

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерних наук і технологій
(повне найменування факультету)

Комп'ютерні системи та мережі
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалаврський
(ступінь вищої освіти)

на тему: РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ
ПЕРСОНАЛІЗОВАНОЇ ПІДТРИМКИ КОРИСТУВАЧА КОМП'ЮТЕРА

Виконав(ла): студент(ка) 4 курсу,
групи КНТ-522

Спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ТРОХИМЧУК В.Є.
(ПРИЗВИЩЕ та ініціали)

Керівник ТЯГУНОВА М.Ю.
(ПРИЗВИЩЕ та ініціали)

Рецензент ЩЕРБАКОВ В.І.
(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра комп'ютерних систем та мереж
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри КУДЕРМЕТОВ Р.К.

«14» квітня 2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ТРОХИМЧУКА Віталія Євгеновича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка інтелектуальної системи персоналізованої підтримки користувача комп'ютера

керівник проєкту (роботи) к.т.н., доцент, ТЯГУНОВА М.Ю.

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 08 » квітня 2026 року № 151

2. Строк подання студентом проєкту (роботи) 01.06.2026 р.

3. Вихідні дані до проєкту (роботи) опис предметної області, специфікація функціональних і нефункціональних вимог, технології розпізнавання мовлення та синтезу голосу (модулі speech recognition, edge tts), використання OpenAI API для генерації відповідей; середовище розробки Python.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Аналіз предметної області;

2) Проєктування системи;

3) Реалізація системи;

4) Результати роботи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ТЯГУНОВА М.Ю.		
Нормоконтроль	ГОЛУБ Т.В.		

7. Дата видачі завдання «14» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області	до 18.04.2026	
2	Визначення та аналіз вимог до програмного забезпечення	до 22.04.2026	
3	Розробка специфікації вимог SRS	до 24.04.2026	
4	Розробка діаграми варіантів використання	до 28.04.2026	
5	Розробка діаграми послідовності	до 01.05.2026	
6	Розробка описів прецедентів	до 05.05.2026	
7	Проектування архітектури програмного забезпечення	до 12.05.2026	
8	Проектування інтерфейсу користувача	до 22.05.2026	
9	Оформлення пояснювальної записки	до 25.05.2026	
10	Проходження нормоконтролю	до 01.06.2026	
11	Перевірка на наявність академічного плагіату	до 03.06.2026	
12	Проходження рецензування	до 10.06.2026	

Студент(ка)

(підпис)

Віталій ТРОХИМЧУК

(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Марія ТЯГУНОВА

(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
55 с., 6 табл., 14 рис., 18 джерел.

ІНТЕЛЕКТУАЛЬНА СИСТЕМА, ГОЛОСОВИЙ АСИСТЕНТ,
ПЕРСОНАЛІЗАЦІЯ, РОЗПІЗНАВАННЯ МОВЛЕННЯ, СИНТЕЗ МОВЛЕННЯ,
LLM, NLU, TTS, ASR, JSON

Об'єкт розробки – інтелектуальна система персоналізованої підтримки користувача персонального комп'ютера.

Мета роботи – спробити керування персональним комп'ютером за рахунок створення прототипу голосового асистента.

У ході роботи проаналізовано технічні вимоги та існуючі підходи до побудови голосових асистентів. Запропоновано гібридну архітектуру: правилоорієнтований маршрутизатор намірів для частих команд і використання LLM як універсального обробника вільних запитів. Спроектовано модульну систему: розпізнавання мовлення (ASR), модуль NLU/Router, набір інструментів (час, курс валют, генерація випадкових чисел, міні-гра), інтеграція з LLM, синтез мовлення (TTS), підсистема персоналізації (user_profile.json), журналювання та конфігурація команд (speech.json).

У результаті виконання роботи створено працездатний прототип інтелектуальної системи персоналізованої підтримки користувача ПК, який може бути розширений інструментами керування застосунками, нагадуваннями, інтеграцією з календарями та офлайн-режимом ASR/TTS. Робота має практичну цінність для підвищення ефективності взаємодії з комп'ютером і може бути основою для подальших досліджень у сфері діалогових систем та персоналізації.

ЗМІСТ

Скорочення та умовні позначки	6
Вступ.....	7
1 Аналіз предметної області.....	8
1.1 Аналіз цільової аудиторії	8
1.2 Аналіз голосових оповіщувачів	10
1.3 Визначення та аналіз вимог до програмного забезпечення.....	13
1.4 Розробка специфікації вимог SRS	15
2. Проєктування системи	19
2.1 Загальна архітектура системи	19
2.2 Моделювання програмного забезпечення	21
2.3 Функціонування системи.....	22
2.4 Розгорнуті описи прецедентів.....	23
2.5 Апаратне забезпечення системи	24
3 Реалізація системи.....	25
3.1 Обґрунтування вибору технологій	25
3.2 Реалізація логічної структури системи	26
3.3 Реалізація основних модулів системи.....	28
4 Результати роботи	41
4.1 Інтерфейс користувача.....	41
4.2 Масштабованість та можливості розвитку	45
Висновки	47
Перелік джерел	49
Додаток А Слайди презентації.....	51

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

- AI – Штучний інтелект (Artificial Intelligence)
- ASR – Розпізнавання мовлення (Automatic Speech Recognition)
- TTS – Синтез мовлення (Text-To-Speech)
- LLM – Велика мовна модель (Large Language Model)
- NLU – Розуміння природної мови (Natural Language Understanding)
- GUI – Графічний інтерфейс користувача (Graphical User Interface)
- API – Програмний інтерфейс застосунків (Application Programming Interface)
- ПК – Персональний комп'ютер
- ПЗ – Програмне забезпечення
- БД – База даних
- ТЗ – Технічне завдання
- WER – Частка помилок розпізнавання слів (Word Error Rate)

ВСТУП

Стрімкий розвиток штучного інтелекту і голосових інтерфейсів змінює звичну взаємодію з персональним комп'ютером. Де раніше потрібні були меню, ярлики й комбінації клавіш, сьогодні достатньо короткої фрази. Однак за простотою для користувача стоять складні інженерні виклики: надійно розпізнати українське мовлення в умовах шуму, швидко визначити намір, зберегти приватність даних і відповісти лаконічно та по суті.

Ця робота присвячена розробленню інтелектуальної системи персоналізованої підтримки користувача ПК з голосовою взаємодією. Запропоновано гібридний підхід: для типових команд — словникові правила з мінімальною затримкою, для “вільних” запитів — велика мовна модель, яка формує короткі й доречні відповіді. Система поєднує модулі ASR і TTS, маршрутизатор намірів, набір локальних інструментів (час, курс валют, випадкові числа, міні-гра), а також підсистему персоналізації на основі профілю користувача й короткої контекстної пам'яті.

Мета роботи – спробити керування персональним комп'ютером за рахунок створення прототипу голосового асистента.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати предметну область та аналогічні інструменти;
- визначити вимоги до системи;
- спроектувати архітектуру системи;
- реалізувати прототип системи;
- проаналізувати отримані результати.

Практичний результат – більш зручна й швидка робота за комп'ютером за рахунок природної мови, а також технологічна основа для подальшого розвитку: офлайн-режиму, розширення інструментів і керування застосунками.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз цільової аудиторії

Персональний комп'ютер є робочим інструментом, у якому користувач постійно перемикається між завданнями, вікнами та програмами. Це створює додаткові витрати уваги на навігацію і пошук потрібних дій. Голосова взаємодія знімає частину цього навантаження: коротка команда дозволяє отримати довідкову інформацію або виконати просту дію без відволікання від поточного контексту [1].

Для настільного середовища важливо поєднати швидкість відгуку, коректне розуміння наміру та акуратне озвучення результату [5], [16]. Саме тому доцільною є гібридна організація діалогу: часті команди обробляються за правилом і повертають відповідь практично миттєво, тоді як відкриті запити передаються мовній моделі, яка здатна сформулювати лаконічну і доречну репліку [2].

Типові користувачі — студенти, офісні працівники, інженери й розробники, які часто виконують дрібні запити: дізнатися час, перевірити курс валюти, отримати випадкове число, швидко поставити пояснювальне питання. Вони очікують мінімальної затримки відповіді, зрозумілої мови та відсутності зайвої “балакучості”. Сценарії легко масштабуються: від побутових запитів до керування застосунками, створення нотаток і нагадувань, читання локальних файлів із формуванням короткого резюме. Звідси випливають вимоги до інтерфейсу: активація за ключовим словом, підтримка коротких фраз українською, стислі відповіді в один-два речення, коректна пауза та інтонація під час озвучення, а також можливість налаштувати голос і тон відповідей. Персоналізація підсилює досвід: профіль з іменем, часовим поясом або містом, бажаним голосом, а також коротка історія діалогу, що дозволяє зберігати контекст і уникати повторень.

Система повинна забезпечити повний цикл “мікрофон → намір → дія → голосова відповідь”. Для цього використовуються: модуль розпізнавання

мовлення для коротких фрагментів до п'яти секунд; маршрутизатор намірів, який спочатку перевіряє словник правил, а за відсутності збігу делегує обробку мовній моделі; набір локальних інструментів (час, курс валюти, генерація випадкових чисел, міні-гра); синтез мовлення для природного озвучення. Конфігурація команд і шаблонів відповідей зберігається у легкому текстовому форматі, що спрощує розширення без зміни коду. Журналювання подій необхідне для діагностики та подальшої оцінки якості: фіксуються розпізнаний текст, визначений намір, часові мітки та затримки на кожному етапі. Нефункціональні вимоги зосереджені на швидкодії й надійності: очікується, що локальні наміри відпрацьовують до двох секунд, а запити з участю мовної моделі вкладаються у чотири секунди. Важливими є стійкість до побутового шуму та коректна обробка збоїв мережі із повторними спробами і тайм-аутами [3].

У предметній області виділяються кілька ключових об'єктів даних. Профіль користувача зберігає атрибути персоналізації: ім'я для звертання, параметри голосу і тону, часовий пояс або місто, інші вподобання. Словник команд описує фрази-входи, допустимі варіанти формулювань, шаблони коротких відповідей і зв'язок із відповідними функціями. Логи телеметрії містять часові мітки, текст розпізнавання, обраний намір, виміряні затримки й коди помилок. Тимчасові аудіофайли створюються під час синтезу і мають негайно видалятися після відтворення. Система взаємодіє як з локальними ресурсами операційної системи (мікрофон, динаміки, системний час), так і з зовнішніми сервісами, наприклад банківськими API для отримання курсу валюти або хмарними службами розпізнавання і генерації відповіді. Вимоги до безпеки включають зберігання секретних ключів у середовищі виконання, мінімізацію і, за потреби, анонімізацію журналів, а також контроль життєвого циклу тимчасових файлів для запобігання витокам.

Основні ризики пов'язані з варіативністю української мови, яка знижує ефективність чисто правилкових підходів; із якістю мікрофонів і акустикою приміщення, що впливають на точність розпізнавання; із залежністю від зовнішніх сервісів, яка призводить до затримок і збоїв. Вони пом'якшуються

гібридним маршрутизатором намірів, розширенням словника, підтвердженням у неоднозначних випадках, ретраями мережевих запитів і локальним кешуванням простих відповідей. Ефективність системи доцільно вимірювати точністю визначення намірів на тестовому корпусі, середнім і медіанним часом відповіді окремо для локальних і “відкритих” запитів, часткою збоїв розпізнавання та мережі, а також суб’єктивною оцінкою зручності взаємодії користувачами. Такий набір показників дозволяє об’єктивно порівнювати альтернативні реалізації та відслідковувати прогрес під час ітерацій розробки.

1.2 Аналіз голосових оповіщувачів

Під час проектування інтелектуальної системи персоналізованої підтримки користувача було проведено аналіз сучасних голосових асистентів, які вже активно застосовуються у побутових, мобільних та робочих середовищах. Найпоширенішими аналогами є Siri від Apple, Google Assistant, Amazon Alexa та Cortana/Microsoft Copilot. Ці системи реалізують голосову взаємодію, підтримують виконання типових команд, пошук інформації, керування сервісами та частково персоналізовані відповіді, однак суттєво відрізняються за архітектурою, екосистемою, рівнем локальної обробки та можливостями розширення [4].

Siri є прикладом асистента, глибоко інтегрованого в екосистему Apple. Вона забезпечує стабільну роботу на пристроях одного виробника, підтримує системні функції, календарі, повідомлення, автоматизації та швидкі команди [5]. Проте Siri значною мірою залежить від закритої інфраструктури Apple, що обмежує її використання на сторонніх пристроях. Крім того, Siri орієнтована на мобільні сценарії, а не на повноцінну роботу в середовищі персонального комп’ютера, що робить її менш придатною для задач, які потребують локальної інтеграції з файловою системою чи сторонніми програмами.

Google Assistant вирізняється високою точністю розпізнавання природної мови та широкими можливостями пошуку інформації завдяки інтеграції з сервісами Google. Його сильна сторона — здатність обробляти складні, розмовні запити та підтримувати широкий спектр пристроїв, включно з Android, ChromeOS та IoT-системами. Водночас Google Assistant майже повністю залежить від хмарної інфраструктури, що унеможлиблює його повноцінну роботу без інтернету. Це створює затримки у відповідях та обмежує використання в умовах, де потрібна локальна обробка команд [5].

Amazon Alexa традиційно вважається одним із найкращих рішень для керування розумним будинком. Вона підтримує величезну кількість IoT-пристроїв, має розвинену систему навичок (skills) та автоматизацій, а також зручні інструменти для розробників. Проте Alexa орієнтована переважно на побутові сценарії та взаємодію з домашніми пристроями, а не на роботу з персональним комп'ютером. Її можливості у сфері локального керування ПК є обмеженими, а більшість функцій також залежить від хмарних сервісів.

Cortana, а в новіших сценаріях Microsoft Copilot, історично були пов'язані з екосистемою Windows і орієнтовані на підвищення продуктивності користувача. Cortana мала можливості голосового керування системою, пошуку файлів, роботи з календарем і нагадуваннями. Проте з часом концепція Cortana була переглянута, і Microsoft змістила фокус на ширші AI-інструменти, інтегровані у Windows 11 та Microsoft 365. Copilot сьогодні виконує роль універсального помічника на основі мовних моделей, але не є класичним голосовим асистентом у традиційному розумінні [6].

Порівняння цих систем показує, що кожен комерційний асистент оптимізований під власну екосистему: Apple — під свої пристрої, Google — під інформаційні сервіси, Amazon — під розумний дім, Microsoft — під офісні та робочі сценарії. Жоден із них не орієнтований на створення легкого, локального, персоналізованого голосового асистента для персонального комп'ютера, який може працювати без глибокої інтеграції у закриту екосистему.

Окремо варто зазначити, що більшість комерційних асистентів використовують складні хмарні моделі для обробки запитів, що створює затримки, залежність від інтернету та обмеження щодо приватності. У той же час локальні сценарії, такі як отримання часу, запуск програм, робота з файлами чи виконання простих команд, можуть бути реалізовані значно швидше та ефективніше без звернення до зовнішніх сервісів.

Таким чином, аналіз аналогічних систем підтверджує доцільність створення спеціалізованого голосового асистента для ПК, який поєднує локальне виконання типових команд із можливістю обробки відкритих запитів за допомогою мовної моделі. Такий підхід дозволяє забезпечити швидкість, автономність, персоналізацію та гнучкість, яких не вистачає існуючим комерційним рішенням. Це робить розробку власного асистента актуальною та обґрунтованою в контексті поставленої задачі.

Таблиця 1.1 – порівняльна голосових асистентів

Характеристика	Siri	Google Assistant	Amazon Alexa	Cortana / Microsoft Copilot
Основна екосистема	Apple	Google	Amazon	Microsoft
Типова сфера використання	Мобільні пристрої	Пошук інформації, мобільні сервіси	Розумний дім	Продуктивність, робочі сценарії
Локальна робота без інтернету	Обмежена	Майже відсутня	Мінімальна	Частково можлива
Глибина інтеграції з ПК	Низька	Середня	Низька	Висока (Windows)
Гнучкість розширення	Низька	Середня	Висока (skills)	Обмежена

Продовження таблиці 1.1

Характеристика	Siri	Google Assistant	Amazon Alexa	Cortana / Microsoft Copilot
Підтримка сторонніх сервісів	Обмежена	Висока	Дуже висока	Середня
Природність мовлення	Висока	Висока	Середня	Середня
Орієнтація на локальні команди	Низька	Низька	Низька	Середня
Залежність від хмари	Висока	Дуже висока	Дуже висока	Висока

1.3 Визначення та аналіз вимог до програмного забезпечення

Розробка інтелектуального голосового асистента потребує чіткого формулювання вимог, які визначають функціональні можливості системи, її поведінку, обмеження та критерії якості. Вимоги формуються на основі аналізу сценаріїв використання, характеристик цільової аудиторії та технічних умов роботи у настільному середовищі [7].

Першочерговим завданням є визначення рівня деталізації вимог. Для цього застосовується дедуктивний підхід: від загальних очікувань користувача до конкретних функцій, параметрів продуктивності та обмежень. Вимоги поділяються на функціональні, нефункціональні та системні [8].

Функціональні вимоги описують, які саме дії має виконувати система: розпізнавати голосові команди, визначати намір, виконувати локальні операції (отримання часу, курс валют, генерація випадкових чисел), взаємодіяти з мовною моделлю для відкритих запитів, синтезувати голосові відповіді та підтримувати персоналізацію.

Нефункціональні вимоги визначають якість роботи системи: швидкодію, точність розпізнавання, стійкість до шуму, надійність мережевих запитів,

коректне видалення тимчасових файлів, а також вимоги до безпеки — зберігання ключів, мінімізацію журналів та захист персональних даних [9].

Системні вимоги описують середовище виконання: операційну систему, доступ до мікрофона та динаміків, наявність інтернет-з'єднання для роботи з API, а також обмеження на тривалість аудіофрагментів і затримку відповіді.

На основі зібраних вимог формується таблиця, що структурує очікування різних типів користувачів.

Таблиця 1.2 – Вимоги до програмного забезпечення

№	Користувач	Вимоги користувача
1	Студент	Отримання коротких відповідей, коректне розпізнавання української мови, можливість ставити довідкові питання
2	Офісний працівник	Швидкий доступ до часу, нагадувань, коротких пояснень; мінімальна затримка відповіді
3	Розробник	Гнучкість налаштувань, можливість розширення словника команд, стабільність API-запитів
4	Універсальний користувач	Природне озвучення, стійкість до шуму, відсутність зайвих фраз, персоналізація голосу

Під час розробки інтелектуальної системи персоналізованої підтримки користувача було сформовано деякі вимоги до її функціонування. Головним є повноцінна підтримка саме української мови, як для розпізнавання запитів користувача так і для синтезу голосових відповідей. Для того щоб асистент постійно не підхвачував слова які говорить користувач передбачено ключове слово для активації. Лише після вимовляння цього слова система переходить в режим приймання слів та їх обробку. Для швидкої роботи та зменшення навантаження на модулі розпізнавання тривалість голосових фраз користувача,

які йдуть на обробку, обмежуються п'ятьма секундами. Важливою вимогою є використання природного синтезу мовлення з коректною інтонацією та зрозумілою вимовою української мови. Це дозволяє приблизити розмову до більш живого діалогу. З архітектурного плану, повинна бути можливість розширення набору підтримуваних команд без внесення змін до програмного коду. Для цього словник команд зберігається у зовнішніх конфігураційних файлах, що спрощує подальший розвиток та супровід системи. Оскільки частина функціональності залежить від зовнішніх вебсервісів, система повинна коректно обробляти помилки мережевого з'єднання та виконувати автоматичні повторні спроби надсилання запитів у разі тимчасових збоїв. Для ефективного використання ресурсів комп'ютера після завершення відтворення синтезованого мовлення система повинна автоматично видаляти створені тимчасові аудіофайли. Цей підхід дозволяє уникнути засмічення непотрібними даними.

1.4 Розробка специфікації вимог SRS

Специфікація вимог SRS визначає повний набір функцій, характеристик та обмежень системи. Вона є основою для подальшого моделювання, проєктування та тестування[10].

Основним призначенням системи є забезпечення голосової взаємодії користувача з персональним комп'ютером українською мовою. Система повинна здійснювати розпізнавання голосових команд, аналізувати зміст отриманих повідомлень та визначати намір користувача на основі заздалегідь сформованих словникових правил. У разі виявлення команд, що належать до локального функціоналу, система виконує відповідну дію без залучення зовнішніх сервісів. До таких функцій належать отримання поточного часу, визначення курсу валют, генерація випадкових чисел та запуск мініігор. Якщо запит користувача виходить за межі визначених команд, він передається мовній моделі для формування змістовної відповіді природною мовою. Отриманий результат озвучується за

допомогою модуля синтезу мовлення. Крім того, система повинна підтримувати персоналізацію голосу та стилю відповідей відповідно до налаштувань користувача.

Для забезпечення своєї роботи програмний продукт взаємодіє з рядом зовнішніх інтерфейсів. Джерелом вхідних даних є мікрофон, через який надходить аудіопотік із голосовими командами користувача. Відтворення сформованих відповідей здійснюється через акустичні пристрої комп'ютера. Для генерації відповідей на складні запити використовується API OpenAI, а отримання актуальної інформації про валютні курси забезпечується через відповідні банківські API [11]. Також система взаємодіє з локальними ресурсами операційної системи для роботи з файлами, отримання поточного часу та виконання інших службових операцій.

Важливими характеристиками системи є швидкодія та зручність використання. Час обробки локальних команд не повинен перевищувати двох секунд, а відповідь, сформована за допомогою мовної моделі, має генеруватися протягом не більше чотирьох секунд. Максимальна тривалість голосового фрагмента, що надходить на розпізнавання, становить п'ять секунд. Якість розпізнавання мовлення залежить від рівня навколишнього шуму та характеристик обладнання користувача, однак система оптимізується для роботи в типових побутових умовах.

До основних нефункціональних вимог належать мобільність, зручність супроводу, захищеність та масштабованість. Мобільність забезпечується можливістю перенесення програмного забезпечення на інші персональні комп'ютери без суттєвих змін конфігурації [12]. Для спрощення супроводу та розширення функціональності конфігурація голосових команд зберігається у форматі JSON, що дозволяє додавати нові команди без модифікації програмного коду. Захист конфіденційних даних реалізується шляхом зберігання ключів доступу до зовнішніх сервісів у середовищі виконання. Масштабованість системи досягається завдяки модульній архітектурі, яка дозволяє інтегрувати

нові функції та сервіси без значних змін існуючої структури програмного забезпечення.

Під час розробки системи враховуються певні проєктні обмеження. Основною мовою програмування обрано Python через широкий набір бібліотек для роботи з технологіями розпізнавання мовлення, синтезу голосу та штучного інтелекту. Частина функціональності залежить від зовнішніх API, тому для повноцінної роботи системи необхідне стабільне підключення до мережі Інтернет. Крім того, вимоги до роботи в режимі реального часу накладають обмеження на продуктивність окремих компонентів і потребують оптимізації процесів обробки голосових даних та взаємодії із зовнішніми сервісами.

У центрі взаємодії перебуває актор “Користувач ПК”, який здійснює голосове звертання до системи. Сценарій починається з активації асистента, що включає розпізнавання ключового слова у вимовленій фразі. Після активації користувач формулює запит. Якщо це прохання назвати поточний час, система визначає відповідний намір і повертає коротку відповідь, яку одразу озвучує. Коли користувача цікавить курс валюти, після визначення наміру виконується звернення до банківського сервісу, отримується актуальне значення, формується стисла фраза та відтворюється голосом. Коли користувач просить згенерувати випадкове число, система локально формує результат і озвучує його. Якщо користувач пропонує зіграти, запускається діалоговий цикл мінігри з підказками “більше” або “менше” до моменту вгадування або завершення. Коли надходить відкрите запитання за ключовим словом “допоможи”, не передбачене у словнику команд, визначення наміру завершується делегуванням до мовної моделі; результатом стає коротка змістовна відповідь, яка також озвучується. Налаштування голосу та тону виконуються окремим сценарієм персоналізації, де користувач змінює параметри профілю, після чого всі наступні відповіді програються з урахуванням нових уподобань. У цих сценаріях задіяні зовнішні актори: сервіс мовної моделі під час відкритих запитів, банківський сервіс під час отримання валютних курсів, а також аудіопристрої операційної системи, через які здійснюється приймання мовлення та відтворення відповіді.

1.4.1 Словник термінів

ASR (Automatic Speech Recognition) — підсистема автоматичного розпізнавання мовлення, яка перетворює аудіосигнал з мікрофона на текст українською мовою. Є першим кроком у конвеєрі обробки команд.

TTS (Text-to-Speech) — підсистема синтезу мовлення, що перетворює згенеровану текстову відповідь на природне голосове озвучення та відтворює її через аудіопристрої ПК.

Намір (intent) — семантична інтерпретація висловлювання користувача як дії, яку необхідно виконати системі. Приклади намірів: дізнатися час, отримати курс валюти, згенерувати число, розпочати міні-гру, поставити відкрите запитання.

Плейсхолдер (placeholder) — маркер у шаблоні відповіді, який під час формування підмінюється фактичним значенням, отриманим від функції. Приклади: [час], [курс], [число], [текст].

Wake word (ключове слово активації) — спеціальне слово або фраза, яка вказує системі, що наступна репліка адресована асистенту й підлягає обробці. У проєкті використовується форма звертання “Аня”.

Маршрутизатор намірів (router) — компонент, який на основі розпізнаного тексту визначає намір користувача. Спочатку застосовує словникові правила зі `speech.json`, а за їх відсутності делегує обробку мовній моделі.

Локальні інструменти (handlers) — функції, які виконуються без звернення до мовної моделі: отримання часу, генерація випадкового числа, міні-гра, звернення до зовнішніх довідкових API (наприклад, курси валют).

Профіль користувача — структура з налаштуваннями персоналізації (ім’я, бажаний голос/тон відповіді, часовий пояс/місто, коротка історія діалогу), що впливає на стиль і вміст відповідей.

1.4.2 Припущення та залежності

Система працює у настільному середовищі (Windows або Linux) з доступом до мікрофона та аудіовиходу. Наявний стандартний мікрофон користувача забезпечує якість сигналу, достатню для впевненого розпізнавання коротких фраз

у типових побутових умовах. Голосові фрагменти, що подаються на ASR, тривають не більш як п'ять секунд, що відповідає сценаріям коротких команд і мінімізує затримку обробки.

Для обробки відкритих запитів та отримання зовнішніх довідкових даних необхідне стабільне підключення до Інтернету. У разі його відсутності система коректно повідомляє про неможливість виконання відповідної дії або повертає локальний результат, якщо сценарій це дозволяє (наприклад, генерація випадкового числа чи поточний системний час).

Поточна реалізація спирається на зовнішні служби розпізнавання мовлення та мовної моделі, тому час відповіді залежить від мережових затримок і доступності сервісів. Для скорочення затримок застосовується обмеження довжини фрази, а для підвищення надійності — обробка тайм-аутів і повтор однієї спроби при тимчасових збоях.

Конфігурація команд і шаблонів відповідей зберігається у файлах формату JSON, що дозволяє розширювати словник без зміни вихідного коду. Секретні ключі доступу до зовнішніх API зберігаються у змінних середовища або окремому захищеному файлі, який не публікується разом з кодом.

Система орієнтована на один активний голосовий канал взаємодії з одним користувачем на робочій станції. Багатокористувацькі сценарії, розпізнавання мовця, повністю офлайн-режим та довготривала діалогова пам'ять розглядаються як перспективні розширення й не входять до обов'язкового набору вимог базової версії.

2. ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Загальна архітектура системи

Архітектура голосового асистента побудована за модульним принципом, що забезпечує гнучкість, масштабованість та можливість незалежного розвитку

окремих компонентів. Система складається з кількох логічних рівнів, кожен з яких виконує власну функцію: розпізнавання мовлення, визначення наміру, виконання дій, генерація голосової відповіді та управління конфігурацією команд (рисунок 2.1).

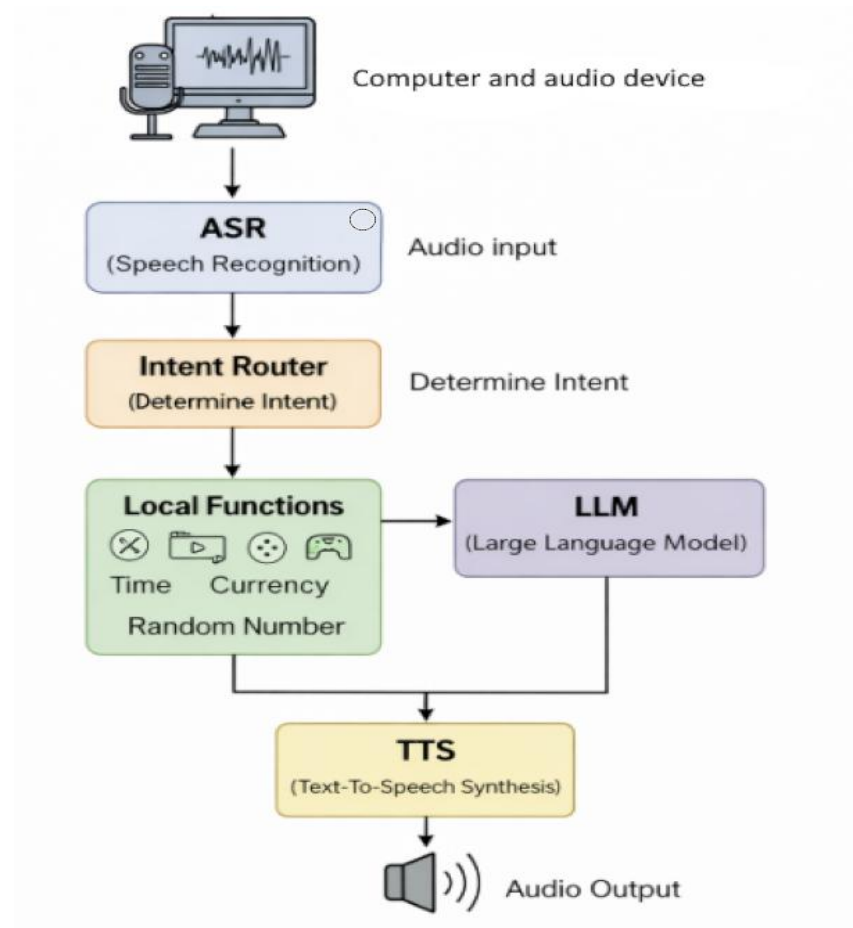


Рисунок 2.1 – Схематичне зображення архітектури системи

Робота розробленого голосового асистента починається з отримання голосової команди користувача через мікрофон. Після цього аудіосигнал передається до модуля розпізнавання мовлення, який за допомогою бібліотеки `speech_recognition` перетворює мовлення на текст. Отриманий текст аналізується системою для виявлення ключового слова активації «Аня». Наявність цього слова свідчить про те, що користувач звертається саме до голосового асистента.

Після активації система виконує обробку розпізнаної команди. Текст порівнюється зі словником команд, який зберігається у файлі `speech.json`. На основі цього порівняння визначається, яку саме дію необхідно виконати. Якщо

команда належить до набору заздалегідь визначених функцій, відбувається виклик відповідної функції з модуля `functions.py`. У випадках, коли запит є складнішим або не входить до переліку стандартних команд, він передається до мовної моделі штучного інтелекту для формування відповіді.

Після отримання результату система формує текст відповіді та передає його до модуля озвучення. Для синтезу мовлення використовується бібліотека `edge_tts`, яка генерує аудіофайл із голосовою відповіддю. Далі створений аудіофайл відтворюється через динаміки комп'ютера, що дозволяє користувачеві отримати результат у звуковому вигляді. Після завершення відтворення тимчасові файли автоматично видаляються, що забезпечує коректну роботу програми та раціональне використання ресурсів операційної системи.

2.2 Моделювання програмного забезпечення

Моделювання програмного забезпечення дозволяє формально описати поведінку системи, взаємодію користувача з голосовим асистентом та внутрішні процеси, що забезпечують роботу всіх функцій. На цьому етапі створюються моделі, які відображають логіку обробки голосових команд, визначення намірів, виконання дій та формування голосових відповідей. Моделі допомагають узгодити вимоги з архітектурою та забезпечують основу для подальшого проєктування [4].

Діаграма варіантів використання описує взаємодію користувача з системою на рівні намірів. Основним актором є користувач ПК, який взаємодіє з асистентом через голосові команди. Система виступає посередником між користувачем і набором функцій, доступних у локальних модулях або через мовну модель.

Основні варіанти використання:

- активація асистента за ключовим словом;
- отримання довідкової інформації (час, курс валют, випадкове число);

- запуск інтерактивної міні-гри;
- постановка відкритого запиту, який обробляється мовною моделлю;
- отримання голосової відповіді;
- персоналізована взаємодія (ім'я, стиль відповіді, голос).

Текстовий опис діаграми

Актор «Користувач» взаємодіє з такими прецедентами:

- «Активувати асистента»;
- «Поставити команду»;
- «Отримати відповідь»;
- «Отримати час»;
- «Отримати курс валют»;
- «Згенерувати випадкове число»;
- «Грати у міні-гру»;
- «Отримати відповідь від мовної моделі».

Прецеденти «Отримати час», «Отримати курс валют» та «Згенерувати число» є спеціалізаціями прецеденту «Виконати локальну функцію».

2.3 Функціонування системи

Функціонування системи для ноачності опишемо діаграмою послідовності. Діаграма послідовності демонструє порядок виконання операцій у системі під час обробки голосової команди. Нижче наведено два ключові сценарії: локальна команда та відкритий запит.

Сценарій 1: Локальна команда («Аня, який зараз час?»)

- Користувач вимовляє фразу;
- Мікрофон передає аудіо в модуль розпізнавання мовлення;
- speech_recognition перетворює аудіо у текст;
- Асистент перевіряє наявність ключового слова активації;

- Текст передається у маршрутизатор намірів;
- Маршрутизатор знаходить збіг у `speech.json` та визначає функцію `get_time()`;
- Функція повертає значення часу;
- Результат підставляється у шаблон відповіді;
- Модуль `edge_tts` генерує аудіофайл;
- Аудіо відтворюється користувачу.

Сценарій 2: Відкритий запит («Аня, поясни що таке штучний інтелект»)

1–5. Ті самі кроки, що й у попередньому сценарії;

- Збігу у словнику команд немає;
- Запит передається у функцію `ai()`, яка формує промпт і надсилає його до OpenAI;
- Мовна модель повертає коротку відповідь;
- Відповідь озвучується через TTS;
- Тимчасовий аудіофайл видаляється.

2.4 Розгорнуті описи прецедентів

Отримання часу. Метою цього прецеденту є надання користувачу актуального системного часу. Користувач вимовляє команду, система розпізнає текст, маршрутизатор визначає намір «час», викликається функція `get_time()`, після чого результат озвучується. Після виконання прецеденту користувач отримує коротку відповідь у голосовому форматі.

Отримання курсу валют. Прецедент передбачає отримання актуального курсу долара. Після розпізнавання команди маршрутизатор визначає намір «курс», викликається API ПриватБанку [10], отримані дані форматуються та озвучуються. У разі недоступності API система повідомляє про помилку або виконує повторну спробу.

Обробка відкритого запиту. Цей прецедент використовується, коли команда не входить до словника. Система передає текст у мовну модель [7], яка формує коротку відповідь. Асистент озвучує її через TTS. Відповідь має бути лаконічною, щоб не перевантажувати користувача.

Міні-гра. Прецедент описує інтерактивну гру, у якій асистент загадує число, а користувач намагається його вгадати. Система аналізує кожну відповідь, порівнює її із загаданим числом і надає підказки. Гра завершується при вгадуванні або за командою «кінець». Якщо користувач вимовляє не число, система просить повторити.

2.5 Апаратне забезпечення системи

Для функціонування інтелектуальної системи голосового асистента не передбачається використання спеціалізованого апаратного забезпечення, оскільки програмний продукт реалізується засобами мови програмування Python і працює на базі стандартної архітектури персонального комп'ютера. Такого асистента можливо запускати як на Windows, Linux так і на MacOS.

Мінімальна конфігурація апаратного забезпечення включає центральний процесор із тактовою частотою не нижче 2,0 ГГц, що забезпечує коректну обробку аудіоданих у режимі реального часу та виконання мережевих запитів до зовнішніх сервісів. Для стабільної роботи системи необхідно щонайменше 4 ГБ оперативної пам'яті, що дозволяє одночасно виконувати модулі розпізнавання мовлення, синтезу голосу та обробки логіки команд.

Для зберігання програмного коду, конфігураційних файлів (зокрема `speech.json`), віртуального середовища (`.venv`), а також тимчасових аудіофайлів формату MP3, необхідно мати не менше 500 МБ вільного дискового простору. Використання твердотілого накопичувача (SSD) є рекомендованим, оскільки

він забезпечує швидший доступ до файлів та скорочує затримки при роботі системи.

Обов'язковим апаратним компонентом є мікрофон, який використовується для введення голосових команд користувача. Якість розпізнавання мовлення безпосередньо залежить від його характеристик та рівня зовнішнього шуму. Для відтворення синтезованої мови використовуються динаміки або навушники, які забезпечують виведення аудіоінформації у режимі реального часу.

Оскільки частина функціональності системи реалізована через API зовнішніх сервісів (зокрема мовної моделі OpenAI та сервісів отримання додаткових даних), обов'язковою умовою є наявність стабільного підключення до мережі Інтернет зі швидкістю не нижче 1 Мбіт/с.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Обґрунтування вибору технологій

Під час розробки системи було обрано набір технологій, які найкраще відповідають вимогам до швидкодії, якості розпізнавання мовлення, природності синтезу голосу та гнучкості архітектури. Основним критерієм вибору інструментів стала можливість забезпечити стабільну роботу асистента в реальному часі, а також простоту інтеграції між окремими компонентами.

Для розпізнавання мовлення використано бібліотеку *speech_recognition*, оскільки вона забезпечує просту інтеграцію з сервісами Google Speech API та підтримує українську мову. Альтернативні рішення, такі як Vosk або Whisper, також розглядалися, однак вони потребують додаткових ресурсів або складнішої конфігурації. Обрана технологія дозволила швидко реалізувати точне розпізнавання коротких голосових команд у побутових умовах.

Для синтезу голосу застосовано *edge_tts*, що забезпечує високу якість озвучення українською мовою та природну інтонацію [8]. На відміну від gTTS

або pyttsx3, які мають обмеження у звучанні або швидкодії, edge_tts дозволяє отримувати аудіо з мінімальною затримкою, що є критичним для голосового асистента.

Генерація відповідей на відкриті запити реалізована через OpenAI API. Це рішення було обрано завдяки здатності моделі формувати короткі, змістовні та контекстно релевантні відповіді. Хоча існують альтернативи, такі як Gemini або Claude, OpenAI забезпечує найкращу якість саме для стислих відповідей, що відповідає концепції асистента, який не перевантажує користувача зайвою інформацією.

Для конфігурації команд використано формат JSON. Він є простим у редагуванні, легко читається та дозволяє розширювати словник команд без зміни програмного коду. Це робить систему гнучкою та зручною для подальшого розвитку. Альтернативні формати, такі як YAML або XML, не дають суттєвих переваг у контексті невеликих конфігураційних файлів.

Таким чином, вибір технологій був зумовлений прагненням забезпечити оптимальний баланс між якістю роботи, простотою інтеграції та можливістю подальшого масштабування системи.

3.2 Реалізація логічної структури системи

Логічна структура голосового асистента побудована за принципом послідовної обробки даних, де кожен рівень виконує чітко визначену функцію та передає результат наступному компоненту. Такий підхід дозволяє забезпечити прозорість роботи системи, спростити налагодження та підвищити гнучкість архітектури. Кожен рівень є незалежним від інших, що дає змогу змінювати або вдосконалювати окремі частини без ризику порушення загальної логіки роботи.

Додатково логічна структура передбачає чітке розмежування відповідальності між рівнями введення, обробки, виконання та виведення

результатів (таблиця 3.1). Це забезпечує стабільність роботи системи навіть у разі збільшення кількості функцій або інтеграції нових модулів.

Таблиця 3.1 – Логічні рівні системи

Рівень	Компоненти	Опис
Введення	Мікрофон, ASR	Отримання та розпізнавання аудіо
Обробка	Маршрутизатор, словник команд	Визначення наміру
Виконання	Локальні функції, OpenAI	Обробка запиту
Виведення	TTS, динаміки	Формування та озвучення відповіді

Перший рівень — введення — відповідає за захоплення голосу користувача та перетворення аудіосигналу на текст. На цьому етапі важливо забезпечити коректне розпізнавання мовлення, оскільки від точності цього процесу залежить правильність подальшої обробки. Бібліотека `speech_recognition` виконує роль інтерфейсу між аудіопотоком і сервісом розпізнавання.

Другий рівень — обробка — визначає, що саме мав на увазі користувач. Маршрутизатор порівнює розпізнаний текст зі словником команд, який містить тригери, ключові слова та відповідні функції. Якщо команда знайдена, система переходить до її виконання; якщо ні — запит передається мовній моделі.

Третій рівень — виконання — реалізує конкретну дію. Це може бути виклик локальної функції (наприклад, отримання часу або запуск гри) або звернення до зовнішнього сервісу, такого як API ПриватБанку чи OpenAI. На цьому етапі система формує текст відповіді, який буде озвучено користувачу.

Четвертий рівень — виведення — відповідає за синтез голосу та відтворення аудіо. Модуль `edge_tts` перетворює текст на природне звучання, після чого аудіофайл відтворюється через динаміки [14]. Після завершення відтворення тимчасові файли видаляються, що забезпечує оптимальне використання ресурсів.

3.3 Реалізація основних модулів системи

Модуль `assistant.py` Відповідає за ініціалізацію мікрофона, налаштування шумозаглушення, прослуховування аудіо, розпізнавання мовлення, перевірку ключового слова та виклик маршрутизатора команд. Це центральний компонент, який координує роботу всієї системи [16], [17].

Модуль `speech.json` Містить список фраз-тригерів, шаблони відповідей та прив'язку до функцій. Дозволяє розширювати систему без зміни коду.

Модуль `functions.py` Містить реалізацію всіх функцій, які може виконати асистент: отримання часу, курсів валют, генерація випадкових чисел, міні-гра, виклик OpenAI [7].

Модуль `output.py` Відповідає за синтез голосу, відтворення аудіо та видалення тимчасових файлів. Основні модулі системи та їх специфікації зазначені в таблиці 3.2.

Таблиця 3.2 – Основні модулі системи

Модуль	Призначення	Ключові функції	Технології
<code>assistant.py</code>	Головний цикл роботи асистента	Розпізнавання мовлення, активація, маршрутизація	<code>speech_recognition</code> , Python
<code>functions.py</code>	Локальні інструменти	Час, курс валют, випадкові числа, міні-гра, OpenAI	<code>requests</code> , <code>random</code> , OpenAI API
<code>output.py</code>	Синтез голосу	Генерація аудіо, відтворення, видалення файлів	<code>edge_tts</code> , <code>playsound</code> , <code>asyncio</code>
<code>speech.json</code>	Конфігурація команд	Тригери, шаблони відповідей, прив'язка до функцій	JSON

3.3.1 Головний модуль `assistant.py`

Модуль `assistant.py` є центральним компонентом системи, який забезпечує основний цикл роботи голосового асистента (лістинг 3.1). У ньому виконується

ініціалізація засобів розпізнавання мовлення, зчитування конфігурації команд, перевірка ключового слова активації та передача запиту на подальшу обробку.

Основною функцією модуля є організація безперервного прослуховування мікрофона, перетворення мовлення у текст за допомогою `speech_recognition` та виклик відповідної дії залежно від змісту команди. Якщо команда не належить до наперед визначених шаблонів, система формує стандартну відповідь про неможливість розпізнавання або передає обробку іншому модулю.

Лістинг 3.1 – Код модуля `assistant.py`

```
import speech_recognition as sr
from output import say
import json
import random
from functions import *
from key import key
import openai
openai.api_key = key

def load_speech():
    with open("speech.json", "r", encoding="utf-8") as file:
        data = json.load(file)
    return data

def speech_commands(text):
    data = load_speech()
    result = False
    for phrase in data:
        for input_words in phrase['input']:
            if input_words in text.lower():
                output = random.choice(phrase['output'])
                function_name = phrase.get("function")
                if function_name:
                    func = globals().get(function_name)
                    if func:
```

```

        output_func = func(text)
        for key in output_func.keys():
            output = output.replace(f'[{key}]',
str(output_func[key]))
            say(output)
            print(output)
            result = True
    return result

def main():
    recognizer = sr.Recognizer()

    with sr.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source, duration=1)
        print("Готовий слухати...")

        while True:
            try:
                audio = recognizer.listen(source, timeout=5,
phrase_time_limit=5)
            except sr.WaitTimeoutError:
                print("Не вдалося розпізнати звук.")
                continue

            try:
                text = recognizer.recognize_google(audio,
language='uk-UA')
                print(f"Ви сказали: {text}")
                if "аня" in text.lower():
                    result = speech_commands(text)
                    if result == False:
                        variants = [
                            "Я тебе не розумію",
                            "Вибачте не можу розібрати",
                            "Повторіть будь-ласка"
                        ]
                        say(random.choice(variants))

```

```

except sr.UnknownValueError:
    print("Не вдалося розпізнати звук.")
except sr.RequestError:
    print("Помилка запиту до Google API.")

main()

```

Модуль *assistant.py* складається з кількох логічних блоків, кожен з яких виконує окрему функцію в роботі голосового асистента. Перший блок — це імпорт бібліотек та ініціалізація зовнішніх залежностей. Тут підключаються модулі для розпізнавання мовлення, синтезу голосу, роботи з JSON-файлами, генерації випадкових значень, а також ключ доступу до OpenAI. Цей блок формує основу для подальшої роботи системи, забезпечуючи доступ до всіх необхідних інструментів.

Наступний блок — функція *load_speech()*, яка відповідає за завантаження словника команд із файлу *speech.json*. Вона відкриває файл, зчитує його вміст і повертає структуру даних, що містить усі фрази-тригери, шаблони відповідей і прив'язки до функцій. Завдяки цьому асистент може працювати з динамічним набором команд, а зміни в поведінці системи можна вносити без редагування коду.

Блок *speech_commands()* виконує роль інтелектуального маршрутизатора. Він отримує текст, сказаний користувачем, і порівнює його з усіма фразами у словнику команд. Якщо знайдено збіг, система формує відповідь, а за потреби викликає функцію з модуля *functions.py*. Цей блок також обробляє підстановки у шаблонах, замінюючи маркери на реальні значення, отримані від функцій. Саме тут відбувається логічне визначення наміру користувача.

Головний блок — це функція *main()*, яка організовує безперервне прослуховування мікрофона. Вона створює об'єкт розпізнавача, налаштовує шумозаглушення та запускає нескінченний цикл, у якому асистент слухає користувача, перетворює мовлення на текст і перевіряє наявність ключового слова «Аня». Якщо активація відбулася, текст передається на обробку. У цьому

блоці також реалізовано обробку помилок, що дозволяє системі працювати стабільно навіть у разі відсутності інтернету або низької якості звуку.

3.3.2 Модуль функціональної логіки `functions.py`

Модуль `functions.py` містить реалізацію основних функцій голосового асистента, які викликаються після визначення наміру користувача (лістинг 3.2). Тут присутні такі функції як отримання поточного часу, генерацію випадкового числа, визначення курсу валют, запуск інтерактивної гри та звернення до мовної моделі для обробки відкритих запитів.

Структура модуля побудована таким чином, щоб кожна функція повертала результат у стандартизованому вигляді, придатному для подальшого підставлення у шаблон голосової відповіді. Це забезпечує уніфікацію взаємодії між логічним рівнем системи та модулем синтезу мовлення.

Окрему роль у модулі виконує функція `ai()`, яка реалізує інтеграцію з OpenAI API та дозволяє формувати короткі відповіді на запити, що не входять до словника команд.

Лістинг 3.2 – Код модуля `functions.py`

```
import time
import random
import requests
from output import say
import speech_recognition as sr
import openai
from key import key

def get_time(text:str):

    current_time = time.localtime()
    output_time = f'{current_time.tm_hour}:{current_time.tm_min}'
    return {"час":output_time}

def get_random_number(text:str):
    return {"число": random.randint(0,100)}
```

```

def get_dollar_currency(text:str):
    result =
requests.get("https://api.privatbank.ua/p24api/pubinfo?json&exchange&coursid=5")
    result = result.json()
    return{"кypc": round(float(result[1]['sale']),2)}

def game(text:str):
    recognizer = sr.Recognizer()
    say("Добре давай зіграємо, я загадала число від 1 до 100 твоя
задача його відгадати, скажи кінець якщо захочеш закінчити гру")
    correct_number = random.randint(0,100)
    with sr.Microphone() as source:
        print('ready game')
        while True:
            try:
                audio = recognizer.listen(source, timeout=5,
phrase_time_limit=5)
            except sr.WaitTimeoutError:
                print("Не вдалося розпізнати звук.")
                continue

            try:
                text = recognizer.recognize_google(audio,
language='uk-UA')
                print(f"Ви сказали: {text}")
                if text.isdigit():
                    number_user = int(text)
                    if number_user == correct_number:
                        say("Вітаю ти вгадав")
                        break
                    elif number_user < correct_number:
                        say(f"Спробуй ще, мое число більше за
{number_user}")
                    elif number_user > correct_number:

```

```

        say(f"Спробуй ще, моє число менше за
{number_user}")
    elif "кінець" in text:
        break
    else:
        say("Скажіть тільки число, нічого зайвого")

except sr.UnknownValueError:
    print("Не вдалося розпізнати звук.")
except sr.RequestError:
    print("Помилка запиту до Google API.")

return{}

def ai(text:str):
    promt = f'''Ти маєш генерувати відповіді до голосового
помічника, без зайвого тексту, все що ти згенеруєш я буду відразу
надсилати користувачу у вигляді звуку
        давай КОРОТКІ відповіді не треба повністю розкривати тему
        Нижче надаю те що запитує користувач
        {text}
        '''
    response = openai.ChatCompletion.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "user", "content": promt}
        ]
    )
    return{"текст":response["choices"][0]["message"]['content'].st
rip()}

```

Модуль *functions.py* містить набір функцій, які реалізують конкретні можливості голосового асистента. На початку модуля імпортуються бібліотеки для роботи з часом, випадковими числами, HTTP-запитами, розпізнаванням мовлення та OpenAI API. Це створює основу для виконання як локальних, так і зовнішніх операцій.

Функція `get_time()` звертається до системного годинника, отримує поточний час і повертає його у вигляді словника. Такий формат дозволяє легко підставляти значення у шаблон відповіді. Аналогічно працює функція `get_random_number()`, яка генерує випадкове число в межах від 0 до 100. Обидві функції є прикладом простих локальних операцій, які виконуються миттєво та не потребують зовнішніх сервісів.

Блок, що відповідає за отримання курсу долара, реалізовано у функції `get_dollar_currency()`. Вона надсилає HTTP-запит до API ПриватБанку, отримує JSON-відповідь, обробляє її та повертає значення курсу. Це дозволяє асистенту працювати з актуальними даними в реальному часі, що робить систему корисною у повсякденних задачах.

Найбільш інтерактивним є блок гри, реалізований у функції `game()`. Тут асистент запускає цикл, у якому користувач намагається вгадати число. Функція слухає голосові відповіді, порівнює їх із загаданим числом і дає підказки. Вона також обробляє ситуації, коли користувач хоче завершити гру. Цей блок демонструє можливість асистента підтримувати діалог і реагувати на дії користувача в реальному часі.

Окремий блок — функція `ai()`, яка забезпечує інтеграцію з OpenAI. Вона формує промпт, надсилає його до мовної моделі та повертає коротку відповідь, оптимізовану для голосового озвучення. Цей блок дозволяє асистенту відповідати на відкриті запитання, які не входять до словника команд, що значно розширює функціональність системи.

3.3.3 Модуль синтезу мовлення `output.py`

Модуль `output.py` відповідає за перетворення текстових відповідей у звуковий формат та їх подальше відтворення (лістинг 3.3). Для синтезу мовлення використовується бібліотека `edge_tts`, яка дозволяє формувати природне українське озвучення. За роль голосу для асистента було використано жіночу версію українського мовлення `uk-UA-PolinaNeural`.

У модулі реалізовано генерацію тимчасового аудіофайлу, його відтворення за допомогою `playsound` та автоматичне видалення після завершення програвання.

Лістинг 3.3 – Код модуля `output.py`

```
import asyncio
import edge_tts
import os
import uuid
from playsound import playsound

VOICE = "uk-UA-PolinaNeural"

async def generate_audio(text, filename):
    communicate = edge_tts.Communicate(text, VOICE)
    await communicate.save(filename)

def say(text):
    # Унікальне ім'я файлу, щоб нічого не блокувалось
    filename = f"{uuid.uuid4()}.mp3"

    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    loop.run_until_complete(generate_audio(text, filename))
    loop.close()

    playsound(filename)

    # Видаляємо файл після програвання
    try:
        os.remove(filename)
    except PermissionError:
        pass
```

Модуль *output.py* складається з двох ключових блоків. Перший — асинхронна функція *generate_audio()*, яка за допомогою бібліотеки *edge_tts* генерує аудіофайл із тексту. Вона створює об'єкт синтезатора, передає йому текст і зберігає результат у тимчасовий файл. Використання асинхронності дозволяє уникнути блокування основного потоку програми.

Другий блок — функція *say()*, яка викликається з інших модулів. Вона створює унікальне ім'я файлу, запускає асинхронний синтез голосу, відтворює аудіо через *playsound*, а після завершення видаляє файл. Така структура забезпечує швидке озвучення відповідей і запобігає накопиченню зайвих файлів у системі. Цей блок є завершальним етапом взаємодії з користувачем, перетворюючи текстові відповіді на природне голосове озвучення.

3.3.4 Конфігураційні файли *key.py* та *speech.json*

Для забезпечення гнучкості та зручності супроводу в системі використано окремі конфігураційні файли. Файл *key.py* призначений для зберігання ключа доступу до зовнішнього сервісу генерації відповідей (лістинг 3.4). Файл *speech.json* містить набір голосових команд, ключових фраз, шаблонів відповідей та прив'язку до відповідних функцій системи (лістинг 3.5).

Використання окремого конфігураційного файлу *speech.json* дозволяє розширювати словник команд без внесення змін у програмний код, що підвищує масштабованість, спрощує модернізацію системи та зменшує нагромадження головного коду.

Лістинг 3.4 – Код файлу *key.py*

```
key = "тут потрібно ввести свій ключ для користування штучним  
інтелектом"
```

Лістинг 3.5 – Код файлу *speech.json*

```
[  
  {  
    "input": ["привіт", "добрий день", "вітаю", "добрий  
вечір", "добрий ранок"],
```

```

        "output":["привіт","привіт-привіт","і вам привіт","рада
вас бачити","добрий день"]
    },
    {
        "input":["як ти","як справи","як самопочуття","що ти"],
        "output":["все круто","все чікі пукі","поганенько","я
добре","все погано"]
    },
    {
        "input":["який зараз час","котра година","підскажи час"],
        "output":["зараз час [час]","наразі [час]","вже [час]"],
        "function": "get_time"
    },
    {
        "input":["рандомне число"],
        "output":["ось вам випадкове число [число]","тримайте
число [число]","звісно ось число [число]"],
        "function": "get_random_number"
    },
    {
        "input":["курс долара"],
        "output":["ви віддасте [курс] рівень за один долар"],
        "function": "get_dollar_currency"
    },
    {
        "input":["зіграємо"],
        "output":["круто зіграли"],
        "function": "game"
    },
    {
        "input":["допоможи"],
        "output":["[текст]"],
        "function": "ai"
    }
]

```

Файл *key.py* складається з одного блоку — змінної, у якій зберігається ключ доступу до OpenAI. Це дозволяє відокремити конфіденційні дані від основного коду, підвищуючи безпеку та спрощуючи розгортання системи.

Файл *speech.json* містить набір блоків, кожен з яких описує окрему команду. У кожному блоці є список фраз-тригерів, варіанти відповідей та, за потреби, назва функції, яку потрібно викликати. Така структура дозволяє легко розширювати словник команд, додаючи нові блоки без зміни програмного коду. Це робить систему гнучкою, масштабованою та зручною для подальшого супроводу.

На основі описаних модулів системи сформовано таблицю інтерфейсів (таблиця 3.3), що відображає типи вхідних даних для кожного компонента та їх взаємозв'язки.

Таблиця 3.3 – інтерфейси модулів системи

Модуль	Вхідні дані	Вихідні дані	Винятки/збої (основні)	Латентність, очікувана
assistant.py (головний цикл)	Аудіопотік з мікрофона; конфігурація команд (speech.json)	Розпізнаний текст; виклик відповідної функції; текст відповіді, переданий у TTS	sr.WaitTimeoutError; sr.UnknownValueError; sr.RequestError; помилки читання JSON	10–50 мс власної логіки (без ASR/TTS)
ASR (через speech_recognition)	Аудіофрагмент ≤5 с	Текст українською	Мережеві таймаути/HTTP-помилки; низька якість сигналу	300–900 мс для коротких фраз
Router (speech_commands)	Текст користувача; правила з speech.json	Визначений намір; шаблон відповіді; параметри підстановок	Відсутність збігів; некоректний шаблон/плейсхолдер	1–5 мс

Продовження таблиці 3.3

Модуль	Вхідні дані	Вихідні дані	Винятки/збої (основні)	Латентність, очікувана
functions.get_time	Текст (ігнорується)	{"час": "HH:MM"}	Немає суттєвих	<1 мс
functions.get_random_number	Текст (ігнорується)	{"число": N}	Немає суттєвих	<1 мс
functions.get_dollar_currency	Текст; HTTP до API банку	{"курс": 00.00} або повідомлення про недоступність	requests.Timeout; requests.HTTPError; ValueError при парсингу	100–400 мс (за наявності мережі)
functions.game	Голосові відповіді користувача в циклі	Голосові підказки; завершення гри	sr.* помилки; некоректний ввід	Інтерактивно; 200–800 мс на кожну репліку (ASR)
functions.ai (LLM)	Текст запиту; профіль (як контекст)	{"текст": "коротка відповідь"}	Мережеві таймаути; помилки API; відсутній ключ	800–2500 мс для короткої відповіді
output.say / edge_tts	Текст відповіді; параметри голосу	MP3 тимчасовий файл; відтворений звук	Помилки синтезу; відтворення; блокування файлу	200–700 мс синтез + відтворення залежно від довжини
Конфігурація (speech.json)	Файл JSON	Об'єкти правил	FileNotFoundError; JSONDecodeError	Одноразово 1–5 мс
Керування тимчасовими файлами	Шлях до MP3	Видалений файл	PermissionError; FileNotFoundError	<5 мс

4 РЕЗУЛЬТАТИ РОБОТИ

4.1 Інтерфейс користувача

Інтерфейс користувача голосового асистента суттєво відрізняється від традиційних графічних інтерфейсів, оскільки взаємодія відбувається виключно через голосові команди та аудіовідповіді. У такому типі систем ключовими аспектами є природність спілкування, швидкість реакції, чіткість формулювань та мінімізація когнітивного навантаження на користувача. У цьому розділі описано концепцію голосового інтерфейсу, сценарії взаємодії, структуру діалогів та особливості подання інформації користувачу.

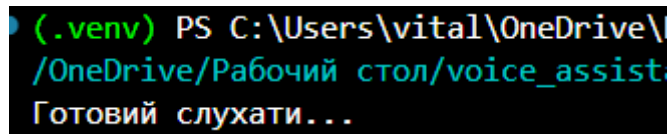
Концепція взаємодії побудована на принципі коротких, однозначних і завершених команд. Користувач активує асистента за допомогою ключового слова «Аня», після чого система переходить у режим очікування наступної фрази. Такий підхід дозволяє уникнути випадкових активацій і забезпечує природний стиль спілкування, подібний до взаємодії з реальним співрозмовником.

Голосовий інтерфейс орієнтований на мінімалізм: відповіді формуються максимально лаконічно, без зайвих уточнень або довгих пояснень. Це дозволяє користувачу швидко отримувати потрібну інформацію, не витрачаючи час на складні діалоги. Важливою характеристикою є також швидкодія — асистент реагує протягом кількох секунд, що створює відчуття безперервної та природної взаємодії.

Взаємодія між користувачем і системою складається з трьох основних етапів: активації, формулювання команди та отримання відповіді (таблиця 4.1). Після активації асистент очікує коротку фразу, яку інтерпретує як намір користувача. Далі система визначає, чи належить команда до локальних функцій, чи потребує звернення до мовної моделі. Після виконання дії асистент озвучує результат і завершує діалог, повертаючись у фоновий режим. Активація Користувач вимовляє ключове слово «Аня», після чого система переходить у режим очікування команди.

Таблиця 4.1 – Етапи голосової взаємодії

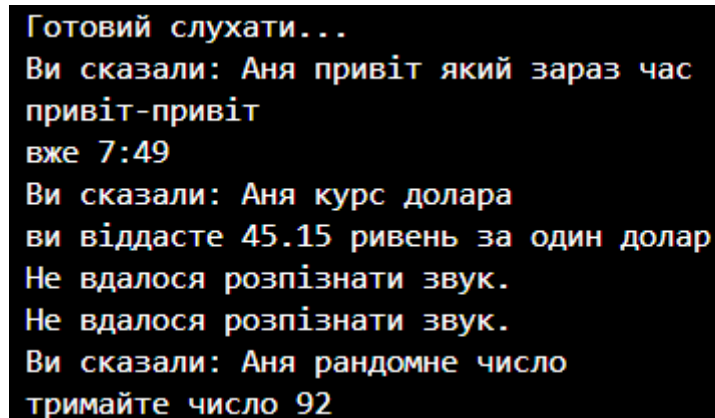
Етап	Опис
Активація	Активація відбувається через консольну панель
Команда	Користувач промовляє ключове слово “Аня”, потім формулює коротку фразу, що містить намір
Відповідь	Асистент виконує дію, формує відповідь і озвучує її



```
(.venv) PS C:\Users\vital\OneDrive\
/OneDrive/Робочий стол/voice_assist
Готовий слухати...
```

Рисунок 4.1 – активація голосового асистента в консолі

Сценарії діалогу демонструють типові ситуації, у яких користувач взаємодіє з асистентом. Наприклад, під час отримання часу користувач активує систему та ставить запитання, після чого асистент одразу озвучує точний час. Запит щодо курсу валют: система звертається до зовнішнього API, отримує актуальні дані та формує коротку відповідь. У випадку генерації випадкового числа асистент виконує локальну функцію та озвучує результат.



```
Готовий слухати...
Ви сказали: Аня привіт який зараз час
привіт-привіт
вже 7:49
Ви сказали: Аня курс долара
ви віддасте 45.15 ривень за один долар
Не вдалося розпізнати звук.
Не вдалося розпізнати звук.
Ви сказали: Аня рандомне число
тримайте число 92
```

Рисунок 4.2 – демонстрація команд який зараз час, курс долара та рандомне число

На рисунку 4.2 можемо побачити “Не вдалося розпізнати звук”, це відбувається через те що за відведений час який дається користувачу для відповіді користувач нічого не говорив.

Цікавою можливістю асистента є міні-гра, яка демонструє його можливість підтримувати просту інтерактивну взаємодію. Користувач говорить ключові слова “Аня зіграємо” далі система загадує число, користувач робить припущення, а асистент реагує на кожну відповідь, доки гра не буде завершена. Це створює ефект діалогу, хоча система не зберігає довготривалий контекст. На рисунку 4.3 ми можемо побачити процес самої міні-гри, варто зауважити що розпізнаванню мовлення тяжко справитись з числами тому буває що числа розпізнаються не повністю, або не правильно. В консолі не виводиться, але після кожної відповіді користувачем виводиться чи число яке загадав асистент більше за те що загадав користувач чи менше.

```
Ви сказали: Аня зіграємо
ready game
Ви сказали: 80
Ви сказали: 10
Не вдалося розпізнати звук.
Не вдалося розпізнати звук.
Ви сказали: 41
Ви сказали: 60
Ви сказали: 5
Ви сказали: 75
Ви сказали: п'ять
Ви сказали: 66
Ви сказали: 7
Ви сказали: 71
Ви сказали: 68
Ви сказали: 67
круто зіграли
```

Рисунок 4.3 – міні-гра в відгадай число

У випадку відкритих запитань асистент передає текст мовній моделі, яка формує коротке пояснення. Це дозволяє відповідати на широкий спектр питань,

не обмежуючись лише локальними функціями. На рисунку 4.4 зображено як формується запит для використання ШІ, ключовим словом для цього після слова “Аня” є слово “допоможи”.

```
(.venv) PS C:\Users\vital\OneDrive\Рабочий стол\voice_assistant>  
/OneDrive/Рабочий стол/voice_assistant/assistant.py  
Готовий слухати...  
Ви сказали: Аня допоможи коли засноване Запоріжжя  
Запоріжжя було засноване в 1770 році.
```

Рисунок 4.4 – використання штучного інтелекту

Оскільки інтерфейс є повністю голосовим, особливу увагу приділено тому, як саме подається інформація. Відповіді формуються короткими, чіткими та зрозумілими реченнями, що дозволяє уникнути перевантаження користувача. Синтез голосу забезпечує природне звучання, а швидкість озвучення підібрана так, щоб інформація сприймалася легко.

Важливою характеристикою є відсутність зайвих пауз, але є виділений час на отримання команди від користувача. Система одразу переходить до синтезу голосу після отримання результату.

Поточна реалізація голосового асистента орієнтована на максимально просту та швидку взаємодію, де користувач отримує відповідь одразу після вимовленої команди. Однак архітектура системи дозволяє суттєво розширити можливості інтерфейсу, перетворивши його на більш універсальний та адаптивний інструмент. Одним із перспективних напрямів розвитку є створення комбінованого інтерфейсу, який поєднує голосову взаємодію з графічними елементами. Це може бути невелике вікно або панель, що відображає історію команд, статус мікрофона, активність асистента або короткі текстові підказки. Такий підхід підвищить зручність використання у ситуаціях, коли голосової відповіді недостатньо або коли користувач бажає переглянути попередні результати.

Ще одним напрямом розвитку є впровадження системи налаштувань, доступної через простий графічний інтерфейс. Користувач міг би змінювати

голос асистента, швидкість озвучення, стиль відповідей або рівень деталізації інформації. Це дозволило б адаптувати асистента до індивідуальних уподобань та зробити взаємодію більш персоналізованою. Крім того, можливість вибору між кількома голосами або мовами відкрила б шлях до використання системи у ширшому колі сценаріїв.

Перспективним є також розширення діалогових можливостей. Хоча поточна версія асистента орієнтована на короткі команди, у майбутньому можна реалізувати підтримку багатокрокових діалогів, коли система здатна уточнювати деталі, запам'ятовувати контекст і продовжувати розмову без повторної активації. Це дозволить створити більш природну модель взаємодії, наближену до реального спілкування.

Інтеграція з іншими застосунками та сервісами. Наприклад, асистент може отримати можливість відкривати програми, керувати музикою, створювати нагадування або взаємодіяти з браузером. Такий функціонал значно розширить практичну цінність системи та зробить її корисною у повсякденних задачах.

Створення та впровадження візуальних індикаторів активності, таких як анімація під час прослуховування або синтезу голосу. Це покращить зворотний зв'язок і допоможе користувачу зрозуміти, на якому етапі обробки перебуває система. Подібні елементи часто використовуються у комерційних голосових асистентах і позитивно впливають на сприйняття інтерфейсу.

Таким чином, інтерфейс голосового асистента має значний потенціал для розвитку. Його можна розширювати як у напрямку покращення зручності використання, так і у напрямку інтеграції з іншими системами та сервісами.

4.2 Масштабованість та можливості розвитку

Архітектура розробленої системи передбачає подальше розширення функціональності без необхідності суттєвих змін у вже реалізованих модулях.

Завдяки модульному підходу та чіткому розмежуванню відповідальності між компонентами, система може еволюціонувати як у напрямку покращення якості розпізнавання мовлення, так і у напрямку розширення набору доступних користувачу можливостей. Зокрема, структура дозволяє інтегрувати альтернативні механізми розпізнавання мовлення, включаючи офлайн-моделі, що забезпечить роботу асистента без доступу до мережі.

Крім того, система може бути доповнена новими функціями керування застосунками, розширеними сценаріями діалогу або графічним інтерфейсом для налаштування параметрів. Архітектура також підтримує можливість впровадження багатомовності та механізмів довготривалої пам'яті, що дозволить асистенту адаптуватися до стилю спілкування конкретного користувача. Таким чином, проєкт має значний потенціал для подальшого розвитку та масштабування відповідно до потреб користувачів і технологічних тенденцій.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено інтелектуальну систему голосової підтримки користувача, яка поєднує можливості розпізнавання мовлення, обробки команд, виконання локальних функцій та генерації відповідей за допомогою мовної моделі. Проведений аналіз предметної області дозволив визначити ключові особливості голосових асистентів, їхні переваги, обмеження та сфери застосування. На основі цього аналізу було сформовано вимоги до програмного забезпечення, які охоплюють функціональні, нефункціональні та системні аспекти роботи асистента.

У процесі моделювання було побудовано діаграми варіантів використання та діаграми послідовності, що відобразили логіку взаємодії користувача з системою та внутрішні процеси обробки голосових команд. Детальний опис прецедентів дав змогу формалізувати поведінку системи в різних сценаріях, включаючи роботу з локальними функціями, інтерактивною міні-грою та відкритими запитамі, які обробляються мовною моделлю.

Під час проєктування архітектури було створено модульну структуру, що забезпечує гнучкість, масштабованість і можливість подальшого розвитку системи. Окремі модулі відповідають за розпізнавання мовлення, маршрутизацію намірів, виконання функцій, синтез голосу та конфігурацію команд. Такий підхід дозволив розділити відповідальність між компонентами та спростити процес супроводу й розширення функціональності. Обґрунтування вибору технологій підтвердило доцільність використання Python, бібліотек для ASR і TTS, а також зовнішніх API для отримання актуальних даних і генерації відповідей.

Проєктування інтерфейсу користувача показало, що голосова взаємодія може бути ефективною та інтуїтивною за умови дотримання принципів лаконічності, природності та швидкої реакції. Сформовані сценарії діалогу демонструють простоту використання асистента та його здатність виконувати як

стандартні команди, так і відкриті запити. Окрему увагу приділено можливостям подальшого розвитку інтерфейсу, включаючи створення комбінованого голосово-графічного середовища та розширення діалогових можливостей.

У результаті виконання роботи було створено працездатний прототип голосового асистента, який відповідає визначеним вимогам, демонструє стабільну роботу та має значний потенціал для подальшого вдосконалення. Отримані результати підтверджують ефективність обраних методів, технологій та архітектурних рішень, а також відкривають можливості для розвитку системи у напрямку інтеграції з іншими сервісами, розширення функціональності та підвищення рівня персоналізації.

ПЕРЕЛІК ДЖЕРЕЛ

1. Sommerville I. *Software Engineering*. 10th ed. — Boston: Pearson, 2016. — 816 p.
2. Pressman R., Maxim B. *Software Engineering: A Practitioner's Approach*. 9th ed. — McGraw-Hill, 2019. — 976 p.
3. IEEE Std 830-1998. *IEEE Recommended Practice for Software Requirements Specifications*. — IEEE Computer Society, 1998.
4. Fowler M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd ed. — Addison-Wesley, 2004. — 208 p.
5. Jurafsky D., Martin J. *Speech and Language Processing*. 3rd ed. — Draft, 2023. URL: <https://web.stanford.edu/~jurafsky/slp3/>
6. Python Software Foundation. *Python Documentation*. URL: <https://docs.python.org/>
7. OpenAI. *OpenAI API Documentation*. URL: <https://platform.openai.com/docs/>
8. Microsoft. *Edge Text-to-Speech (edge-tts) Documentation*. URL: <https://learn.microsoft.com/>
9. Richardson L., Ruby S. *RESTful Web Services*. — O'Reilly Media, 2007. — 454 p.
10. ПриватБанк. *API ПриватБанку*. URL: <https://api.privatbank.ua/>
11. Bishop C. *Pattern Recognition and Machine Learning*. — Springer, 2006. — 738 p.
12. Shneiderman B., Plaisant C., Cohen M., Jacobs S. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 6th ed. — Pearson, 2017. — 624 p.
13. Nielsen J. *Usability Engineering*. — Morgan Kaufmann, 1994. — 362 p.
14. ISO/IEC/IEEE 42010:2011. *Systems and Software Engineering — Architecture Description*. — ISO, 2011.

15. McConnell S. *Code Complete*. 2nd ed. — Microsoft Press, 2004. — 960 p.
16. SpeechRecognition Library. *SpeechRecognition Documentation*. URL: <https://pypi.org/project/SpeechRecognition/> ([pypi.org in Bing](#))
17. AsyncIO Documentation. *Asynchronous I/O in Python*. URL: <https://docs.python.org/3/library/asyncio.html> ([docs.python.org in Bing](#))
18. JSON.org. *Introducing JSON*. URL: <https://www.json.org/>

ДОДАТОК А

СЛАЙДИ ПРЕЗЕНТАЦІЇ

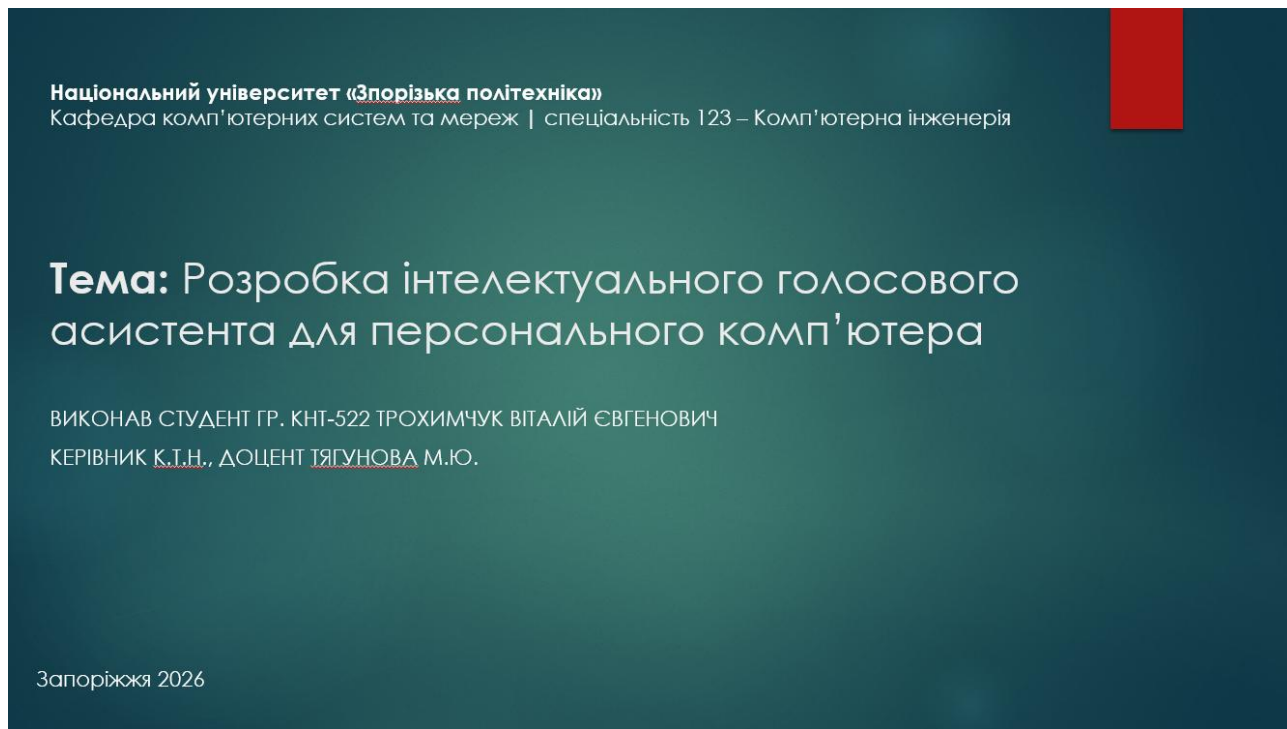


Рисунок А.1 – Слайд 1

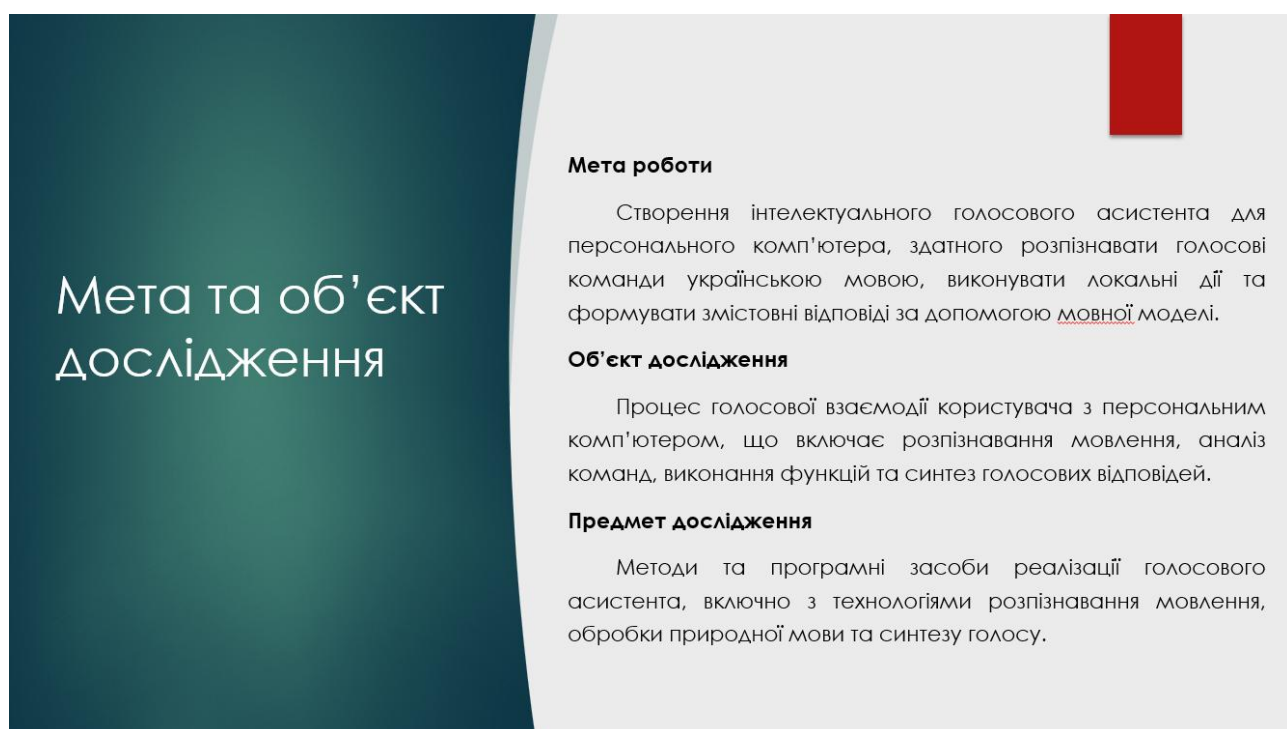


Рисунок А.2 – Слайд 2

Актуальність роботи

Голосові технології стають одним із ключових напрямів розвитку сучасних інтерфейсів взаємодії людини з комп'ютером. Більшість популярних голосових асистентів орієнтовані на мобільні пристрої або закриті екосистеми, що обмежує їх використання на персональних комп'ютерах. Зростає потреба у швидкому доступі до інформації, автоматизації рутинних дій та зручних інструментах підтримки користувача. Використання мовних моделей відкриває можливість створення інтелектуальних систем, здатних відповідати на відкриті запити та адаптуватися до стилю спілкування користувача. Розробка власного голосового асистента для ПК дозволяє забезпечити автономність, персоналізацію та гнучкість, яких не вистачає існуючим комерційним рішенням.

Рисунок А.3 – Слайд 3

Аналіз аналогів

Серед найпоширеніших голосових асистентів, які використовуються сьогодні, можна виділити Siri, Google Assistant, Amazon Alexa та Microsoft Copilot (Cortana). Siri забезпечує глибоку інтеграцію з екосистемою Apple та стабільну роботу на пристроях цього виробника, однак її можливості на персональних комп'ютерах є обмеженими через закритість платформи. Google Assistant вирізняється високою точністю розпізнавання природної мови та широкими можливостями пошуку, проте повністю залежить від хмарної інфраструктури й не орієнтований на локальне виконання команд. Amazon Alexa є одним із найкращих рішень для керування «розумним домом», має розвинену систему навичок, але слабо інтегрується з персональними комп'ютерами та використовується переважно у побутових сценаріях. Microsoft Copilot (раніше Cortana) орієнтований на офісні та робочі задачі, інтегрований у Windows, але вже не позиціонується як класичний голосовий асистент.

Siri (Apple)

**Google
Assistant**

Amazon Alexa

**Microsoft Copilot
/ Cortana**

Рисунок А.4 – Слайд 4

Архітектура системи

Архітектура розробленого голосового асистента побудована за модульним принципом і складається з кількох взаємопов'язаних компонентів. Центральним елементом є модуль `assistant.py`, який відповідає за активацію асистента, отримання аудіосигналу з мікрофона, розпізнавання мовлення та маршрутизацію команд. Модуль `functions.py` реалізує локальні функції системи, такі як отримання часу, генерація випадкових чисел, визначення курсу валют та взаємодія з мовною моделлю. Модуль `output.py` забезпечує синтез голосових відповідей українською мовою та їх відтворення. Додаткові конфігураційні файли, зокрема `speech.json` та `key.py`, містять словник команд і ключі доступу до зовнішніх сервісів. Така архітектура забезпечує гнучкість, масштабованість і можливість подальшого розширення функціональності.

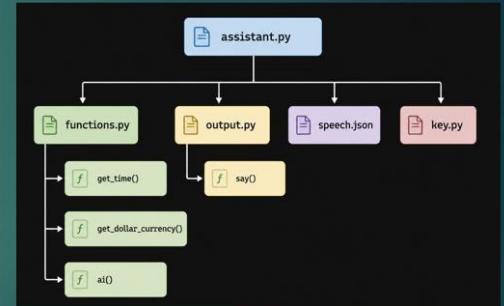


Рисунок А.5 – Слайд 5

Алгоритм роботи системи

Робота голосового асистента відбувається послідовно, за чітко визначеним алгоритмом. Користувач активує асистента та промовляє команду в мікрофон, після чого аудіосигнал передається до модуля розпізнавання мовлення. Розпізнаний текст потрапляє до головного модуля `assistant.py`, який аналізує команду, співставляє її зі словником у `speech.json` та визначає, яку функцію необхідно виконати. Далі викликаються відповідні функції з модуля `functions.py` або зовнішньої мовної моделі, після чого сформована відповідь передається до модуля `output.py`, де здійснюється синтез голосу та відтворення аудіо для користувача.

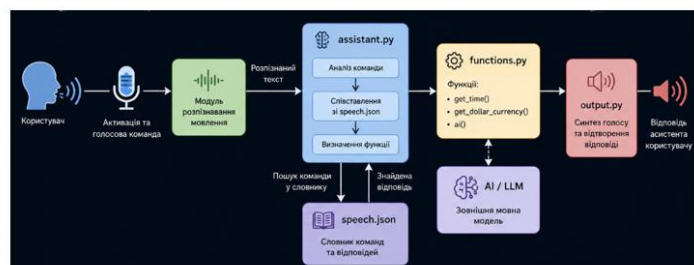


Рисунок А.6 – Слайд 6

Реалізація системи

Система реалізована мовою програмування Python із використанням низки спеціалізованих бібліотек. Для розпізнавання мовлення застосовується відповідна бібліотека, що забезпечує перетворення аудіосигналу на текст українською мовою. Логіка обробки команд та виклику функцій реалізована в модулі `assistant.py`, тоді як модуль `functions.py` містить окремі функції для отримання часу, курсу валют, генерації випадкових чисел та взаємодії з мовною моделлю. Синтез голосових відповідей здійснюється за допомогою модуля `output.py`, який формує аудіосигнал на основі тексту та відтворює його користувачу. Конфігураційні дані та ключі доступу до API зберігаються у файлах `speech.json` та `key.py`.

Рисунок А.7 – Слайд 7

Інтерфейс користувача

Взаємодія користувача з системою здійснюється у форматі голосового інтерфейсу. Користувач промовляє команду в мікрофон, не використовуючи клавіатуру чи графічні елементи, а асистент автоматично розпізнає мовлення, аналізує зміст запиту та формує відповідь. Результат роботи системи повертається у вигляді голосового повідомлення, що робить взаємодію інтуїтивно зрозумілою та зручною. Такий підхід дозволяє використовувати асистента для виконання повсякденних задач на персональному комп'ютері без необхідності переходу між різними програмами чи вікнами.

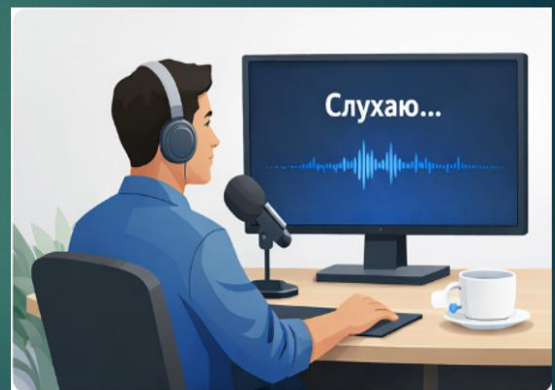


Рисунок А.8 – Слайд 8

Тестування та результати

Для оцінки якості роботи голосового асистента було проведено тестування за кількома критеріями, зокрема точністю розпізнавання мовлення, швидкістю формування відповіді та стабільністю роботи системи. Під час тестування використовувалися різні типи команд: запити часу, інформаційні запити, команди з використанням мовної моделі та побутові фрази. Отримані результати показали, що система коректно розпізнає більшість команд українською мовою, забезпечує прийнятний час відповіді та стабільно працює в умовах реального використання.

```
Готовий слухати...  
Ви сказали: Аня привіт який зараз час  
привіт-привіт  
вже 7:49  
Ви сказали: Аня курс долара  
ви віддасте 45.15 ривень за один долар  
Не вдалося розпізнати звук.  
Не вдалося розпізнати звук.  
Ви сказали: Аня рандомне число  
тримайте число 92
```

```
(.venv) PS C:\Users\vital\OneDrive\Рабочий стол\voice_assistant>  
/OneDrive/Рабочий стол/voice_assistant/assistant.py"  
Готовий слухати...  
Ви сказали: Аня допоможи коли засноване Запоріжжя  
Запоріжжя було засноване в 1770 році.
```

Рисунок А.9 – Слайд 9