-100 | A | 5 (відмінно) |
| 80-89 | B | 4 (добре) |
| 70-79 | C | |
| 60-69 | D | 3 (задовільно) |
| 50-59 | E | |
| 35-49 | FX | 2 (незадовільно) |

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології в економіці»

**Затверджую**
Завідувач кафедри _____ Філоненко М.М.
«23» грудня 2023

**Завдання на кваліфікаційну роботу**
**здобувачу вищої освіти**
**Усачик Денис Костянтинович**
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «***Розробка персонального голосового асистента для ефективної інтеракції з користувачем***»
   *Затверджена наказом ректора від «20» грудня 2023 № 1701*
2. Строк здачі закінченої роботи *«25» грудня 2024*
3. Мета роботи: *розробка персонального голосового асистента на мові програмування Python, який забезпечить ефективну та природну інтеракцію з користувачем шляхом використання сучасних технологій штучного інтелекту.*
   Об'єкт дослідження: *методологія та технології розпізнавання мови, принципи взаємодії користувача із інтерфейсами на основі штучного інтелекту.*
   Предмет дослідження: *інструментарій розробки персональних голосових асистентів на мові Python з використанням спеціалізованих бібліотек.*
   Орієнтовна база дослідження: *сучасні методи та засоби розпізнавання мови, інструментарій розробки функціональних мобільних асистентів, що використовують алгоритми штучного інтелекту для спілкування із користувачами.*
4. Вхідні дані до роботи: *мовні моделі штучного інтелекту, спеціалізовані бібліотеки мови Python, існуючі рішення мобільних асистентів.*
5. Перелік графічного матеріалу :*рисунки інтерфейсів додатку, структурні схеми, алгоритми, лістинги програм.*
6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)
   *РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ АСПЕКТИ СТВОРЕННЯ ГОЛОСОВОГО АСИСТЕНТА*
   *1.1.Сутність голосових асистентів та методів розпізнавання мови*
   *1.2. Аналіз існуючих рішень реалізації голосових асистентів*
   *1.3.Огляд інформаційних технологій та інструментів для розробки голосових асистентів*
   *РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ГОЛОСОВОГО АСИСТЕНТА*
   *2.2. Розробка голосового асистента у середовищі розробки*
   *2.3. Тестування голосового асистента та оцінка ефективності голосового асистента*

*РОЗДІЛ 3. ПЕРСПЕКТИВИ РОЗВИТКУ ГОЛОСОВОГО АСИСТЕНТА*
*3.1. Порівняння розробленого голосового асистента з існуючими рішеннями*
*3.2.Використання голосового асистента в реальних умовах, потенційні користувачі*
*3.3.Розрахунки комерційної вигоди від розробки голосового асистента*

7. Календарний план виконання роботи

| № п/п | Назва етапів кваліфікаційноїроботи | Строк виконання етапів роботи | |
|---|---|---|---|
| | | За планом | Фактично |
| 1 | 2 | 3 | 4 |
| 1 | Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи | До 15.12.2023 | «13» 12 2023 р. |
| 2 | **І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень) | 30.05.2024 | «27» 05 2024 р. |
| 3 | **ІІ розділ** (аналіз експериментальної частини про методики досліджень) | 09.09.2024 | «05» 09 2024 р. |
| 4 | **ІІІ розділ** (проведені теоретичні і (або) експериментальні дослідження та проектні рішення) | 22.11.2024 | «20» 11 2024 р. |
| 5 | **Висновки** | 16.12.2024 | «16» 12 2024 р. |
| 6 | Доопрацювання роботи з урахуванням зауважень наукового керівника | 26.12.2024 | «26» 12 2024 р. |
| 7 | Отримання відгуку керівника | 25.12.2024 | «25» 12 2024 р. |
| 8 | Отримання зовнішньої рецензії | 30.01.2025 | «30» 01 2025 р. |

8. Дата видачі завдання «23» грудня 2023
9. Керівник кваліфікаційної роботи _____ (підпис) Редич О.В. -
10. Керівник освітньо-професійної програми _____ (підпис) Горбовий А.Ю.
11. Завдання до виконання отримав здобувач вищої освіти _____ (підпис) Усачик Д.К.

# АНОТАЦІЯ

## «Розробка персонального голосового асистента для ефективної інтеракції з користувачем»

Метою роботи є розробка персонального голосового асистента на мові програмування Python, який забезпечить ефективну та природну інтеракцію з користувачем шляхом використання сучасних технологій штучного інтелекту.

Завдання роботи є провести дослідження технологій створення голосових асистентів та створити персонального голосового асистента, який буде ефективно інтерактувати з користувачами.

Предметом дослідження методи та засоби розробки персонального голосового асистента на мові Python із застосуванням технологій штучного інтелекту для забезпечення ефективної та природної інтеракції з користувачем.

Об’єктом дослідження процес взаємодії користувача з голосовим асистентом.

Робота має 3 розділи: теоретико-методологічні аспекти створення голосових асистентів, проектування та розробка голосового асистента, перспективи розвитку голосового асистента.

В першому розділі проводиться огляд та аналіз предметної сфери та обґрунтування вибору засобів для створення голосового асистента.

В другому розділі проводиться розробка алгоритму голосового асистента з описом його функціонування, а також саме кодування та тестування.

У третьому розділі проводиться порівняння голосового асистента з існуючими рішеннями, а також рекомендація для подальшого розвитку.

Кваліфікаційна робота містить 56 сторінок, 1 таблицю, 4 рисунка, список використаних джерел 43 найменувань, 3 додатки

Ключові слова: Голосовий асистент, Python, OpenAI, синтез голосу, розпізнавання голосу.

Рік виконання кваліфікаційної роботи: 2025

Рік захисту роботи: 2025

# ANNOTATION

**«Development of a Personal Voice Assistant for Effective User Interaction»**

The aim of the work is to develop a personal voice assistant using the Python programming language, which ensures effective and natural interaction with the user through the application of modern artificial intelligence technologies.

The objectives of the work include conducting research on technologies for creating voice assistants and developing a personal voice assistant that can interact effectively with users.

The subject of the research involves methods and tools for developing a personal voice assistant in Python, utilizing artificial intelligence technologies to ensure efficient and natural interaction with users.

The object of the research is the process of user interaction with the voice assistant.

The work consists of three sections: theoretical and methodological aspects of creating voice assistants, design and development of a voice assistant, and prospects for the development of voice assistants.

The first section provides a review and analysis of the subject area and substantiates the choice of tools for creating a voice assistant.

The second section involves the development of the voice assistant algorithm with a description of its functionality, as well as coding and testing.

The third section compares the developed voice assistant with existing solutions and provides recommendations for further development.

The qualification work comprises 56 pages, 1 table, 4 figures, a list of 43 references, and 4 appendix.

Keywords: Voice Assistant, Python, OpenAI, Speech Synthesis, Speech Recognition.

Year of Completion: 2025

Year of Defense: 2025

# ЗМІСТ

**ВСТУП**..... 3

**РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ АСПЕКТИ СТВОРЕННЯ ГОЛОСОВОГО АСИСТЕНТА**..... 6

**1.1. Сутність голосових асистентів та методів розпізнавання мови** .....6

**1.2. Аналіз існуючих рішень.** ..... 13

**1.3. Огляд інформаційних технологій та інструментів для розробки голосового асистента** .....18

**РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ГОЛОСОВОГО АСИСТЕНТА**..... 25

**2.1. Проєктування архітектури та алгоритму роботи голосового асистента .**.....25

**2.2. Розробка голосового асистента у середовищі розробки** .....30

**2.3. Тестування голосового асистента та оцінка ефективності голосового асистента** .....45

**РОЗДІЛ 3. ПЕРСПЕКТИВИ РОЗВИТКУ ГОЛОСОВОГО АСИСТЕНТА**..... 50

**3.1. Порівняння розробленого голосового асистента з існуючими рішеннями** .....50

**3.2. Використання голосового асистента в реальних умовах, потенційні користувачі** .....52

**3.3. Розрахунки комерційної вигоди від розробки голосового асистента** .53

**ВИСНОВКИ**..... 57

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**..... 59

**ДОДАТКИ**. ..... 64

**Додаток А** ..... 64

**Додаток Б** ..... 65

**Додаток В** .....67

# ВСТУП

У сучасному світі інформаційних технологій голосові асистенти стали невід'ємною частиною повсякденного життя. Вони допомагають користувачам виконувати різноманітні завдання, такі як пошук інформації та керування пристроями до організації особистого часу. Розвиток штучного інтелекту та машинного навчання сприяє створенню більш інтелектуальних та адаптивних асистентів, які здатні розуміти та реагувати на потреби користувача.

Проте, незважаючи на значний прогрес у цій галузі, багато існуючих голосових асистентів мають обмежені можливості персоналізації та не завжди забезпечують ефективну взаємодію з користувачем. Це створює потребу в розробці нових рішень, які б враховували індивідуальні особливості та вподобання користувача, підвищуючи тим самим якість взаємодії.

**Актуальність теми** обумовлена зростаючим попитом на інтелектуальні системи, здатні забезпечити більш індивідуальний підхід до користувача. Персональні голосові асистенти можуть значно підвищити ефективність виконання повсякденних завдань та покращити користувацький досвід.

**Метою** даної роботи є розробка персонального голосового асистента на мові програмування Python, який забезпечить ефективну та природну інтеракцію з користувачем шляхом використання сучасних технологій штучного інтелекту.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- Проаналізувати існуючі голосові асистенти та технології, що використовуються для їх реалізації.

- Розробити архітектуру персонального голосового асистента з урахуванням вимог ефективної взаємодії.

- Реалізувати асистента на Python, інтегруючи модулі розпізнавання мови, обробки природної мови та синтезу мовлення.

- Провести тестування асистента та оцінити його ефективність у взаємодії з користувачем.

- Розробити рекомендації щодо подальшого вдосконалення асистента.

**Стан проблеми.** Голосові асистенти, такі як Siri, Google Assistant, Amazon Alexa та Microsoft Cortana, стали важливою частиною сучасного життя, дозволяючи користувачам взаємодіяти з пристроями за допомогою природної мови. Вони допомагають виконувати різноманітні завдання — від пошуку інформації до керування розумними пристроями в домі.

Проте існуючі асистенти мають ряд обмежень такі як: обмежена персоналізація, стандартні налаштування не завжди відповідають індивідуальним потребам користувача; недостатня підтримка менш розповсюджених мов та діалектів; голосові асистенти можуть не враховувати емоційний стан або специфіку запиту користувача; взаємодія з сторонніми додатками та пристроєм може бути обмеженою.

Для вирішення цих проблем розробники зосереджуються на створенні адаптивних алгоритмів, що враховують особливості користувача; використанні сучасних методів NLP для кращого розуміння запитів; розробці відкритих рішень, доступних на різних платформах.

У цьому контексті виникає потреба в розробці персонального голосового асистента, який поєднає переваги сучасних технологій та врахує індивідуальні потреби користувача. Мова Python є оптимальним вибором для такої розробки завдяки своїй гнучкості, широкому набору бібліотек та активній спільноті

**Об'єктом дослідження** є процес взаємодії користувача з голосовим асистентом.

**Предметом дослідження** є методи та засоби розробки персонального голосового асистента на мові Python із застосуванням технологій штучного інтелекту для забезпечення ефективної та природної інтеракції з користувачем.

**Методи дослідження** включають аналіз літературних джерел, дослідження існуючих рішень, системне проєктування, програмування,

експериментальне тестування та аналіз результатів, а також методи машинного навчання.

**Практична значущість отриманих результатів** роботи полягає в тому, що розроблений асистент має можливість працювати з глибинними нейронними мережами. Та може бути використаний як основа для подальших досліджень та розробок у сфері голосових технологій та штучного інтелекту. Серед переваг розробленого асистента у порівнянні з аналогами, є краща підтримка української мови, можливість змінювати його функціонал, а також проста взаємодія з ним.

**Особистий внесок автора.** В рамках даного проєкту автор самостійно здійснив усі етапи розробки голосового асистента: від аналізу вимог, проєктування архітектури та вибору технологій до написання коду, інтеграції сторонніх API, тестування та налагодження. Автор розробив свій механізм, що підтримує основні голосові команди, діалоговий режим, адаптацію під українську мову та продемонстрував його роботу у реальних умовах.

**Наукова новизна проєкту** роботи полягає в розробці підходу до створення персонального голосового асистента, який адаптується до потреб користувача та забезпечує ефективну інтеракцію завдяки використанню передових технологій обробки природної мови та використанні бібліотеки difflib для реалізації роботи алгоритму Левенштейна.

**Структура роботи.** Дипломна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі проаналізовано сучасний стан та тенденції розвитку голосових асистентів. У другому розділі розглянуто методи та інструменти, необхідні для розробки асистента. Третій розділ присвячений реалізації програмного забезпечення. У четвертому розділі описано результати тестування та проведено аналіз ефективності розробленого рішення.

# РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ АСПЕКТИ СТВОРЕННЯ ГОЛОСОВОГО АСИСТЕНТА

## 1.1. Сутність голосових асистентів та методів розпізнавання мови

Голосовий асистент — це програмне забезпечення, який використовує технології розпізнавання та синтезу мовлення для забезпечення взаємодії між людиною і комп'ютером за допомогою природної людської мови. Він може відповідати на запити, виконувати команди, надавати інформацію та ініціювати дії на основі контексту або попереднього досвіду взаємодії з користувачем.

Основною метою голосового асистента є спрощення доступу до інформації та функцій пристроїв, роблячи взаємодію більш інтуїтивною та природною. Це досягається завдяки використанню технологій розпізнавання мови (Automatic Speech Recognition), обробки природної мови (Neuro-linguistic programming) та синтезу мовлення (Text-to-Speech).

Голосовий асистент складається з кількох основних компонентів, які працюють у тісній взаємодії:

***Модуль розпізнавання мови (ASR)*** — приймає аудіосигнал від користувача та перетворює його в текст за допомогою акустичних та мовних моделей. Забезпечує підтримку різних мов та акцентів.

***Модуль обробки природної мови (NLP)*** — аналізує отриманий текст для визначення намірів та сутностей. Використовує синтаксичний та семантичний аналіз для розуміння контексту і може враховувати емоційний стан користувача.

***Модуль синтезу мовлення (TTS)*** — Перетворює текстову відповідь у природне мовлення, налаштовуючи голос, інтонацію та швидкість для забезпечення зрозумілості та природності.

При розпізнаванні мови, часто використовується алгоритм Левенштейна. «У теорії інформації і комп'ютерній лінгвістиці міра відмінності двох послідовностей символів (рядків). Обчислюється як мінімальна кількість

операцій вставки, видалення і заміни, необхідних для перетворення одної послідовності в іншу» [27]. Варто розглянути його математичний розрахунок. «Для розрахунку відстані Левенштейна найчастіше застосовують простий алгоритм, в якому використовується матриця розміром (n + 1) * (m + 1), де n і m - довжини порівнюваних рядків. Окрім цього вартість операцій вилучення, заміни та вставки вважається однаковою» [27]. Для конструювання матриці використовують таке рекурсивне рівняння (1.1).

$$D_{0,0} = 0$$

$$D_{i,j} = min \begin{cases} D_{i-1,j-1} + 0 \\ D_{i-1,j-1} + 1 \\ D_{i-1,j} + 1 \\ D_{i,j-1} + 1 \end{cases} \quad (1.1)$$

Де, $D_{0,0}$ початкове значення матриці, коли обидва рядки порожні, дорівнює нулю.

$D_{i,j}$ це елемент матриці, що позначає мінімальну кількість операцій для перетворення перших $i$ символів одного рядка в перші $j$ символів іншого рядка.

$D_{i-1,j-1} + 0$ якщо символи на позиціях $i$ та $j$ однакові, жодних додаткових операцій не потрібно.

$D_{i-1,j-1} + 0$ якщо символи різні, потрібно виконати заміну одного символу на інший.

$D_{i-1,j} + 1$ якщо символ доданий у другий рядок.

$D_{i,j-1} + 1$ якщо символ видалений з першого рядка.

Відстань Левенштейна має такі межі:

- Мінімальне значення дорівнює різниці в довжині порівнюваних рядків.
- Максимальне значення не перевищує довжини найдовшого з рядків.
- Відстань дорівнює нулю лише тоді, коли обидва рядки повністю ідентичні (символи на кожній позиції збігаються).

Серед основних методів розпізнавання мови можна виділити наступні:

***Акустичне моделювання*** — використовуються для перетворення звукових сигналів у послідовність фонем. При цьому використовуються глибокі нейронні мережі для моделювання зв’язку між акустичними сигналами та фонемами; рекурентні нейронні мережі для роботи із послідовними даними, як-от аудіо; transformer-архітектури використовують самонавчання для розпізнавання мовлення.

***Моделювання мовлення*** — визначають, яке слово або послідовність слів є ймовірнішими. При цьому використовуються N-грами, де ймовірність визначається на основі частоти послідовностей слів у корпусі даних; нейронні мережі, які можуть вивчати контекстуальні залежності.

***Аналіз сигналу*** — обробка аудіо включає перетворення звукового сигналу в цифрову форму для подальшого аналізу. При цьому використовується перетворення Фур'є, для перетворення звуку у спектральне представлення; Мел-фільтровані коефіцієнти кепстра для вилучення ключових характеристик звуку; та спектрограми, як вхід до глибоких моделей.

***Глибинні нейронні мережі*** (згорткова нейромережа) — це багаторівневі структури, що складаються з вхідного, прихованих і вихідного шарів, де кожен шар складається з великої кількості нейронів. Завдяки багаторівневій архітектурі DNN здатні розпізнавати складні залежності в даних. При створенні голосових асистентів такі мережі використовують для обробки природної мови та синтезу або розпізнавання мовлення. Глибинні мережі моделюють нелінійні залежності у мовних даних, перетворюючи звукові сигнали у текст чи текст у синтезоване мовлення.

$$y = \sigma(\sum_{i=1}^{n} w_i x_i + b) \quad (1.2)$$

Де, $x_i$ це вхідні значення.

$w_i$ це ваги.

$b$ це зміщення.

$\sigma$ це активаційна функція.

$y$ це вихід нейрону.

Загальну структури згорткової нейромережі можна побачити на рисунку 1.1.

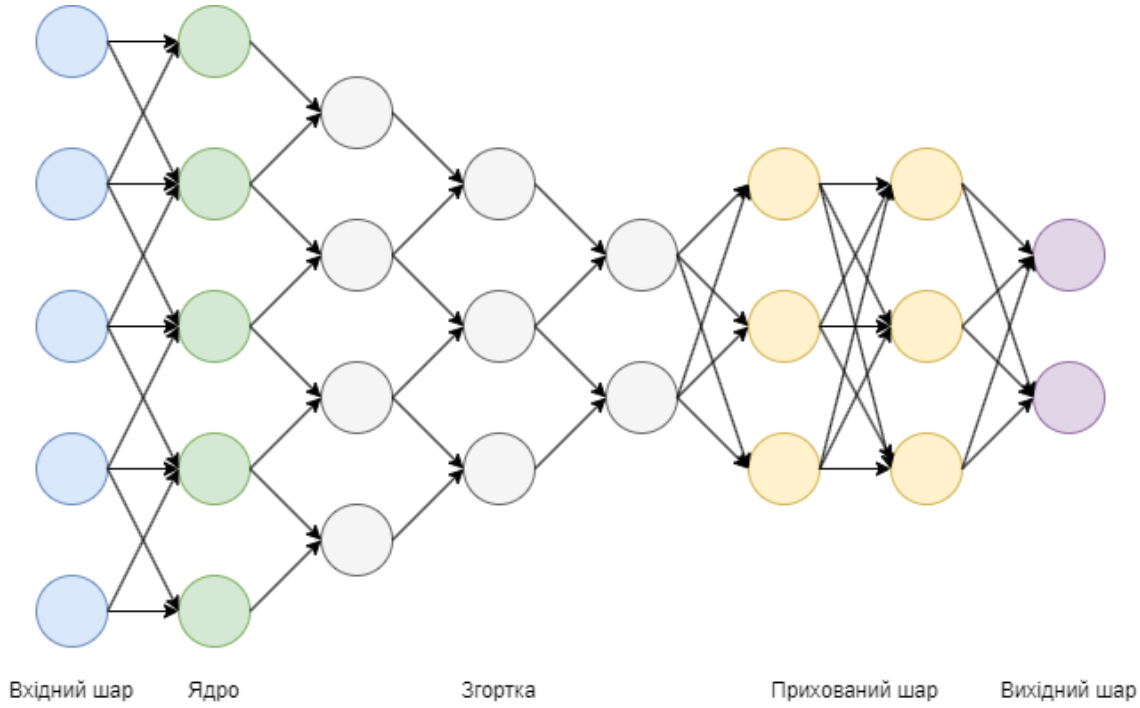



Рис. 1.1 Структура згорткової нейромережі

- Вхідний шар. Обробляє початкові дані, наприклад, зображення або сигнали. Кожен піксель або вхідна величина передається далі.
- Згортковий шар. Застосовує ядра згортки, які проходять по входу та виділяють локальні ознаки, наприклад, контури або текстури. Це дозволяє виявляти патерни незалежно від їхнього місця розташування.
- Вихідний шар. Забезпечує результат, наприклад, класи з ймовірностями або інші значення залежно від завдання.

«***Рекурентна нейронна мережа*** — це клас штучних нейронних мереж, у яких з'єднання між вузлами утворюють орієнтований у часі граф. Це створює внутрішній стан мережі, що дозволяє їй проявляти динамічну поведінку в часі. На відміну від нейронних мереж прямого поширення, РНМ можуть

використовувати свою внутрішню пам'ять для обробки довільних послідовностей входів. Це робить їх застосовними до таких задач, як розпізнавання несегментованого неперервного рукописного тексту та розпізнавання мовлення» [28].

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \quad (1.3)$$

Де, $h_t$ це поточний стан прихованого шару.

$h_{t-1}$ це попередній стан прихованого шару.

$x_t$ це вхідний вектор у моменті часу t.

$W_{hh}$ це матриця ваг для зв'язків між попереднім і поточним станами прихованого шару.

$W_{xh}$ це матриця ваг для зв'язків між вхідним вектором і прихованим шаром.

$b_h$ це вектор зміщення.

$\sigma$ це функція активації.

Рекурентні нейронні мережі мають ключове значення у роботі голосових асистентів. Вони використовуються для обробки природної мови, що дає змогу асистентам розпізнавати та створювати текст або мовлення, зрозуміле людині. Зокрема, РНМ допомагають у завданнях розпізнавання голосу, аналізу контекстуальних даних та формування відповідей, що сприяє комфортному та ефективному спілкуванню з користувачем.

***Перетворення Фур'є*** — це математичний інструмент, який дозволяє представити часову функцію у вигляді сукупності синусоїдальних компонентів різних частот. Іншими словами, воно перетворює сигнал з часової області в частотну, що дає змогу аналізувати його частотний склад. Це особливо корисно для обробки сигналів, таких як звукові хвилі або електричні коливання.

Математично перетворення Фур'є функції визначається так (1.4)

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \quad (1.4)$$

Де, F(ω) це частотне представлення функції.

$\omega$ це кутова частота.

$i$ це уявна одиниця.

У контексті голосових асистентів перетворення Фур'є відіграє важливу роль у розпізнаванні та обробці мовлення. Звукові сигнали, що містять мовлення, можуть бути складними та містити широкий спектр частот. Застосовуючи перетворення Фур'є, ці сигнали можна розкласти на окремі частотні компоненти, що спрощує їх аналіз та обробку. Це дозволяє системам розпізнавання мовлення ефективніше ідентифікувати фонеми, слова та інші мовні елементи, забезпечуючи точну та швидку відповідь голосового асистента.

***Кепстральні коефіцієнти*** — це набір числових значень, які характеризують спектральні властивості сигналу, зокрема його частотний вміст. Вони отримуються шляхом обчислення оберненого перетворення Фур'є від логарифма амплітудного спектра сигналу. Цей підхід дозволяє ефективно відокремити інформацію про джерело звуку від характеристик середовища його поширення.

У сфері голосових асистентів кепстральні коефіцієнти, особливо мел-частотні кепстральні коефіцієнти (MFCC), широко використовуються для розпізнавання мовлення. MFCC відображають, як людське вухо сприймає різні частоти, що робить їх надзвичайно корисними для ідентифікації та аналізу мовних сигналів. Застосування MFCC дозволяє системам голосових асистентів точно розпізнавати слова та фрази, адаптуючись до різних голосів і акцентів користувачів.

Голосові асистенти знаходять застосування в різних сферах, наприклад, керування розумним будинком, встановлення будильників, нагадувань, швидкий доступ до функцій телефону, відправлення повідомлень на телефоні, навігація в автомобілі, медіа та зв'язком під час водіння, організація бізнес-

зустрічей, управління календарем, пошук інформації в Інтернеті, допомога у навчанні.

Зростання популярності голосових асистентів обумовлено їх здатністю робити технології більш доступними та зручними. Вони підвищують ефективність, дозволяючи швидше виконувати завдання без ручного введення, покращують доступність для людей з обмеженими можливостями та забезпечують персоналізацію, адаптуючись до звичок та вподобань користувача.

Крім того, голосові асистенти сприяють розвитку нових ринків та відкривають можливості для інновацій у різних галузях, інтегруючи технології в повсякденне життя.

Попри успіхи, голосові асистенти стикаються з низкою викликів:

- Забезпечення захисту особистих даних користувачів є критично важливим завданням.
- Потреба в покращені здатності асистентів розуміти складні запити та контекст взаємодії.
- Розширення підтримки різних мов та діалектів для глобальної доступності.
- Вирішення питань, пов'язаних із впливом на суспільство та можливим заміщенням людської праці.

Перспективи розвитку включають інтеграцію з іншими формами штучного інтелекту, розширення функціональності та покращення взаємодії з користувачем на більш інтуїтивному рівні.

Сутність голосового асистента полягає в забезпеченні природної та ефективної взаємодії між людиною та комп'ютером за допомогою голосу. Використовуючи сучасні технології, голосові асистенти стають все більш розумними та адаптивними, відкриваючи нові можливості для покращення якості життя та підвищення продуктивності в різних сферах діяльності.

## 1.2. Аналіз існуючих рішень

У цьому аналізі варто розглянути два основних види існуючих голосових асистентів, це комерційні голосові асистенти та голосові асистенти з відкритим вихідним кодом. Розглянемо їх переваги, недоліки та вплив на користувацький досвід.

Почнемо з комерційних голосових асистентів:

1. ***Apple Siri*** — «це персональний помічник і питально-відповідальна система, адаптована під iOS» [42], розроблений компанією Apple, інтегрований у пристрої екосистеми Apple, включаючи iPhone, iPad, Mac, Apple Watch, HomePod та інші. Siri забезпечує голосове управління пристроями, пошук інформації, автоматизацію задач і взаємодію з додатками. Підтримує понад 20 мов, включаючи українську.

***Переваги Apple Siri:***

- Siri працює безпосередньо з усіма продуктами Apple, забезпечуючи безшовний досвід користувача.
- Apple акцентує увагу на безпеці даних, обробляючи багато запитів локально на пристрої та мінімізуючи збір даних.
- Можливість створювати індивідуальні автоматизації та голосові команди для додатків.
- Інтуїтивний інтерфейс і можливість активації за допомогою команди "Hey Siri".

***Недоліки Apple Siri:***

- Siri важко інтегрувати зі сторонніми додатками та сервісами за межами екосистеми Apple.
- У порівнянні з Google Assistant, Siri інколи не справляється зі складними чи контекстуальними запитами.
- Для багатьох функцій потрібне стабільне підключення до мережі.
- Асистент малоефективний на пристроях, які не належать до екосистеми Apple.

2. ***Google Assistant*** — це «розумний персональний асистент, розроблений компанією Google і представлений на конференції Google I/O 18 травня 2016 року» [40]. Асистент здатний відповідати на запити, виконувати команди, інтегруватися з Google-сервісами та сторонніми додатками. Підтримує понад 30 мов, включаючи українську.

***Переваги Google Assistant:***

- Google Assistant ефективно розуміє контекстуальні та складні запити завдяки використанню сучасних AI-технологій.
- Працює з Gmail, Google Calendar, Google Maps та іншими сервісами для полегшення управління завданнями.
- Асистент доступний на багатьох пристроях, включаючи смартфони, смарт-колонки, телевізори та автомобілі.
- Інтеграція з великою кількістю сторонніх додатків та розумних пристроїв.

***Недоліки Google Assistant:***

- Обробка даних у хмарі може викликати занепокоєння щодо приватності користувача.
- Асистент вимагає стабільного підключення до мережі для більшості функцій.
- Хоча Google Assistant підтримує сторонні пристрої, його інтеграція найефективніша в межах Google-екосистеми.
- Через широку функціональність і можливості персоналізації, новим користувачам може бути важко освоїти всі функції.

Продовження огляду комерційних голосових асистентів продовжено в Додатку Б.

Тепер розглянемо голосові асистенти з відкритим вихідним кодом. Відкриті голосові асистенти набувають все більшої популярності як альтернатива комерційним рішенням. Вони надають можливість кастомізації, підвищують рівень конфіденційності та дозволяють спільноті розробників брати участь у їх вдосконаленні.

1. ***Mycroft*** — це відкритий голосовий асистент із відкритим вихідним кодом, який орієнтований на кастомізацію, конфіденційність та інтеграцію з різними пристроями. Він може працювати на таких платформах, як Linux, Raspberry Pi, Windows і macOS. Підтримує кілька мов, включаючи англійську; українська мова може бути додана спільнотою. Ліцензія: Apache License 2.0.

***Переваги Mycroft:***

- Mycroft можна модифікувати відповідно до індивідуальних потреб, що робить його ідеальним для розробників та ентузіастів.
- Асистент може працювати без інтернету, забезпечуючи локальну обробку даних.
- Mycroft сумісний із Linux, Windows, macOS та пристроями на базі Raspberry Pi.
- Розвиток проєкту активно підтримується спільнотою ентузіастів, які створюють нові функції.
- Підтримка інтеграції зі сторонніми сервісами та пристроями, включаючи розумний дім.

***Недоліки Mycroft:***

- У порівнянні з комерційними асистентами, набір функцій може бути обмеженим.
- Може потребувати технічних знань для встановлення та конфігурації.

2. ***Open Assistant*** — це голосовий асистент із відкритим вихідним кодом, створений для інтеграції сучасних AI-технологій та надання можливостей кастомізації. Проєкт спрямований на забезпечення прозорості, конфіденційності та адаптації під потреби користувачів і розробників. Підтримка залежить від спільноти; основна мова — англійська. Різні ліцензії для компонентів, часто MIT або Apache

***Переваги Open Assistant:***

- Користувачі та розробники можуть переглядати, модифікувати та покращувати функціональність асистента.

- Використання передових моделей машинного навчання та NLP для забезпечення природного спілкування.
- Дає можливість вибору компонентів для реалізації окремих функцій (ASR, NLP, TTS).
- Забезпечує конфіденційність і дозволяє уникнути передачі даних у хмару.
- Постійний розвиток проєкту завдяки внескам ентузіастів із відкритої спільноти.

***Недоліки Open Assistant:***

- Проєкт може бути нестабільним, а функціональність - обмеженою.
- Налаштування та інтеграція асистента можуть бути складними для новачків.
- Підтримка мов залежить від спільноти; українська мова може бути не доступною без додаткової роботи.
- Для використання сучасних AI-моделей потрібні значні обчислювальні ресурси.
- Немає гарантії технічної допомоги, що може ускладнити вирішення проблем.

Продовження огляду голосових асистентів з відкритим вихідним кодом продовжено в Додатку В.

Розглянувши деякі основні приклади готових рішень голосових асистентів можна сформувати порівняльну таблицю між комерційними голосовими асистентами та відкритими голосовими асистентами (див. табл. 1).

Таблиця 1

Порівняння голосових асистентів

| **Критерії** | **Комерційні голосові асистенти** | **Відкриті голосові асистенти** |
|---|---|---|
| **Доступність** | Доступні на популярних платформах та пристроях | Потребують налаштування та інтеграції |
| **Функціональність** | Широкий набір готових функцій | Обмежений базовий набір функцій |

| | | |
|---|---|---|
| **Мовна підтримка** | Більшість підтримують понад 20 мов (переважно присутня українська) | Обмежена підтримка мов; додавання нових мов потребує зусиль |
| **Кастомізація** | Мінімальна, обмежена екосистемою розробника | Максимальна, можливість змінювати функціональність |
| **Конфіденційність** | Дані обробляються в хмарі; залежність від політики конфіденційності розробника | Локальна обробка даних, повний контроль над інформацією |
| **Інтеграція** | Глибока інтеграція в екосистему розробника | Можливість інтеграції з різними платформами |
| **Вартість** | Безкоштовні для користувачів, але монетизуються через екосистему | Безкоштовні, потребують лише обладнання |
| **Продуктивність** | Висока, завдяки потужним хмарним ресурсам | Залежить від локального обладнання |
| **Гнучкість** | Обмежена екосистемою розробника | Повна свобода у використанні |
| **Підтримка** | Доступна технічна підтримка | Відсутня офіційна підтримка, залежність від спільноти |

Комерційні голосові асистенти підходять для користувачів, які цінують готову функціональність, простоту використання та доступність на популярних платформах. Проте питання конфіденційності та обмеження кастомізації можуть стати недоліком.

Відкриті голосові асистенти ідеальні для ентузіастів та розробників, які шукають конфіденційність, гнучкість і можливість адаптації асистента під свої потреби. Їх використання вимагає більше зусиль і технічної підготовки.

Кінцевий вибір залежить від потреб користувача: чи потрібна проста у використанні система з широкими можливостями, чи гнучкий, конфіденційний інструмент із можливістю налаштування.

## 1.3. Огляд інформаційних технологій та інструментів для розробки голосового асистента

Розробка персонального голосового асистента для ефективної інтеракції з користувачем потребує використання сучасних інформаційних технологій та інструментів. У цьому підрозділі розглядаються основні засоби, які були застосовані при створенні асистента: середовище розробки Visual Studio Code, система контролю версій Git та платформа GitHub, мова програмування Python та бібліотеки pyttsx3, speech_recognition і openai.

***Visual Studio Code*** — це безкоштовне кросплатформне середовище розробки від Microsoft, яке підтримує широкий спектр мов програмування, включаючи Python. Visual Studio Code надає розробнику інструменти для зручного написання, відлагодження та тестування коду. Завдяки розширенням та інтеграції з іншими сервісами, Visual Studio Code стає потужним інструментом для розробки голосових асистентів.

***Git*** — це розподілена система контролю версій, яка дозволяє відстежувати зміни в коді та співпрацювати з іншими розробниками. GitHub — це веб-платформа для хостингу Git-репозиторіїв, яка надає додаткові можливості для управління проектами та спільної роботи. Використання Git та GitHub забезпечує надійне збереження коду, історію змін та спрощує процес командної розробки.

***Python*** — це високорівнева, інтерпретована мова програмування загального призначення, створена Гвідо ван Россумом і вперше випущена у 1991 році. Вона відома своєю простою та читабельною синтаксичною структурою, яка дозволяє розробникам швидко писати зрозумілий код. Python підтримує кілька парадигм програмування, включаючи об'єктно-орієнтоване, процедурне та функціональне програмування.

Перевага Python в простоті та читабельності коду. Синтаксис Python нагадує природну мову, що робить код більш зрозумілим і легким для підтримки. Використання відступів для визначення блоків коду сприяє створенню структурованих та читабельних програм.

Python постачається з обширною стандартною бібліотекою, яка охоплює широкий спектр функціональностей, від роботи з файлами та мережевими протоколами до управління базами даних та веб-розробки. Існує безліч сторонніх бібліотек і фреймворків, таких як NumPy, pandas, TensorFlow та інші, які розширюють можливості Python у різних галузях, включаючи науку про дані, машинне навчання та штучний інтелект.

Python працює на різних операційних системах, таких як Windows, macOS, Linux та інших, що забезпечує гнучкість у розробці та розгортанні додатків на різних платформах без змін у коді.

Обґрунтувати використання Python у проєкті можна тим, що він підтримує бібліотеки для обробки мовлення та штучного інтелекту. Python має потужні інструменти для обробки природної мови (NLP), що може бути корисним для вдосконалення взаємодії між користувачем та асистентом. Python легко інтегрується з такими середовищами розробки, як Visual Studio Code, та системами контролю версій, як Git та GitHub, що спрощує процес розробки та співпраці. Велика кількість доступних ресурсів, документації та активна спільнота розробників полегшують вирішення проблем та обмін знаннями, що є цінним для успішної реалізації проєкту.

Перейдемо до розгляду бібліотек, які необхідні для реалізації голосового асистента.

***Бібліотека pyttsx3*** — це кросплатформенна бібліотека для синтезу мовлення в Python, яка дозволяє перетворювати текст у голос. Вона працює в офлайн-режимі, не вимагає інтернет-з'єднання та підтримує різні операційні системи. На відміну від деяких інших бібліотек, pyttsx3 використовує системні механізми синтезу мовлення, такі як SAPI5 на Windows, NSSpeechSynthesizer на macOS та eSpeak на Linux.

Бібліотека pyttsx3 взаємодіє з системними API для синтезу мовлення, надаючи розробнику простий інтерфейс для озвучення тексту. Основний об'єкт бібліотеки — Engine, який керує процесом синтезу. Через цей об'єкт можна:

- Додавати текст для озвучення.

- Налаштовувати параметри голосу (швидкість, гучність, тембр).
- Керувати чергою озвучення (додавати, видаляти або змінювати повідомлення).

Ось приклад коду для озвучення тексту (див. лістінг 1).

Лістинг 1 — Приклад озвучення тексту

```
import pyttsx3
engine = pyttsx3.init()
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[6].id)
engine.say("Привіт! Я ваш голосовий асистент.")
engine.runAndWait()
```

Цей код озвучує текст «Привіт! Я ваш голосовий асистент.» До того ж є можливість вибору голосу з встановлених пакетів голосів. Існує декілька пакетів українських голосів, як жіночих так і чоловічих, це дуже полегшить і покращить розробку голосового асистента.

У контексті розробки персонального голосового асистента pyttsx3 виконує ключову функцію синтезу мовлення, забезпечуючи природну та інтерактивну взаємодію з користувачем. Основні переваги використання pyttsx3 у проєкті:

- Не потребує інтернет-з'єднання, що підвищує швидкість реакції та забезпечує конфіденційність даних.
- Підтримує основні операційні системи без необхідності змін у коді.
- Дозволяє налаштовувати параметри голосу під потреби користувача або специфіку додатку.
- Легко інтегрується з іншими бібліотеками та модулями Python, такими як speech_recognition для розпізнавання мовлення.

***Бібліотека speech_recognition*** — це популярна Python-бібліотека, призначена для розпізнавання мовлення за допомогою різних постачальників (API) та сервісів. Вона надає уніфікований інтерфейс для взаємодії з декількома механізмами розпізнавання мовлення, включаючи онлайн-сервіси Google

Speech Recognition, IBM Watson, Microsoft Azure, а також офлайн-рушії (наприклад, CMU Sphinx).

Основна ідея бібліотеки полягає в тому, що розробник може працювати з єдиним, зручним у використанні інтерфейсом, незалежно від того, який сервіс чи рушій для розпізнавання використовується "під капотом". Це спрощує процес інтеграції розпізнавання мовлення у додатки, дозволяє легко змінювати постачальника послуг та робити застосунок гнучкішим.

Принцип роботи та базова логіка використання speech_recognition зазвичай включає такі етапи:

- Створення екземпляра розпізнавача (Recognizer). Об’єкт класу Recognizer відповідає за керування процесом розпізнавання мовлення. Він містить методи для завантаження аудіо-даних, обробки та передачі їх у вибраний рушій розпізнавання.
- Отримання аудіо-даних. Аудіо може бути отримане з різних джерел, наприклад, з мікрофону з використанням класу Microphone можна записати голос у режимі реального часу. Або з аудіо-файлу, бібліотека дозволяє завантажувати та обробляти файли у форматах WAV, AIFF, AIFF-C та ін.
- Обробка аудіо-даних. Після отримання аудіо створюється об’єкт типу AudioData, який містить PCM-дані звуку. Recognizer викликає певний метод recognize_*, який передає ці дані відповідному сервісу чи рушію розпізнавання, повертаючи текстовий результат.
- Налаштування параметрів. Деякі методи розпізнавання дозволяють налаштовувати мову розпізнавання, налаштування довжини паузи, максимальний час очікування відповіді, використання ключів API тощо.

В цій бібліотеці можна визначити наступні ключові класи та методи

Recognizer — основний клас для розпізнавання мовлення.

- listen(source): записує аудіо з джерела у форматі AudioData.

- recognize_google(audio_data, key=None, language="..."): використовує Google Speech Recognition для перетворення аудіо у текст.
- recognize_sphinx(audio_data, language="..."): використовує офлайн-рушій CMU Sphinx.
- Крім Google та Sphinx, підтримуються інші методи: recognize_ibm(), recognize_houndify(), recognize_wit(), recognize_bing(), recognize_azure(), залежно від наявності ключів API та конфігурації.

Microphone — клас, що дозволяє записати аудіо з мікрофона.

- Microphone(device_index=None): ініціалізує мікрофон. Параметр device_index може використовуватися для вибору конкретного пристрою, якщо їх декілька.

AudioFile — клас для роботи з аудіо-файлами.

- Використання контекстного менеджера with AudioFile('path_to_file.wav') as source: дозволяє зчитати дані з файлу та підготувати їх для розпізнавання.

У контексті розробки голосового асистента бібліотека speech_recognition є одним із ключових компонентів.

Вона дозволяє отримувати голосові команди від користувача через мікрофон; використовувати онлайн сервіси розпізнавання мовлення для якісного перетворення мовлення у текст; застосовувати офлайн-рушії у разі відсутності інтернет-з’єднання; налаштовувати мовні моделі, покращувати точність розпізнавання завдяки адаптації до навколишнього шуму.

Це створює умови для гнучкого та надійного голосового введення. Коли бібліотека speech_recognition поєднується з pyttsx3 для синтезу мовлення, розробник може створити повноцінного голосового асистента, що слухає користувача, розуміє його запити та відповідає в зручній для користувача формі.

Таким чином, speech_recognition спрощує інтеграцію можливостей розпізнавання мовлення в Python-програми, надаючи розробнику все необхідне для перетворення звуку у текст за допомогою різних рушіїв та сервісів.

***Бібліотека difflib*** — це потужний інструмент для роботи з порівнянням послідовностей. Вона дозволяє визначати відмінності між текстовими рядками, списками або іншими ітерабельними структурами даних. Вона містить функції, які базуються принципах алгоритму Левенштейна і можуть використовуватися для задач, де необхідно визначити схожість між рядками.

***Бібліотека openai*** — це офіційна Python-бібліотека для взаємодії з OpenAI API, яка надає доступ до потужних моделей обробки природної мови та інших інструментів штучного інтелекту (наприклад, мовних моделей серій GPT). Завдяки цій бібліотеці розробники можуть інтегрувати у свої додатки можливості генерації тексту, відповідей на запитання, перекладу, резюмування та інші функції, що базуються на сучасних нейронних моделях.

Основна ідея взаємодії з OpenAI API полягає в обміні запитами та відповідями.

Для роботи з OpenAI API потрібен API-ключ, який можна отримати після реєстрації в OpenAI. Цей ключ використовується для підтвердження прав доступу до сервісу. Використання API є платним, але дозволяє гнучко використовувати залучені кошти на використання різних мовних моделі або на різні проєкти.

Запит зазвичай містить вказівку моделі, тип операції, а також параметри, такі як:

- prompt (текст-запит або контекст для генерації)
- max_tokens (обмеження кількості токенів у відповіді)
- temperature (ступінь "креативності" відповіді)
- інші параметри залежно від типу виклику

У відповідь на запит сервіс повертає згенерований текст або інші дані, залежно від методу. Отриманий результат можна обробити у коді, вивести користувачу, зберегти, передати до інших компонентів системи.

Для голосового асистента найбільш ймовірний сценарій — використання можливостей мовної моделі для аналізу та генерації відповідей на запити користувачів, що приходять після розпізнавання мовлення.

У контексті розробки персонального голосового асистента openai API може слугувати ядром для обробки природної мови. Поєднання з speech_recognition (розпізнавання мовлення) та pyttsx3 (синтез мовлення) дає можливість:

- Розуміння голосових запитів. Після того, як команда користувача перетворена у текст, її можна надіслати до OpenAI API для аналізу.
- Генерація змістовних відповідей. На основі користувацького запиту модель повертає відповіді у зрозумілому і логічному форматі.
- Контекстний діалог. За допомогою форматів ChatCompletion можна забезпечити контекстну розмову, де асистент "запам'ятовує" попередні повідомлення у діалозі.

Це створює основу для інтелектуальної взаємодії з користувачем: голосові команди, що надходять від користувача, після розпізнавання надсилаються до OpenAI API, обробляються та генерують повернення інформативних або корисних відповідей, які потім озвучуються за допомогою pyttsx3.

Таким чином, бібліотека openai надає зручний інтерфейс для інтеграції сучасних моделей штучного інтелекту в Python-застосунки. Завдяки їй, голосовий асистент може не лише розуміти запити користувачів, а й надавати адаптивні, контекстно-залежні та інтелектуальні відповіді.

# РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ГОЛОСОВОГО АСИСТЕНТА

## 2.1. Проєктування архітектури та алгоритму роботи голосового асистента

При розробці голосового асистента важливо визначити системні та функціональні вимоги, які забезпечать ефективну взаємодію з користувачем та високу якість роботи програмного продукту. Основні вимоги можна сформулювати таким чином.

Функціональні вимоги:

- Асистент повинен підтримувати голосовий ввід користувача та перетворювати його у текст.
- Асистент повинен синтезувати мовлення для озвучування відповідей.
- Асистент має здійснювати логічну обробку запитів користувача, залучаючи моделі штучного інтелекту (в даному випадку, OpenAI API) для формування змістовних та контекстуальних відповідей.
- Повинна бути можливість розширення функціоналу за допомогою нових команд.

Нефункціональні вимоги:

- Висока точність розпізнавання голосу за допомогою бібліотеки speech_recognition.
- Продуктивність та швидка реакція на голосові запити.
- Стабільність роботи в офлайн-режимі (хоча б для базового функціоналу синтезу мовлення).
- Безпека обміну даними з API сторонніх сервісів (зберігання ключів у файлі конфігурації або змінних оточення).
- Можливість додавання нових функцій та команд без суттєвих змін у базовій архітектурі.

Зручність використання:

- Інтуїтивний інтерфейс взаємодії (користувач дає команди голосом та отримує відповіді у голосовому форматі).
- Адаптація під різний оточуючий шум (автоматичне калібрування).

Для забезпечення логічної організації коду та зручності масштабування проєкт голосового асистента буде структуровано у вигляді директорії з такими основними файлами.

***__init__.py*** — порожній файл, який позначає директорію як Python-пакет та дозволяє імпортувати модулі всередині нього.

***main.py*** — Основний виконуваний файл, у якому відбувається ініціалізація голосового асистента, запуск циклу обробки команд та інтеграція всіх компонентів.

***commands.py*** — файл, що містить варіацію окремих команд та функцій асистента. Команди можуть бути реалізовані у вигляді окремих функцій або класів, які обробляють певний тип голосового запиту (наприклад, отримання поточної погоди, запуск музики, пошук інформації в інтернеті тощо).

***config.py*** — Файл налаштувань (конфігурацій). Містить ключі доступу до API, налаштування голосового рушія, параметри роботи асистента (швидкість мовлення, мова розпізнавання, тощо).

Також є файли «LICENSE.md» та «README.md». Хоч вони і не належать до функціоналу самого голосового асистента, але вони виконують роль документації та інструкції як їм користуватись.

Розберемо роботу основного виконуваного файлу main.py (див. рис. 2.1). Цей файл є основним голосового асистента і керує процесом, описаним у блок-схемі.




Рис. 2.1 Блок-схема алгоритму роботи файлу main.py

Основні етапи роботи:

- На початку коду імпортуються всі необхідні бібліотеки: speech_recognition, pyttsx3, openai, та інші, потрібні для функціонування голосового асистента. Також імпортуються внутрішні модулі commands та config. Завдяки імпорту бібліотек у програмі стають доступними інструменти для розпізнавання мовлення, синтезу голосу та виклику OpenAI API.
- Ініціалізація OpenAI API, токен від якого зберігається в зовнішньому модулі конфігурації — config.py.
- Ініціалізується початковий "промпт" або контекст для взаємодії з мовною моделлю.
- Ініціалізація голосового синтезатора. Використовуючи бібліотеку pyttsx3, у main.py створюється екземпляр голосового синтезатора.

Задане мовлення, швидкість, голос та інші параметри. Це дозволить асистенту відповісти користувачеві голосом.

- Вибір мікрофона. За допомогою speech_recognition вибирається джерело звуку. На цьому етапі може проводитись калібрування шуму, щоб підвищити точність розпізнавання.
- Цикл очікування команд. Основна логіка main.py побудована навколо циклу, який спочатку слухає голос користувача, отримує від нього голосову фразу, перетворює її у текст, і у результаті маємо текстовий запит користувача.
- Порівняння запиту з командами. Застосовується логіка визначення, що саме хотів сказати користувач. Використання алгоритму пошуку найбільш близької команди для обробки неточних запитів.
- Виконання команд. Якщо розпізнано певну ключову фразу, наприклад "Давай поговоримо", асистент переходить у режим діалогу з користувачем, використовуючи OpenAI API. На цьому етапі користувач задає питання, а асистент формує та озвучує відповідь. Діалог триває до тих пір, поки користувач не скаже фразу, яка означатиме завершення розмови (наприклад, "Досить говорити"). А також виконання інших команд: "Відкрий браузер", "Відкрий Youtube", "Відкрий Steam", "Відкрий Spotify", "Відкрий Telegram", "Заблокуй робочий стіл" при яких виконується відповідна дія.
- Повернення в режим прослуховування. Після виконання конкретної команди, якщо це не режим постійного діалогу, асистент знову повертається до циклу очікування голосових команд.
- Час очікування. Якщо користувач довго не говорить (умовно більше 15 секунд), асистент припиняє активний режим прослуховування і переходить у режим очікування активаційної фрази (в даному випадку він реагує на своє ім’я — "Айден"). Це означає, що асистент перестає "ставити питання" чи "чекати негайних команд", доки знову не буде сказана спеціальна активаційна фраза.

- Зрештою програма може бути зупинена користувачем або системно.

Важливо зазначити про використання алгоритму Левенштейна для спрощення розуміння команд користувача. При взаємодії з голосовим асистентом користувач може вимовляти команди неідеально: можуть бути присутні помилки вимови, неточності або варіації у формулюванні. Для підвищення толерантності асистента до таких неточностей використовується принцип пошуку найбільш схожого рядка на основі алгоритму Левенштейна або споріднених методів порівняння рядків.

Алгоритм Левенштейна дозволяє визначити, наскільки два рядки подібні, вимірюючи мінімальну кількість редагувань (вставок, видалень, замін символів), необхідну для перетворення одного рядка в інший. Чим менше правок потрібно, тим більше рядки схожі. На практиці це означає, що, навіть якщо користувач неправильно вимовив команду чи використав синоніми, асистент зможе інтерпретувати її коректно, підібравши команду, яка найкраще відповідає вхідному голосовому запиту.

У Python, для спрощення реалізації алгоритму Левенштейна та подібних метрик схожості рядків, часто використовують модуль difflib. Він має функцію get_close_matches(), яка порівнює заданий рядок з переліком можливих команд та повертає найбільш схожі варіанти.

Отже, якщо користувач неправильно вимовив команду, але його запит нагадує одну з існуючих команд, асистент зможе це врахувати. Наприклад:

Користувач вимовив: "Паговорим" замість "Поговоримо".

Якщо в commands є "поговоримо", функція get_close_matches() може визначити, що ці слова дуже схожі, і повернути "поговоримо" як найбільш ймовірну відповідність. Результатом буде "поговоримо", і асистент виконає необхідну команду.

Таким чином, використання алгоритму, реалізованого через функції модуля difflib, дозволяє голосовому асистенту бути більш "толерантним" до помилок користувача, що покращує якість взаємодії та зручність використання системи.

Підводячи підсумок підрозділу, можна сказати, що таке проєктування та алгоритм роботи голосового асистента забезпечує структурованість коду, зрозумілу архітектуру та можливість розширення функціональності. Вимоги, визначені на початку, гарантують, що асистент буде ефективним, зручним у використанні та стане основою для подальшого вдосконалення і масштабування системи.

## 2.2. Розробка голосового асистента у середовищі розробки

Перед початком роботи над створенням голосового асистента необхідно підготувати середовище розробки. Для цього необхідно встановити необхідні бібліотеки за допомогою pip.

Для встановлення speechrecognition:

*pip install speechrecognition*

Для встановлення speechrecognition:

*pip install pyttsx3*

Для встановлення speechrecognition:

*pip install openai*

А також створити файл конфігурації config.py. У цьому файлі зберігатимуться конфіденційні дані та API-ключ для OpenAI. Файл виглядатеме так:

*GPT_Token = "ВАШ_OPENAI_API_KEY"*

Цей ключ не варто поширювати цей ключ публічно.

У робочій директорії потрібно створити config.py, commands.py, main.py та інші допоміжні файли за потреби.

Нижче наведено фрагменти коду з файлу main, після яких буде детально розібрано роботу застосунка.

Лістинг 2 — Імпортування бібліотек

```
import speech_recognition as sr
import pyttsx3
import time
```

*import openai*

*import os*

*import webbrowser*

*import subprocess*

*from difflib import get_close_matches*

*import config*

*import commands*

*import threading*

*import ctypes*

Пояснення необхідних модулів про які не було згадано раніше:

***time*** — це модуль для роботи з часом (для реалізації очікування).

***os*** — це нативний модуль для взаємодії з операційною системою (обробка шляхів, змін оточення).

***webbrowser*** — це дає змогу відкривати веб-сторінки за вказаною URL-адресою за замовчуванням.

***subprocess*** — це дозволяє запускати системні команди або програми.

***from difflib import get_close_matches*** — це використовується для пошуку схожих рядків (порівняння команд користувача із зарезервованими командами).

***threading*** — це для виконання деяких завдань у фонових потоках, не блокуючи основний процес (наприклад, паралельний запис звуку та обробка команд).

***ctypes*** — це дозволяє взаємодіяти з системними DLL (наприклад, для блокування робочого столу у Windows).

Лістинг 3 — Ініціалізація OpenAI API-ключа

*openai.api_key = config.GPT_Token*

*messages = [*

*{"role": "system", "content": "Ти голосовий асистент. Твоє ім'я AiDen. Ти маєш відповідати на питання користувача та виконувати поставлені задачі."}*

*]*

Тут відбувається передача API-ключа до клієнта OpenAI. Після цього виклику можна використовувати функції openai.Completion.create() чи openai.ChatCompletion.create() для отримання відповідей від моделі.

Масив messages встановлює початковий контекст для мовної моделі. Повідомлення з роллю «system» визначає базову поведінку та інструкції для асистента. Це означає, що модель буде поводитись як голосовий асистент AiDen, відповідати на питання користувача і виконувати завдання, які користувач поставить. Надалі в цей масив будуть додаватися user та assistant повідомлення для ведення розмови.

Далі розглянемо наступний фрагмент коду, який відповідає за ініціалізацію голосового синтезатора та вибір мікрофона користувачем.

Лістинг 4 — Ініціалізація голосового синтезатора

```
engine = pyttsx3.init()
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[6].id)
```

pyttsx3.init() створює екземпляр об’єкта синтезатора голосу. engine.getProperty('voices') отримує список доступних голосів. Кожен голос має свій індекс. engine.setProperty('voice', voices[6].id) встановлює голос асистента, використовуючи 7-й за порядком голос (індексація з 0). Це завчасно встановлений український голос.

Залежно від системи та встановлених голосових рушіїв, кількість доступних голосів може варіюватися. Цей рядок можна налаштувати, змінюючи індекс, щоб вибрати інший голос.

Завантаження списку команд із зовнішнього файлу commands

```
commands = commands.word_list
```

Передбачається, що у файлі commands.py існує змінна word_list — список доступних команд. Це список рядків з командами та словник варіацій існуючих

команд. Після цього звертання commands буде містити перелік команд, які може розпізнавати асистент.

Подія для зупинки прослуховування

*stop_listening_event = threading.Event()*

Створюється об’єкт події (event) для керування потоками. stop_listening_event може використовуватися для сигналізації іншим потокам про необхідність припинити прослуховування мікрофона або виконання інших завдань. Це корисно для керування життєвим циклом асистента та можливості коректно завершувати його роботу.

Лістинг 5 — Функція вибору мікрофону

```
def list_microphones():
    print("Доступні мікрофони:")
    for index, name in enumerate(sr.Microphone.list_microphone_names()):
        print(f"{index}: {name}")
    while True:
        try:
            selected_index = int(input("Введіть номер мікрофона, який бажаєте використовувати: "))
            if 0 <= selected_index < len(sr.Microphone.list_microphone_names()):
                return selected_index
            else:
                print("Невірний номер. Спробуйте ще раз.")
        except ValueError:
            print("Будь ласка, введіть числове значення.")
```

Ця функція дозволяє користувачу інтерактивно вибрати мікрофон, якщо на системі є декілька пристроїв введення звуку. Спочатку виводиться список доступних мікрофонів із їхніми індексами, далі користувач вводить номер мікрофона. Перевіряється коректність введеного числа, якщо індекс виходить

за межі списку або некоректний, користувачу пропонується спробувати ще раз. Якщо все гаразд, функція повертає індекс вибраного мікрофона.

Така реалізація зручна, якщо потрібен контроль над тим, який саме мікрофон використовувати. Якщо користувач має кілька пристроїв введення (наприклад, вбудований мікрофон ноутбука, USB-мікрофон, Bluetooth-гарнітуру тощо), ця функція спростить вибір.

Вибір мікрофона користувачем

*device_index = list_microphones()*

Після виклику list_microphones() змінна device_index міститиме індекс обраного мікрофона. Це значення згодом буде використовуватися при ініціалізації об’єкта Microphone з speech_recognition для прослуховування голосових команд з конкретного пристрою.

Нижче наведений фрагмент коду, який реалізує функцію пошуку найбільш близької команди на основі вхідного голосового запиту, використовуючи поріг схожості

Лістинг 6 — Функція пошуку найбільш близької команди

```
def match_command(voice):
    closest_match = get_close_matches(voice, commands, n=1, cutoff=0.6)
    return closest_match[0] if closest_match else None
```

Пояснення логіки роботи.

Аргумент voice — це рядок, який містить текст, розпізнаний із голосу користувача. Наприклад, користувач сказав "відкрий бравзер" замість "відкрий браузер".

Функція get_close_matches() — використовується з модуля difflib. Вона порівнює рядок voice зі списком еталонних рядків (у нашому випадку зі змінною commands, яка містить список доступних голосових команд). Параметри виклику:

- voice: рядок, який потрібно порівняти зі списком.
- commands: список, серед якого здійснюється пошук схожих варіантів.

- n=1: повернути лише один найсхожіший варіант.
- cutoff=0.6: поріг схожості. Це означає, що якщо знайдена найбільш схожа команда має коефіцієнт схожості з voice менше 0.6 (60%), то вона не буде повернута.

Усередині get_close_matches() використовується алгоритм, подібний до алгоритму Левенштейна, який визначає, наскільки близькі два рядки. Чим більше букв та послідовностей співпадає, тим вищий рейтинг схожості.

В результаті виклику get_close_matches() отримуємо список варіантів, відсортований за найбільшою схожістю. Оскільки n=1, ми отримаємо або пустий список, або список з одним елементом.

Якщо closest_match містить хоча б один елемент, повертається цей елемент. Якщо ж список пустий (тобто, не знайшлося команди з достатньо високою схожістю), функція повертає None.

Припустимо, користувач каже «відкрий бравзер», функція порівнює «відкрий бравзер» з кожною командою в commands та визначає найсхожіший варіант.

Оскільки "відкрий бравзер" дуже схожий на "відкрий браузер" (можливо відміняється лише однією літерою), коефіцієнт схожості буде досить високим (припустимо 0.8), то функція поверне "відкрий браузер".

Якщо б користувач сказав щось зовсім відмінне, наприклад "машина червона", і серед команд немає нічого схожого, то get_close_matches() поверне пустий список, і функція поверне None.

Лістинг 7 — Основна функція

```
def handle_command(recognizer, source):
    global stop_listening_event
    while not stop_listening_event.is_set():
        try:
            recognizer.adjust_for_ambient_noise(source)
            audio = recognizer.listen(source, timeout=5, phrase_time_limit=15)
```

*voice = recognizer.recognize_google(audio, language="uk-UA").lower()*

*print("Розпізнано: " + voice)*

*matched_command = match_command(voice)*

handle_command(recognizer, source) — функція, яка приймає два аргументи recognizer (використовується для розпізнавання мовлення) та source (джерело звуку)

Використання global stop_listening_event означає, що функція звертається до глобальної змінної stop_listening_event, яка є об'єктом типу threading.Event. Якщо цей прапорець встановлений, це сигнал для виходу з циклу та припинення роботи прослуховування команд.

while not stop_listening_event.is_set(): — поки подія зупинки не встановлена, функція продовжує прослуховувати голос користувача.

recognizer.adjust_for_ambient_noise(source) — адаптує розпізнавач до фонового шуму. Це підвищує точність розпізнавання.

audio = recognizer.listen(source, timeout=5, phrase_time_limit=15) — розпізнавач слухає голос користувача:

- timeout=5 означає, що якщо протягом 5 секунд користувач нічого не скаже, listen припинить очікування.
- phrase_time_limit=15 означає, що розпізнавання окремої фрази користувача не буде тривати більше 15 секунд.

Отриманий аудіо-сигнал перетворюється у текст за допомогою сервісу Google Speech Recognition. language="uk-UA" означає, що розпізнавання виконується для української мови. lower() приводить результат до нижнього регістру для уніфікації порівнянь.

print("Розпізнано: " + voice) — виводить у консоль розпізнаний текст.

matched_command = match_command(voice) — виклик функції match_command(), яка порівнює розпізнаний текст з доступними командами.

Далі йде низка умов if/elif для перевірки, яка саме команда була розпізнана:

Лістинг 8 — Якщо розпізнано «давай поговоримо»

```
if matched_command == "давай поговоримо":
    print("AiDen: Про що поговоримо? Що Вас цікавить?")
    engine.say("Про що поговоримо? Що Вас цікавить?")
    engine.runAndWait()
    handle_conversation(recognizer, source)
```

Якщо користувач хоче поговорити, асистент переходить у режим діалогу. Він озвучує запитання «Про що поговоримо? Що Вас цікавить?» та викликає функцію handle_conversation(recognizer, source), яка буде описана далі.

Лістинг 9 — Якщо розпізнано «відкрий браузер»

```
elif matched_command == "відкрий браузер":
    print("AiDen: Відкриваю браузер.")
    engine.say("Відкриваю браузер.")
    engine.runAndWait()
    try:
        webbrowser.get('windows-default').open('http://www.google.com')
    except webbrowser.Error:
        print("Не вдалося знайти браузер. Відкриваю за замовчуванням.")
        webbrowser.open('http://www.google.com')
```

Асистент озвучує дію «Відкриваю браузер» та використовує модуль webbrowser для відкриття Google у браузері за замовчуванням. Якщо не вдається знайти браузер, відкриває URL за допомогою методів за замовчуванням.

Лістинг 10 — Якщо розпізнано «відкрий youtube»

```
elif matched_command == "відкрий youtube":
```

```
print("AiDen: Відкриваю YouTube.")
engine.say("Відкриваю YouTube.")
engine.runAndWait()
webbrowser.open('https://www.youtube.com')
```

Аналогічно до браузера, асистент промовляє дію та відкриває сторінку YouTube у браузері.

Лістинг 11 — Якщо розпізнано «запусти spotify»

```
elif matched_command == "запусти spotify":
    print("AiDen: Запускаю Spotify.")
    engine.say("Запускаю Spotify.")
    engine.runAndWait()
    try:
subprocess.Popen([r"C:\Users\User\AppData\Roaming\Spotify\Spotify.exe"])
    except FileNotFoundError:
        print("Не вдалося знайти Spotify. Переконайтеся, що він встановлений.")
```

Асистент спробує запустити Spotify, використовуючи subprocess.Popen() з повним шляхом до виконавчого файлу. Якщо файл не знайдено, виводиться повідомлення про помилку.

Лістинг 12 — Якщо розпізнано «запусти steam»

```
elif matched_command == "запусти steam":
    print("AiDen: Запускаю Steam.")
    engine.say("Запускаю Steam.")
    engine.runAndWait()
    try:
subprocess.Popen([r" E:\Steam\steam.exe"])
    except FileNotFoundError:
```

*print("Не вдалося знайти Steam. Переконайтеся, що він встановлений.")*

Подібно до Spotify, асистент спробує запустити Steam. Якщо файл не знайдено, виводиться повідомлення.

Лістинг 13 — Якщо розпізнано «запусти telegram»

```
elif matched_command == "запусти telegram":
    print("AiDen: Запускаю Telegram.")
    engine.say("Запускаю Telegram.")
    engine.runAndWait()
    try:
        subprocess.Popen([r"E:\Programs\Telegram Desktop\Telegram.exe"])
    except FileNotFoundError:
        print("Не вдалося знайти Telegram. Переконайтеся, що він встановлений.")
```

Асистент запускає Telegram з вказаного шляху. Якщо файл не знайдено, повідомляє про це користувача.

Лістинг 14 — Якщо розпізнано «заблокуй робочий стіл»

```
elif matched_command == "заблокуй робочий стіл":
    print("AiDen: Блокую робочий стіл.")
    engine.say("Блокую робочий стіл.")
    engine.runAndWait()
    ctypes.windll.user32.LockWorkStation()
```

Якщо користувач просить заблокувати робочий стіл, асистент відповідає голосом і текстом: «Блокую робочий стіл». Виклик ctypes.windll.user32.LockWorkStation() є системною функцією Windows, яка

блокує поточну сесію (вимагає повторного введення пароля для входу до робочого столу).

Лістинг 15 — Якщо розпізнано «дякую»

```
elif matched_command == "дякую":
    print("AiDen: Будь-ласка, завжди готовий допомогти")
    engine.say("Будь-ласка, завжди готовий допомогти")
    engine.runAndWait()
```

Якщо користувач каже "дякую", асистент чемно відповідає: "Будь-ласка, завжди готовий допомогти". Це просто ввічлива відповідь без додаткових дій.

Лістинг 16 — Якщо розпізнано «айден»

```
elif matched_command == "айден":
    print("AiDen: Слухаю Вас!")
    engine.say("Слухаю Вас!")
    engine.runAndWait()
```

Якщо користувач звертається до асистента по імені "Айден", асистент відповідає: "Слухаю Вас!". Це слугує сигналом про готовність асистента приймати команди.

Лістинг 17 — Якщо нічого не розпізнано

```
else:
    print("AiDen: Я не розумію цю команду.")
    engine.say("Я не розумію цю команду.")
    engine.runAndWait()
```

Якщо жодна з перелічених команд не була розпізнана як схожа з вхідним запитом, асистент відповідає: "Я не розумію цю команду." і озвучує це голосом. Таким чином користувач отримує зворотний зв'язок, що команда не була розпізнана.

Виняткові ситуації (sr.WaitTimeoutError, sr.UnknownValueError, sr.RequestError) обробляються для більш надійної та стабільної роботи асистента, забезпечуючи стійкість програми до несприятливих умов (відсутність реакції користувача, проблеми з розпізнаванням, помилки сервісу).

Далі наведено покроковий розбір функції handle_conversation(recognizer, source), яка реалізує діалоговий режим спілкування з голосовим асистентом.

Лістинг 18 — Функція діалогового режиму з асистентом

```
def handle_conversation(recognizer, source):
    global stop_listening_event
    while not stop_listening_event.is_set():
        try:
            recognizer.adjust_for_ambient_noise(source)
            audio = recognizer.listen(source, timeout=60, phrase_time_limit=15)
            voice = recognizer.recognize_google(audio, language="uk-UA").lower()
            print("Розпізнано: " + voice)
            matched_command = match_command(voice)
            if matched_command == "досить говорити":
                print("AiDen: Приємно було поговорити.")
                engine.say("Приємно було поговорити.")
                engine.runAndWait()
                handle_command(recognizer, source)
                break
```

Верхня частина коду була розібрана попередньо. matched_command = match_command(voice) порівнює розпізнану фразу з доступними командами. Якщо команда збігається з "досить говорити", асистент озвучує завершальну фразу та викликає функцію handle_command(recognizer, source), що повертає користувача у режим звичайного прослуховування команд.

Виконуючи break для виходу з циклу handle_conversation, фактично припиняючи діалоговий режим.

Лістинг 19 — Звернення до мовної моделі

```
messages.append({"role": "user", "content": voice})
chat = openai.ChatCompletion.create(
   model="gpt-3.5-turbo", messages=messages
)
reply = chat.choices[0].message.content
print(f"AiDen:  {reply}")
engine.say(reply)
engine.runAndWait()
```

messages.append({"role": "user", "content": voice}): Додає репліку користувача до історії повідомлень messages. Цей масив використовується для збереження контексту розмови.

chat = openai.ChatCompletion.create(...) відправляє контекст розмови та запит до моделі "gpt-3.5-turbo". Відповідь моделі зберігається у reply.

Асистент виводить отриману відповідь у консоль та озвучує її за допомогою engine.say() та engine.runAndWait().

Це забезпечує інтерактивний діалоговий режим: користувач говорить — модель розуміє запит, формує відповідь, а асистент озвучує її. Також модель пам'ятає контекст розмови завдяки списку messages.

Лістинг 20 — Обробка винятків

```
except sr.WaitTimeoutError:
   print("AiDen: Час очікування команди закінчився. Повертаюсь у режим очікування.")
   return
except sr.UnknownValueError:
   print('Голос не розпізнано')
except sr.RequestError:
```

*print('Невідома помилка!')*

sr.WaitTimeoutError якщо користувач мовчить довше, ніж timeout, асистент повідомляє про закінчення очікування та повертається у режим очікування команд (завершує handle_conversation).

sr.UnknownValueError якщо голос не був розпізнаний, виводиться відповідне повідомлення, цикл продовжується.

sr.RequestError помилка сервісу розпізнавання, виводиться повідомлення про невідому помилку, але асистент продовжує цикл.

Розберемо останню, але не менш важливу функцію створення розпізнавача

Лістинг 21 — Створення розпізнавача та відкриття мікрофона

```
def listen_for_activation():
    recognizer = sr.Recognizer()
    with sr.Microphone(device_index=device_index) as source:
```

Створюється новий екземпляр розпізнавача голосу recognizer. Контекстний менеджер with sr.Microphone(device_index=device_index) відкриває мікрофон, заданий раніше обраним індексом device_index.

Всередині with блоку тепер використовується цей мікрофон як джерело звуку.

Лістинг 22 — Основний цикл очікування активаційної фрази

```
while True:
    try:
        print("AiDen: Очікую своє ім'я...")
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.listen(source)
        voice = recognizer.recognize_google(audio, language="uk-UA").lower()
matched_command = match_command(voice)
```

Цей цикл постійно працює, доки програма не буде зупинена користувачем. На кожній ітерації виводиться повідомлення «Очікую своє ім'я...» — асистент слухає, чи назве користувач його ім’я.

adjust_for_ambient_noise(source) знову калібрує розпізнавач під навколишній шум.

recognizer.listen(source) записує наступну сказану користувачем фразу.

Розпізнавач за допомогою recognize_google перетворює аудіо у текст, voice містить цей текст у нижньому регістрі.

Виклик match_command(voice) визначає, чи схожий розпізнаний текст на одну з доступних команд. Це може бути корисним, якщо користувач скаже щось схоже на ім’я асистента.

Лістинг 23 — Активація при згадці імені асистента

```
if matched_command == "айден" in voice:
    print("AiDen: Слухаю ваші команди.")
    engine.say("Слухаю ваші команди.")
    engine.runAndWait()
    handle_command(recognizer, source)
```

Оператор «айден» in voice повертає True або False, залежно від того, чи міститься підрядок «айден» у змінній voice. Основний намір коду — активувати асистента коли користувач вимовить "айден". Як тільки умова істинна асистент озвучує: «Слухаю ваші команди.»

Викликається handle_command(recognizer, source), яка переходить у режим активного прийому команд.

Отже, listen_for_activation() — це режим "пасивного слухання". Він чекає, доки користувач не скаже «Айден». Після цього асистент прокидається і починає обробляти команди у режимі handle_command().

Лістинг 23 — Обробка помилок

```
except sr.UnknownValueError:
    print('Голос не розпізнано')
```

```
except sr.RequestError:
    print('Невідома помилка!')
```

sr.UnknownValueError: Якщо не вдалося розпізнати сказане (поганий звук, шум, нерозбірлива мова), виводиться повідомлення "Голос не розпізнано".

sr.RequestError: Якщо виникла помилка з мережею чи сервісом розпізнавання, виводиться "Невідома помилка!".

## 2.3. Тестування голосового асистента та оцінка ефективності голосового асистента

Після розробки голосового асистента, наступним ключовим кроком є тестування функціональності, стабільності роботи та оцінка ефективності взаємодії з користувачем. Тестування допомагає виявити недоліки, помилки та проблемні моменти у розпізнаванні мовлення, реагуванні на команди та формуванні відповідей, а також дозволяє отримати зворотний зв’язок від реальних користувачів.

На початковому етапі варто перевірити роботу окремих модулів та функцій асистента:

- Перевірка точності перетворення голосу в текст за допомогою speech_recognition. Варто тестувати мову у різних умовах: з фоновим шумом, з різним тембром голосу та темпом мовлення.
- Переконатися, що асистент коректно озвучує відповіді. Перевірити різні голоси та налаштування швидкості й гучності.
- Переконатися у стабільній роботі з OpenAI API, правильності формування запитів та обробки відповідей.

Після успішного тестування окремих компонентів слід перевірити інтеграцію:

***Послідовність виконання команд.*** Вимовити активаційну фразу, потім різні команди . Переконатися, що асистент виконує потрібні дії (див рис. 2.1).

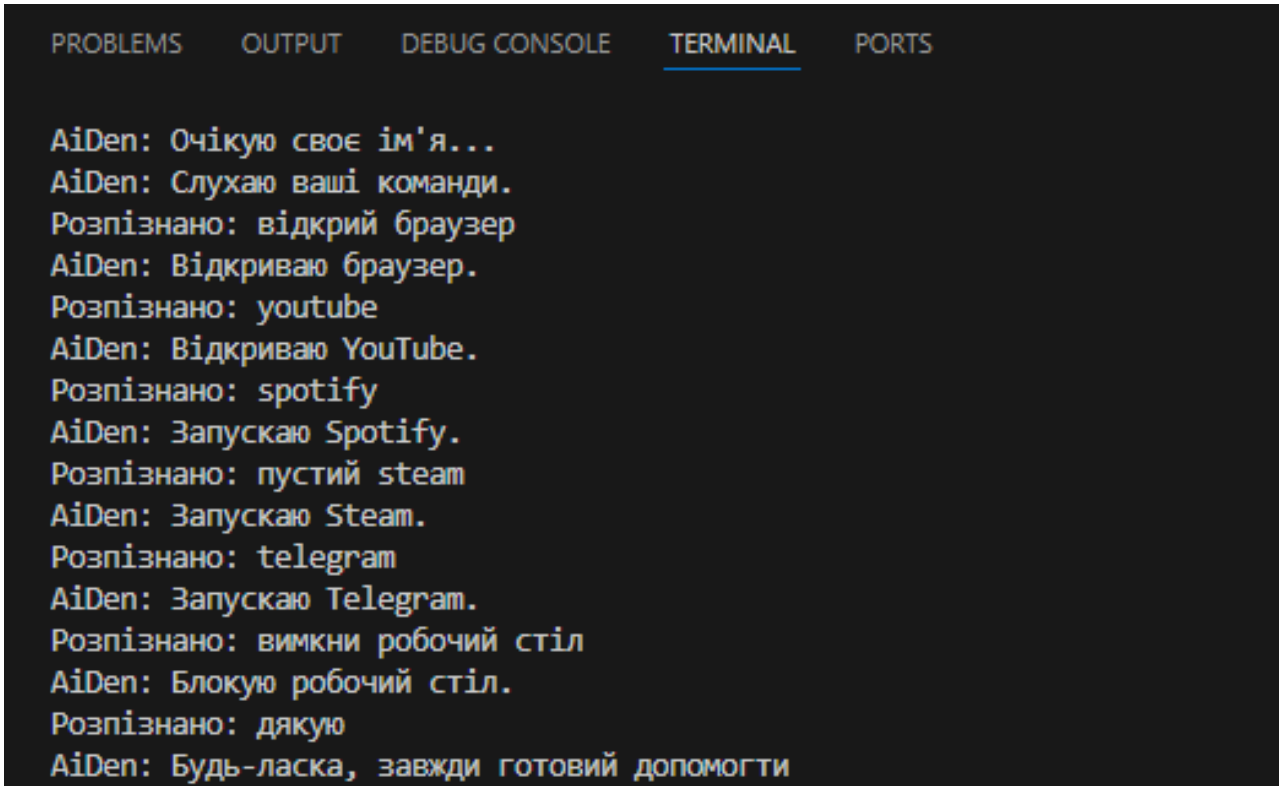



Рис. 2.1 Тестування виконання команд

Даний скріншот демонструє тест роботи голосового асистента після вимовляння активаційної фрази.

Після вимовляння активаційної фрази «Айден» асистент успішно перейшов у режим активного слухання команд і повідомив користувача про готовність приймати команди: «AiDen: Слухаю ваші команди.»

На скріншоті видно, що було вимовлено команди з певними відхиленнями у вимові, але асистент їх коректно інтерпретував. Наприклад:

«пустий steam» — замість «запусти steam»

Помітно, що асистент реагував на команди навіть зі спотвореною вимовою та виконував відповідні дії:

«відкрий браузер» — Відкриття браузера

«youtube» — Відкриття YouTube

«spotify» — Запуск Spotify

«пустий steam» — Запуск Steam

«telegram» — Запуск Telegram

«вимкни робочий стіл» — Блокування робочого столу

«дякую» → Відповідь «Будь-ласка, завжди готовий допомогти»

Умови тестування, тест проводився у звичайних кімнатних умовах з невеликим фоновим шумом. Це демонструє, що асистент достатньо стійкий до незначних фонів і може коректно розпізнавати команди при нормальному рівні шуму.

З першого тесту можна зробити наступні висновки:

- Асистент успішно реагує на активаційну фразу.
- Точність розпізнавання є задовільною навіть при певних відхиленнях у вимові команд.
- Команди успішно виконуються: запуск браузера, відкриття YouTube, Spotify, Steam, Telegram, а також блокування робочого столу.
- Асистент надає ввічливі відповіді та підтверджує виконання команд голосом.

Ці результати свідчать про те, що розроблений голосовий асистент добре справляється з основними завданнями.

***Діалоговий режим.*** Наступним кроком необхідно протестувати режим розмови з асистентом через функцію handle_conversation. Перевірити, чи модель коректно відповідає на користувацькі питання, чи правильно завершує розмову за командою «досить говорити» (див. рис. 3).

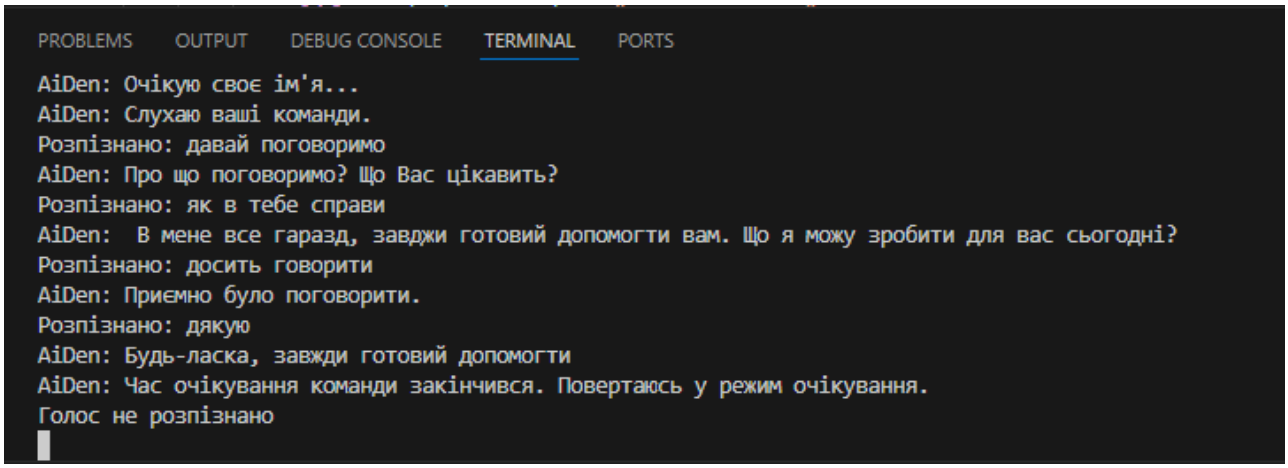



Рис. 2.2 Тестування діалогового режиму

Даний скріншот демонструє процес тестування діалогового режиму голосового асистента.

Після активації «Айден» асистент переходить у режим очікування команд та сприймає наступну фразу «давай поговоримо». Асистент розпізнав команду і перейшов у діалоговий режим, запитавши: «Про що поговоримо? Що Вас цікавить?». Користувач задав питання: «як у тебе справи». Асистент зрозумів запит і надав відповідь у формі діалогу: «В мене все гаразд, завжди готовий допомогти вам. Що я можу зробити для вас сьогодні?». Користувач сказав: «досить говорити». Асистент коректно розпізнав команду і ввічливо завершив діалог: «Приємно було поговорити». Після цього він повернувся до звичайного режиму очікування команд та повідомив про це користувача. Коли користувач деякий час нічого не говорить, асистент виводить: «Час очікування команди закінчився. Повертаюсь у режим очікування». Це означає, що асистент повернувся до пасивного прослуховування, чекаючи активаційної фрази.

Це демонструє інтеграцію з мовною моделлю, яка формує змістовні відповіді та підтримує контекст попередніх реплік.

На скріншоті також видно повідомлення «Голос не розпізнано», що свідчить про обробку виключень та інформування користувача, якщо розпізнавання мови не дало результатів.

З другого тесту можна зробити наступні висновки:

- Асистент успішно переходить з режиму очікування у діалоговий режим за командою «давай поговоримо».
- Запитує користувача про тему розмови та надає змістовні відповіді.
- Коректно завершує діалог за командою «досить говорити».
- Відповідає на ввічливі репліки, зберігаючи дружній тон.
- Режим очікування та обробка таймаутів працюють коректно.
- Асистент дає зворотний зв’язок про нерозпізнані фрази.

Цей тест підтверджує, що діалоговий режим реалізований успішно асистент може підтримувати прості розмови, розуміти команди керування діалогом і надавати логічні, релевантні відповіді.

Тестування та оцінка ефективності голосового асистента є невід’ємною частиною його життєвого циклу. Послідовне, комплексне тестування допомагає виявити слабкі місця, покращити точність, продуктивність та якість взаємодії з користувачем. Окрім технічних критеріїв, оцінка ефективності включає зручність використання, задоволеність користувачів та простоту масштабування. В результаті ретельного тестування розробник може приймати зважені рішення щодо подальшого вдосконалення асистента та адаптації під реальні потреби користувачів.

# РОЗДІЛ 3. ПЕРСПЕКТИВИ РОЗВИТКУ ГОЛОСОВОГО АСИСТЕНТА

## 3.1. Порівняння розробленого голосового асистента з існуючими рішеннями

Переваги розробленого голосового асистента:

- Гнучкість та кастомізація. На відміну від готових комерційних рішень, розроблений асистент можна вільно налаштовувати та розширювати. Додавання нових команд, зміна логіки обробки мовлення, інтеграція з новими сервісами чи API може бути виконана за потребою, без обмежень, які накладає закрите програмне забезпечення.
- Локалізація та підтримка української мови. Хоча відомі голосові асистенти поступово поліпшують підтримку української мови, розроблене рішення з самого початку орієнтоване на україномовний контент.
- Контроль конфіденційності. Оскільки код асистента відкритий для розробника, він має повний контроль над тим, які дані надсилаються до зовнішніх сервісів та API. Це спрощує дотримання вимог безпеки та конфіденційності порівняно з хмарними асистентами великих корпорацій.
- Автономність. Можливість використовувати асистента частково чи повністю офлайн (наприклад, синтез мовлення через pyttsx3 без підключення до мережі) може бути корисною у середовищах з обмеженим доступом до інтернету, що не завжди можливо у відомих комерційних рішеннях.

Недоліки розробленого голосового асистента:

- Обмеженість ресурсів та масштабів. Відомі асистенти, такі як Google Assistant чи Alexa, спираються на масштабні обчислювальні ресурси, великі набори даних та потужні хмарні сервіси. Це дозволяє їм

розпізнавати складні мовні конструкції та підтримувати широкий спектр функцій. Розроблений асистент має доступ до значно менших ресурсів, що може впливати на точність розпізнавання, продуктивність та глибину інтеграції.

- Обмежений екосистемний підхід. Комерційні асистенти часто інтегровані у велику екосистему продуктів та сервісів (розумний дім, стрімінгові платформи, поштові клієнти, календарі тощо). Розроблене рішення потребує додаткових зусиль з інтеграції та налаштування для роботи з новими сервісами, а користувачеві доведеться самостійно розширювати функціонал.
- Менша точність розпізнавання та знань. Без доступу до масштабних моделей мовлення чи великих мовних моделей останніх генерацій асистент може гірше розуміти складні запити або не мати знань у специфічних сферах.
- Відсутність брендової підтримки та довіри користувачів. Відомі асистенти мають певний кредит довіри через свої відомі бренди. Користувачі знають, хто стоїть за продуктом, та розраховують на довгострокову підтримку. Розроблене рішення — це індивідуальний проєкт, що може вимагати додаткових зусиль для завоювання довіри користувачів та покращення користувацького досвіду.

Розроблений голосовий асистент може бути цікавим вибором для специфічних застосувань, де потрібна гнучкість, локалізація та контроль над даними. Водночас, він поступається відомим комерційним рішенням у масштабованості, точності розпізнавання та інтеграції з великою екосистемою продуктів. Залежно від потреб користувача (наприклад, орієнтація на локальний ринок, висока конфіденційність, можливість глибокої кастомізації) розроблений асистент може мати свої унікальні переваги, які недоступні у масових продуктах.

## 3.2. Використання голосового асистента в реальних умовах, потенційні користувачі

Розроблений голосовий асистент орієнтований на виконання ролі персонального помічника в домашніх умовах, виступаючи зручним інтерфейсом для автоматизації та прискорення рутинних процесів роботи за домашнім комп’ютером. Його використання сприяє економії часу, підвищенню продуктивності та створенню більш комфортних умов для взаємодії з технікою.

Швидке відкриття браузера, пошук у мережі, запуск медіаплеєра чи стрімінгових сервісів відбувається без потреби у фізичній взаємодії з комп’ютером. Це особливо зручно, коли користувач зайнятий іншими справами — приготуванням їжі, роботою за робочим столом без клавіатури під рукою чи навіть відпочинком на дивані.

Асистент може легко відкрити YouTube, Spotify чи інші ресурси одним голосовим запитом. Завдяки цьому користувач не витрачає час на навігацію по меню, пошук потрібних файлів чи програм. Одним словом, асистент спрощує доступ до контенту.

Можливість поспілкуватися з асистентом у діалоговому режимі дозволяє швидко отримати відповіді на прості питання, підказки чи поради. Це створює відчуття взаємодії з інтелектуальним співрозмовником, а не зі складною технічною системою.

Заблокувати робочий стіл або виконати інші системні дії (наприклад, вимкнути звук, перемкнути між вікнами) можна без потреби вручну здійснювати відповідні дії. Це підвищує зручність щоденного використання комп’ютера.

Потенційні користувачі:

- Люди, які проводять багато часу за комп’ютером вдома (наприклад, дистанційно працюючі, студенти, геймери чи просто активні користувачі мережі), можуть отримати вигоду від прискорення рутинних операцій.

- Голосовий асистент може стати справжнім помічником для людей з порушеннями рухового апарату, адже вони зможуть керувати комп’ютером голосом, без потреби користуватися мишею чи клавіатурою.
- Ті, хто цікавиться новими технологіями, штучним інтелектом або просто хоче мати більше контролю над налаштуваннями системи, оцінять можливість тонкої кастомізації асистента й експерименти з командами та інтеграціями.

Персональний голосовий асистент вдома — це не лише зручність, а й крок до більш природної взаємодії з комп’ютером. Такий інструмент допомагає заощадити час, зробити роботу, розваги та інформаційний пошук більш ефективними. Оскільки асистент зосереджений саме на домашньому використанні, він найкраще підходить для тих, хто прагне створити у себе комфортне та інтелектуально кероване середовище для роботи та відпочинку.

## 3.3. Розрахунки комерційної вигоди від розробки голосового асистента

Для оцінки комерційної вигоди необхідно розглянути орієнтовну бізнес-модель та провести приблизні фінансові розрахунки. Залежно від стратегії монетизації, продукт може приносити дохід за рахунок прямих продажів, підписок або інтеграції з платними сервісами. Нижче наведено спрощений приклад оцінки.

Цільова аудиторія: Домашні користувачі, які шукають зручних рішень для автоматизації процесів на своєму ПК.

Модель монетизації: Разовий продаж ліцензії або підписка (наприклад, 1–2 долари на місяць) за покращену версію з додатковими функціями чи інтеграціями.

Головна перевага це локалізація під українську мову, гнучкість та кастомізація.

Основний недолік це менша функціональність порівняно з великими гравцями ринку.

Варіанти ціноутворення:

- Разовий платіж за ліцензію «Pro»-версії асистента: 10 доларів за копію.
- Місячна підписка 1 долар за розширений функціонал (додаткові команди, інтеграція з OpenAI без додаткової оплати).

Для приблизного розрахунку візьмемо підпискову модель, адже вона забезпечує стабільніший грошовий потік.

Основні щомісячні витрати на утримання продукту можна представити у вигляді формули (3.1):

$$C_{month} = C_{devs} + C_{hosting} + C_{api} + C_{marketing}, \text{ (3.1)}$$

Де $C_{devs}$- витрати на розробників.

$C_{hosting}$- витрати на хостинг та домен.

$C_{api}$- витрати на користування API (наприклад, OpenAI)

$C_{marketing}$- витрати на маркетинг

Таким чином:

$$C_{month} = 1000 + 50 + 50 + 100 = 1200 \text{ дол./місяць}$$

Розрахуємо доходи від підписки, припустимо, що доступний платний тариф із середньою ціною $p$ = 1 дол./місяць. Якщо з усіх активних користувачів із загальної бази $U$ користувачів конвертується у платну підписку певний відсоток $\alpha$, тоді кількість платних користувачів становитиме (3.2):

$$U_{paid} = \alpha U, \text{ (3.2)}$$

Доходи на місяць при цьому будуть (3.3):

$$R_{month} = U_{paid} \times p = \alpha U \times p \text{ , (3.3)}$$

Наприклад, якщо $\alpha$ = 10%, то (3.4):

$$R_{month} = 0{,}1U_{paid} \times 1 = 0{,}1U, \text{ (3.4)}$$

Розрахуємо точку беззбитковості. Для виходу на беззбитковість потрібно, щоб щомісячні доходи дорівнювали витратам (3.5):

$$R_{month} = C_{month}, \text{ (3.5)}$$

Підставляючи $R_{month} = 0{,}1U$ та $C_{month} = 1200$ (3.6):

$$U = \frac{1200}{0{,}1} = 12000, \text{ (3.6)}$$

Отже, для досягнення беззбитковості за умови 10% платних користувачів і ціни 1 дол./місяць, необхідна база щонайменше 12 000 активних користувачів.

Оцінимо прибутковість при масштабування. Припустимо, що вдається залучити 20 000 активних користувачів, з яких 10% платять 2000 користувачів (3.7):

$$R_{month} = U_{paid} \times p = 0{,}1 \times 20000 \times 1 = 2000, \text{ (3.7)}$$

Якщо витрати зросли до $C_{month} = 1400$ дол./місяць через невеликі додаткові витрати на підтримку (3.8):

$$\text{Чистий прибуток} = R_{month} - C_{month} = 2000 - 1400 = 600, \text{ (3.8)}$$

Можливості покращення маржинальності. Формула для розрахунку необхідної кількості користувачів при різних тарифах:

$$U = \frac{C_{month}}{\alpha \times p}, \quad (3.10)$$

При збільшенні ціни, наприклад $p$ = 2 дол./місяць, кількість користувачів для беззбитковості зменшується:

$$U = \frac{1200}{0{,}1 \times 2} = \frac{1200}{0{,}2} = 6000, \quad (3.10)$$

Покращуючи будь-який із цих показників (збільшуючи аудиторію, підвищуючи ціну чи конверсію, оптимізуючи витрати), можна досягти стабільного прибутку та рентабельності.

Комерційна вигода залежить від масштабів проникнення на ринок, проценту конверсії у платних користувачів та ефективності маркетингу. На старті розроблений голосовий асистент матиме високі витрати та низькі доходи. Однак, при досягненні широкої користувацької бази (десятки тисяч користувачів) та створенні додаткових платних функцій проект може стати прибутковим.

Отже, реальний сценарій отримання комерційної вигоди потребує збільшення аудиторії, активної маркетингової діяльності, вдосконалення функціоналу й оптимізації витрат.

# ВИСНОВКИ

В ході магістерської дипломної роботи було розроблено персонального голосового асистента, призначеного для ефективної інтеракції з користувачем у домашніх умовах. Асистент реалізований з використанням мови програмування Python, бібліотек для розпізнавання та синтезу мовлення, а також інтеграції з OpenAI API для забезпечення діалогового режиму з елементами штучного інтелекту. Особливий наголос було зроблено на підтримці української мови та можливості гнучкого налаштування під потреби конкретного користувача.

В теоретичній частині роботи здійснено огляд існуючих інформаційних технологій та інструментів, застосовуваних для розробки голосових асистентів, охарактеризовано стандарти та протоколи, а також обґрунтовано вибір середовища розробки, мов програмування та бібліотек. Було проаналізовано переваги використання Python, роботу з бібліотеками speech_recognition, pyttsx3 і openai, а також продемонстровано принципи інтеграції з системами контролю версій і середовищем розробки.

У процесі проєктування визначено вимоги до системи, спроєктовано архітектуру голосового асистента, алгоритми його роботи та механізми обробки голосових команд. Реалізовано функціонал розпізнавання мовлення, синтезу голосової відповіді, вибору та коригування мікрофона, а також перевірено здатність асистента розуміти команди з похибками у вимові за допомогою алгоритму пошуку найбільш близьких відповідностей. Особлива увага приділена діалоговому режиму, де інтегровано модель ChatGPT для надання інтелектуальних відповідей та підтримання контексту розмови.

Практичне тестування виконано у реальних домашніх умовах із помірним фоновим шумом. Результати тестування підтвердили працездатність системи: асистент коректно розпізнає активаційну фразу, реагує на відхилення у вимові команд, виконує прості системні операції, відкриває веб-сторінки та програми. Діалоговий режим дозволив продемонструвати можливість використання

асистента як інтелектуального співрозмовника, що відповідає на інформаційні запити користувача. Проведено також первинну оцінку комерційної вигоди, аналіз економічної доцільності розробки та перспективи масштабування для досягнення беззбитковості.

Порівняння розробленого рішення з існуючими відомими голосовими асистентами виявило низку переваг: гнучкість налаштувань, орієнтацію на українську мову, легку інтеграцію з новими сервісами та можливість контролю конфіденційності. Водночас було визначено недоліки: менша точність розпізнавання у складних умовах, відсутність великої екосистеми готових інтеграцій, необхідність самостійно розширювати функціонал та оптимізувати продуктивність.

Таким чином, виконана робота продемонструвала повний цикл розробки персонального голосового асистента: від ідеї та проєктування до реалізації, тестування, оцінки ефективності та аналізу перспектив розвитку. Серед переваг розробленого асистента, є краща підтримка української мови, можливість змінювати його функціонал, а також проста взаємодія з ним. Отримані результати можуть бути основою для подальшого вдосконалення, додавання нових функцій, інтеграцій з IoT-пристроями, розширення діалогових можливостей та комерціалізації проєкту. Це відкриває перспективи створення корисного продукту для широкого кола користувачів, здатного суттєво спростити взаємодію з комп’ютером та підвищити ефективність повсякденних завдань.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Аналіз корисності розробки комп’ютерно-інтегрованої системи голосового управління базовими функціями комп’ютера та проблеми використання голосових помічників / В.Р. Жуковський // Інформатика, обчислювальна техніка та автоматизація. – 2021. – №6. – С. 71–75.
2. Використання ембедінгів голосу в інтегрованих системах для діаризації мовлення / І. Заєць, В. Бридінський, Д. Сабодашко, Ю. Хома, Х. Руда, М. Швед // Комп'ютерні системи та мережі. – 2024. – Вип. 6, №1. – С. 54–66.
3. Використання штучного інтелекту в системах голосової автентифікації / О. В. Голубєва, О. В. Ковальов // Матеріали 24-ї Міжнародної науково-практичної конференції молодих учених і студентів 'Радіоелектроніка та молодь у XXI столітті'. – Харків: ХНУРЕ, 2020. – С. 95–96.
4. Використання штучного інтелекту в системах голосової автентифікації / С.М. Радченко // Радіоелектроніка та інформатика. – 2023. – №2. – С. 56–63.
5. Використання штучного інтелекту у вивченні іноземної мови здобувачами освіти / Майя Кадемія, Інесса Візнюк, Анна Поліщук, Сергій Долинний // Сучасні інформаційні технології та інноваційні методики навчання в підготовці фахівців: методологія, теорія, досвід, проблеми. – 2022. – Вип. 63. – С. 153-163.
6. Використання штучного інтелекту у вищій освіті / О.В. Топузов, С.В. Алексєєва // Український педагогічний журнал. – 2024. – №1. – С. 5–11.
7. Вивчення основ штучного інтелекту при створенні голосового асистенту / В.О. Болілий, Л.П. Суховірська, С. Абуватфа // Педагогічні науки. – 2023. – №1. – С. 89–94.
8. Голосова ідентифікація диктора як один із сучасних біометричних методів ідентифікації особи / Х. В. Луценко, К. В. Нікулін // Теорія та

практика судової експертизи і криміналістики. – 2019. – Вип. 1. – С. 239–245.

9. Голосовий асистент для роботи з ПК : кваліфікаційна робота освітнього рівня «Бакалавр» / Н.М. Колачик. – Тернопіль : ТНТУ, 2023. – 59 с.
10. Дослідження та розробка голосового асистента на мобільні платформи для навчальних цілей / Б. Грибач // Вісник ТНТУ. – 2018. – №4. – С. 44–50.
11. Інформаційна система перетворення звукового україномовного тексту в письмовий на основі методів NLP та машинного навчання / Ю. Тищук, В. Висоцька, О. Власенко // Системи обробки інформації. – 2022. – Вип. 12. – С. 23–51.
12. Інформаційна технологія розпізнавання статі за голосом / Д. Коштура // Системи обробки інформації. – 2023. – Вип. 13. – С. 350–360.
13. Метод розпізнавання голосових команд з допомогою штучних нейронних мереж / А.В. Мелянчук // Науковий вісник Західноукраїнського національного університету. – 2023. – №2. – С. 44–50.
14. Методи і технології біометричного розпізнавання суб’єкта за голосом / М. В. Ткаченко, Р. М. Федоренко, І. Г. Зотова // Збірник наукових праць Центру воєнно-стратегічних досліджень НУОУ імені Івана Черняховського. – 2018. – №2 (63). – С. 110–113.
15. Можливості використання штучного інтелекту в освітньому процесі закладів середньої освіти в умовах воєнного стану / О.В. Топузов, С.В. Алексєєва // Український педагогічний журнал. – 2024. – №1. – С. 5–11.
16. Оцінка ефективності засобів розпізнавання ключових слів у голосовому сигналі / А.В. Ткаченко // Інформатика, обчислювальна техніка та автоматизація. – 2023. – №3. – С. 112–118.
17. Перспективи застосування інструментів на основі штучного інтелекту у викладанні української мови як іноземної / О.М. Паламар, М.С. Науменко // Український освітній журнал. – 2024. – №1. – С. 12–19.

18. Порівняльний аналіз методів реалізації голосової біометрії / О. В. Бережний, О. В. Ковальов // Вісник Вінницького політехнічного інституту. – 2021. – №2. – С. 112–118.
19. Програмний модуль голосового асистента для синтезу та розпізнавання мовлення / І.В. Петренко, О.М. Сидоренко // Вісник НУ «Львівська політехніка». – 2022. – №895. – С. 98–104.
20. Розробка алгоритму ідентифікації та обробки голосових команд та програмна реалізація системи голосового управління / О.В. Мельничук // Наукові праці НГУ. – 2023. – №5. – С. 123–130.
21. Розробка голосового помічника на основі нейросистем / О.І. Ковальчук, О.В. Ковальчук // Автоматизація технологічних та бізнес-процесів. – 2022. – №14. – С. 45–52.
22. Розроблення веб-застосунку голосового асистента з використанням технології штучного інтелекту : кваліфікаційна робота / С.О. Оліщук. – Київ : ДУІКТ, 2024. – 62 с.
23. Розроблення і дослідження інструментальних засобів для нейромережевого аналізу голосу слухачів системи дистанційного навчання / Д.О. Чернишев, В.М. Михайленко, Л.А. Терейковська // Управління розвитком складних систем. – 2020. – Вип. 43. – С. 110–115.
24. Система розпізнавання голосу : дипломний проект ... бакалавра : 6.050102 Комп’ютерна інженерія / У.С. Стефанішина. – Київ: НТУУ "КПІ", 2020. – 85 с.
25. Стратегія розвитку штучного інтелекту в Україні / за ред. І.В. Сергієнка. – Київ : Інститут штучного інтелекту, 2023. – 256 с.
26. Сучасні методи автоматичної ідентифікації диктора за голосом / М.В. Ткаченко, Р.М. Федоренко, Д.С. Берестов // Збірник наукових праць Центру воєнно-стратегічних досліджень НУОУ імені Івана Черняховського. – 2019. – №2 (64). – С. 54–60.

27. Учасники проектів Вікімедіа. Відстань Левенштейна – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Відстань_Левенштейна (дата звернення: 14.01.2025).
28. Учасники проектів Вікімедіа. Рекурентна нейронна мережа – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Рекурентна_нейронна_мережа (дата звернення: 22.01.2025).
29. Штучний інтелект в освіті: використання без порушення принципів академічної доброчесності / С.П. Паламар, М.С. Науменко // Освітній дискурс. – 2024. – №2. – С. 45–53.
30. Штучний інтелект в освіті: використання без порушення принципів академічної доброчесності / С.П. Паламар, М.С. Науменко // Освітній дискурс. – 2024. – №2. – С. 45-53.
31. Штучний інтелект в освітньому процесі та наукових дослідженнях / О. Панухник // Галицький економічний вісник. – 2023. – Том 83, №4. – С. 202-211.
32. Штучний інтелект та відкрита наука в освіті / М.В. Марієнко, М.П. Шишкіна, О.А. Коновал // Фізико-математична освіта. – 2022. – №24. – С. 381–388.
33. Штучний інтелект та відкрита наука в освіті / М.В. Марієнко, М.П. Шишкіна, О.А. Коновал // Фізико-математична освіта. – 2023. – Том 38, №1. – С. 51-60.
34. A Comparative Analysis of Smart Voice Assistants / J. Smith, A. Johnson // IEEE Conference on Artificial Intelligence. – 2021. – С. 45–52.
35. Artificial Intelligence-based Voice Assistant / M. Brown, L. Davis // IEEE Transactions on Neural Networks and Learning Systems. – 2020. – Т. 31, №9. – С. 3456–3465.
36. Automatic Speech Recognition: Systematic Literature Review / S. Wilson, D. Martinez // IEEE Access. – 2021. – Т. 9. – С. 123456–123467.

37. Deep Learning Enabled Semantic Communications With Speech Recognition and Synthesis / Y. Zhang, H. Chen // IEEE Communications Magazine. – 2022. – Т. 60, №5. – С. 42–48.
38. Deep learning-based expressive speech synthesis: a systematic review of approaches, challenges, and resources / H. Barakat, O. Turk, C. Demiroglu // EURASIP Journal on Audio, Speech, and Music Processing. – 2024. – №11. – С. 1–25.
39. Enhanced AI Voice Assistance using Machine Learning and NLP / R. Lee, S. Kim // IEEE International Conference on Machine Learning and Applications. – 2022. – С. 789–794.
40. Google Assistant – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Google_Assistant (дата звернення: 14.01.2025).
41. Recent Advances in Synthesis and Interaction of Speech, Text, and Visual Data / A. Kumar, N. Patel // MDPI Electronics. – 2024. – Т. 13, №9. – С. 1726.
42. Siri – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Siri (дата звернення: 14.01.2025).
43. Voice Assistant Using Artificial Intelligence / D. Thompson, E. White // SSRN Electronic Journal. – 2022. – С. 1–10.

_____ (дата)

_____ (підпис)

# ДОДАТКИ

**Додаток А**

**Технічна документація**

# AiDen

Що таке **AiDen**? Це персональний голосовий асистент, який вміє:

- підтримувати діалог з користувачем зберігаючи контекст бесіди;
- відкривати необхідні програми, такі як, браузер, Youtube, Spotify, Steam, Telegram;
- блокувати робочий стіл.

## Важливо

Цей проект розробляється як магістерська робота здобувача вищої освіти рівня магістр.

## Оновлення

[08 Грудня 2024]

- Випущено першу версію голосового асистента.

## Встановлення

1. Встановити наступні пакети:

```
pip install pyttsx3
```

```
pip install speechRecognition
```

```
pip install openai
```

2. Клонувати проект:

```
git clone https://github.com/X3NK/AiDen-VoiceAssistant
```

3. Додати токен **OpenAi** у файлі `config.py` .

4. Зайти в директорію.

5. Запустити файл `main.py`

```
python main.py
```

## Використання

1. Виконати код в будь-якому IDE.

2. Обрати відповідний мікрофон з списку. Після цього потрібно сказати слово активації "Айден", та сказати будь-яку з наступних команд:

- `Давай поговорим`
  *Дозволяє поспілкуватись з асистентом на будь-які теми.*
- `Відкрий браузер`
  *Запустить браузер та відкриє Google.*
- `Відкрий Youtube`
  *Запустить Youtube в браузері*
- `Відкрий Spotify`
  *Відкриє Spotify*
- `Відкрий Steam`
  *Відкриє Steam*
- `Відкрий Telegram`
  *Відкриє Telegram*
- `Заблокуй робочий стіл`
  *Заблокує робочий стіл користувача*

Команди можна говорити з невеликими відхиленнями від точності команди.

## Автор

Усачик Денис Костянтинович студент KMI-23-1

**Додаток Б**

**Огляд комерційних голосових асистентів**

3. ***Amazon Alexa*** — це голосовий асистент, розроблений компанією Amazon, інтегрований у розумні пристрої, такі як Amazon Echo, Fire TV, а також сторонні IoT-пристрої. Alexa спеціалізується на управлінні розумним будинком, автоматизації завдань і доступі до сервісів Amazon. Підтримує обмежену кількість мов; українська не підтримується.

   ***Переваги Amazon Alexa:***

   - Alexa підтримує тисячі додаткових функцій, які можна додати для персоналізації асистента.
   - Асистент підтримує велику кількість IoT-пристроїв, що робить його ідеальним для автоматизації побутових завдань.
   - Легкий доступ до Amazon Music, Audible, а також здійснення покупок через Amazon.
   - Alexa працює не тільки на пристроях Amazon, але й на розумних телевізорах, колонках та інших гаджетах.
   - Інтуїтивне управління за допомогою голосових команд.

   ***Недоліки Amazon Alexa:***

   - Alexa не підтримує українську мову, що значно обмежує її використання в україномовному середовищі.
   - Зберігання та аналіз голосових даних на серверах Amazon може викликати занепокоєння щодо приватності.
   - Більшість функцій Alexa працює лише за наявності стабільного підключення до мережі.
   - У порівнянні з Google Assistant, Alexa менш ефективна у розумінні складних або контекстуальних запитів.
   - Alexa більше орієнтована на англомовний ринок та регіони з високою присутністю Amazon.

4. ***Microsoft Cortana*** — це голосовий асистент, розроблений компанією Microsoft, інтегрований у Windows 10 та деякі продукти Microsoft. В основному орієнтований на підвищення продуктивності та управління завданнями в бізнес-середовищі. Підтримує близько 13 мов; українська не підтримується.

   ***Переваги Microsoft Cortana:***

   - Cortana працює з Office 365, Outlook, Teams та іншими продуктами Microsoft, спрощуючи управління електронною поштою, календарем та завданнями.
   - Асистент пропонує функції, спрямовані на організацію робочих процесів, наприклад, планування зустрічей та нагадування.
   - Глибока інтеграція з Windows 10 дозволяє керувати системними функціями, такими як пошук файлів або налаштування.
   - Підтримка голосових команд для базових завдань.

   ***Недоліки Microsoft Cortana:***

   - Cortana поступається конкурентам (Google Assistant, Siri) у гнучкості та можливостях для повсякденного використання.
   - Основна орієнтація на корпоративних користувачів обмежує її привабливість для звичайних користувачів.
   - Підтримка на iOS та Android зведена до мінімуму або припинена.
   - У порівнянні з конкурентами, Cortana має менше інтеграцій зі сторонніми сервісами.

**Додаток В**

**Огляд голосових асистентів з відкритим вихідним кодом**

3. ***Jasper*** — це відкритий голосовий асистент, розроблений для роботи на пристроях із обмеженими ресурсами, таких як Raspberry Pi. Проєкт орієнтований на простоту використання, конфіденційність та можливість працювати без постійного підключення до інтернету. Ліцензія: MIT License

***Переваги Jasper:***

- Можливість кастомізації та адаптації під індивідуальні потреби користувача.
- Забезпечує високий рівень конфіденційності та зменшує залежність від хмарних сервісів.
- Jasper може бути встановлений на Raspberry Pi, що робить його доступним для ентузіастів.
- Призначений для швидкого запуску базової функціональності без складної конфігурації.
- Можливість інтеграції з додатковими сервісами та розширення функціональності через розробку власних модулів.

***Недоліки Jasper:***

- Обмежена функціональність: У порівнянні з комерційними асистентами, Jasper має менше "з коробки" можливостей.
- Українська мова не підтримується за замовчуванням і потребує додаткових зусиль для інтеграції.
- У порівнянні з іншими відкритими проєктами, активність спільноти та кількість оновлень є обмеженими.
- Асистент може мати проблеми з обробкою багатоконтекстних або складних запитів через базові алгоритми NLP.
- Підтримка та розвиток проекту здебільшого базуються на внесках спільноти, що може впливати на стабільність і масштабованість.

4. ***Rhasspy*** — це відкритий голосовий асистент із відкритим вихідним кодом, орієнтований на локальну обробку даних і підтримку роботи без постійного підключення до інтернету. Асистент створений для інтеграції з системами розумного дому, такими як Home Assistant. Підтримує платформи Linux, Docker, Raspberry Pi. Ліцензія: MIT License

   ***Переваги Rhasspy:***

- Забезпечує високий рівень конфіденційності, оскільки всі дані обробляються локально на пристрої.
- Можливість вибору та налаштування компонентів для ASR (розпізнавання мови), NLU (обробка природної мови), TTS (синтез мовлення).
- Підтримує платформи, такі як Home Assistant, Node-RED, та інші IoT-рішення.
- Можливість додавання української та інших мов через налаштування спільноти.
- Дає змогу адаптувати асистента під специфічні завдання та вимоги.

   ***Недоліки Rhasspy:***

- Потребує технічних знань для встановлення, конфігурації та оптимізації роботи.
- Відносно невелика кількість користувачів, що впливає на кількість готових рішень та швидкість оновлень.
- Потребує додаткових зусиль для впровадження української мови.
- У порівнянні з комерційними асистентами, Rhasspy має обмежений набір функцій "з коробки".
- Може вимагати пристроїв із достатньою потужністю для стабільної роботи в локальному режимі.

5. ***SUSI.AI*** — це відкритий голосовий асистент, розроблений організацією FOSSASIA. Він надає платформу для створення та

інтеграції інтелектуальних чат-ботів і голосових помічників у різні додатки та пристрої. Ліцензія: Apache License 2.0

***Переваги SUSI.AI:***

- SUSI.AI дозволяє розробникам переглядати, змінювати та вдосконалювати код відповідно до своїх потреб, сприяючи інноваціям та гнучкості.
- Підтримує інтеграцію з різними платформами, включаючи веб-додатки, мобільні пристрої та апаратні рішення, що забезпечує широку сумісність.
- Проєкт підтримується спільнотою розробників, які постійно працюють над покращенням функціональності та додаванням нових можливостей.
- Можливість налаштування обробки даних на локальних серверах, що підвищує рівень конфіденційності та безпеки.
- SUSI.AI підтримує різні мови, що робить його доступним для користувачів з різних регіонів.

***Недоліки SUSI.AI:***

- SUSI.AI може не мати такого ж рівня складності та кількості функцій, як провідні комерційні голосові асистенти.
- Для налаштування та кастомізації асистента потрібні певні технічні навички, що може бути складним для користувачів без відповідного досвіду.
- У порівнянні з популярними комерційними рішеннями, SUSI.AI має меншу базу користувачів, що може впливати на швидкість виявлення та виправлення помилок.
- Хоча SUSI.AI підтримує інтеграцію з різними платформами, його сумісність з деякими апаратними пристроями може бути обмеженою.

- Як проєкт з відкритим кодом, SUSI.AI не має офіційної служби підтримки, що може ускладнити вирішення технічних проблем для деяких користувачів.