Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
__Олена ОЛЬХОВСЬКА
(підпис)
«___»_____2026р.

КВАЛІФІКАЦІЙНА РОБОТА
на тему
«РОЗРОБКА ІНТЕЛЕКТУАЛЬНОГО ГОЛОСОВОГО ПОМІЧНИКА З ВИКОРИСТАННЯ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ»

зі спеціальності 122 «Комп’ютерні науки»
освітня програма «Комп’ютерні науки»
ступеня бакалавр

Виконавець роботи Кірпіков Артем Володимирович
_____«__»_____2026р.
(підпис)
Науковий керівник к.ф.-м.н., Олексійчук Ю.Ф
_____«__»_____2026р.
(підпис)

Рецензент

ПОЛТАВА 2026

РЕФЕРАТ

Записка: 76 с.., 17 табл., 2 рис. 2 додатки, 25 джерел.

ГОЛОСОВИЙ ПОМІЧНИК, ШТУЧНИЙ ІНТЕЛЕКТ, РОЗПІЗНАВАННЯ МОВЛЕННЯ, ВЕЛИКА МОВНА МОДЕЛЬ, СИНТЕЗ МОВЛЕННЯ, ОБРОБКА ПРИРОДНОЇ МОВИ.

Об'єкт розробки – процес голосової взаємодії користувача з інтелектуальною комп'ютерною системою.

Мета роботи – проектування та програмна реалізація інтелектуального голосового помічника на основі методів штучного інтелекту з підтримкою української мови.

Методи дослідження – методи глибокого навчання, трансформерні архітектури нейронних мереж, векторне представлення тексту, промпт-інжиніринг, середовище розробки Visual Studio Code та мова програмування Python 3.11.

Здійснено огляд сучасних підходів до побудови голосових помічників, проаналізовано наявні програмні рішення, виявлено їх переваги та недоліки. Спроектовано модульну архітектуру системи з 6 компонентів. Програмно реалізовано повний конвеєр обробки голосових запитів на основі моделей Whisper, sentence-transformers та LLM.

Здійснена програмна реалізація забезпечує підтримку 8 типів намірів, контекстний діалог та синтез мовлення українською мовою.

В результаті тестування виявлено, що середній показник WER становить 9,4 %, точність класифікації намірів – 89 %, а загальний час відповіді – 4,8 с на CPU.

ЗМІСТ

ВСТУП 5
1. ПОСТАНОВКА ЗАДАЧІ 7
1.1 Змістовна постановка задачі розробки інтелектуального голосового помічника 7
1.2 Модель задачі та її характеристики 9
2 ІНФОРМАЦІЙНИЙ ОГЛЯД 13
2.1 Огляд робіт, де розглянуте аналогічне до теми роботи завдання 13
2.2 Позитивні аспекти оглянутих робіт 17
2.3 Вади розробок з оглянутих робіт 20
2.4 Необхідність та актуальність теми роботи 23
3 ТЕОРЕТИЧНА ЧАСТИНА 27
3.1 Проектування архітектури програмного забезпечення голосового помічника 27
3.2 Графічне представлення архітектури системи 30
3.3 Обгрунтування вибору програмних засобів та методів штучного інтелекту для реалізації завдання роботи 35
4 ПРАКТИЧНА ЧАСТИНА 39
4.1 Опис процесу програмної реалізації інтелектуального голосового помічника 39
4.2 Опис програми 43
4.3 Перевірка валідності. Дослідження можливостей програмної реалізації 46
4.4 Необхідна користувачу програми інструкція 50
ВИСНОВКИ 54
РЕКОМЕНДАЦІЇ 56
СПИСОК ЛІТЕРАТУРИ 58
ДОДАТКИ 61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні скорочення / позначення	Пояснення
АРМ	автоматизоване робоче місце
АШІ	архітектура штучного інтелекту
ВМ	велика мовна модель
ГП	голосовий помічник
НМ	нейронна мережа
ОПМ	обробка природної мови
API	Application Programming Interface – інтерфейс програмування застосунків
ASR	Automatic Speech Recognition – автоматичне розпізнавання мовлення
CPU	Central Processing Unit – центральний процесор
CUDA	Compute Unified Device Architecture – архітектура паралельних обчислень NVIDIA
GPU	Graphics Processing Unit – графічний процесор
LLM	Large Language Model – велика мовна модель
NLU	Natural Language Understanding – розуміння природної мови
TTS	Text-to-Speech – синтез мовлення з тексту
VAD	Voice Activity Detection – виявлення активності голосу
WER	Word Error Rate – частота помилок на рівні слів
YAML	Yet Another Markup Language – формат серіалізації даних

ВСТУП

Стрімкий розвиток технологій штучного інтелекту в останнє десятиліття докорінно змінив підходи до взаємодії людини з комп'ютерними системами. Голосові інтерфейси, які ще кілька десятиліть тому вважалися елементом наукової фантастики, сьогодні стали повноцінним інструментом повсякденного використання — від побутових смарт-пристроїв до складних корпоративних рішень. Проте попри значний прогрес у цій галузі, більшість наявних комерційних рішень або орієнтовані на англомовний ринок, або мають суттєві обмеження у розумінні контексту, адаптивності та можливості розширення функціональності. Це породжує актуальну науково-практичну задачу розробки інтелектуального голосового помічника, здатного ефективно вирішувати широке коло завдань з урахуванням специфіки запитів користувача.

Актуальність теми обумовлена кількома чинниками. По-перше, зростаючим попитом на безконтактні та природномовні інтерфейси управління програмним забезпеченням у різних сферах — від освіти й медицини до промисловості та сфери послуг. По-друге, стрімким розвитком методів глибокого навчання, трансформерних моделей обробки природної мови та нейронних мереж, що відкривають якісно нові можливості для розпізнавання мовлення та генерації відповідей. По-третє, недостатньою кількістю відкритих і адаптованих рішень, орієнтованих на потреби україномовних користувачів. Метою дипломної роботи є проектування та програмна реалізація інтелектуального голосового помічника з використанням сучасних методів штучного інтелекту, здатного розпізнавати голосові команди, обробляти природномовні запити та формувати релевантні відповіді або виконувати відповідні дії.

Для досягнення поставленої мети необхідно вирішити такі задачі:

1. проаналізувати наявні підходи до побудови голосових помічників та виявити їх переваги й недоліки;

2. обґрунтувати вибір архітектурних рішень та інструментальних засобів реалізації;
3. спроектувати архітектуру програмного забезпечення голосового помічника;
4. реалізувати систему розпізнавання мовлення, модуль обробки природної мови та модуль генерації відповідей;
5. провести тестування розробленої системи та оцінити якість її роботи.

Об'єктом дослідження є процес автоматизованої обробки природномовних голосових запитів користувача з використанням методів штучного інтелекту.

Предметом дослідження є методи, алгоритми та програмні засоби побудови інтелектуального голосового помічника на основі технологій розпізнавання мовлення та обробки природної мови.

У процесі виконання роботи використовувались такі методи дослідження: аналіз і синтез наукових джерел для формування теоретичного підґрунтя; методи порівняльного аналізу існуючих рішень; методи об'єктно-орієнтованого проектування при розробці архітектури системи; методи машинного навчання та глибокого навчання при реалізації компонентів обробки мовлення; методи тестування програмного забезпечення при перевірці валідності розробленої системи.

Практична цінність роботи полягає у створенні програмного продукту, який може бути використаний як самостійне рішення або як основа для подальшого розширення та інтеграції у більш складні інформаційні системи. Розроблений голосовий помічник демонструє реалізацію повного циклу обробки голосового запиту — від акустичного введення до формування змістовної відповіді — що становить як науковий, так і прикладний інтерес.

Структура роботи відповідає поставленій меті та задачам. Робота складається зі вступу, чотирьох розділів, висновків та списку використаних джерел.

1. ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовна постановка задачі розробки інтелектуального голосового помічника

Сучасний етап розвитку інформаційних технологій характеризується стрімким проникненням систем штучного інтелекту в усі сфери людської діяльності. Серед найбільш динамічно зростаючих напрямів виділяється розробка інтелектуальних голосових помічників – програмних систем, здатних сприймати природномовні голосові команди, аналізувати їх семантичний зміст та формувати змістовні відповіді або виконувати відповідні дії в автоматизованому режимі. Від простих голосових меню з фіксованими сценаріями реагування до складних мультимодальних систем, що розуміють контекст розмови та адаптуються до потреб конкретного користувача, – цей шлях пройдено менш ніж за 2 десятиліття. Проте попри значний прогрес галузі, значна частина наявних рішень зосереджена переважно на англомовному ринку, тоді як потреби україномовних користувачів залишаються значною мірою незадоволеними.

Змістовна постановка задачі розробки інтелектуального голосового помічника полягає у проектуванні та програмній реалізації програмної системи, що забезпечує повний цикл обробки голосового запиту користувача: від первинного акустичного сигналу до формування релевантної відповіді чи виконання конкретної команди. Система має функціонувати в умовах реального використання, тобто успішно справлятися з природними варіаціями мовлення – різними темпами, інтонаціями, акцентами, фоновими шумами та неформальними конструкціями мови. Принципова відмінність від традиційних командних інтерфейсів полягає у здатності системи розуміти природну мову, а не лише точні ключові слова, що суттєво розширює практичне застосування такого помічника.

З технічної точки зору задача декомпозується на 3 взаємопов'язані підзадачі. Перша підзадача стосується автоматичного розпізнавання мовлення (Automatic Speech Recognition, ASR) – перетворення акустичного сигналу на текстове представлення. Ця підзадача є ключовою вхідною ланкою і визначає якість роботи всього конвеєра, оскільки похибки на цьому етапі накопичуються та впливають на точність подальшої обробки. Друга підзадача охоплює обробку природної мови (Natural Language Processing, NLP) – семантичний аналіз отриманого тексту, визначення наміру (intent) користувача, виділення ключових сутностей (entity extraction) та розуміння контексту діалогу. Третя підзадача полягає у генерації відповіді та її озвучуванні засобами синтезу мовлення (Text-to-Speech, TTS), що забезпечує природний зворотний зв'язок із користувачем.

Об'єктом розробки є програмна система інтелектуального голосового помічника, що реалізує повний конвеєр обробки голосового запиту на основі сучасних методів глибокого навчання та трансформерних архітектур. Предметом розробки є методи, алгоритми і програмні засоби, що забезпечують функціонування кожного з компонентів зазначеного конвеєра. Цільовою аудиторією системи є широке коло користувачів, які взаємодіють з персональним комп'ютером або мобільним пристроєм у режимі реального часу.

Ключовими функціональними вимогами до розроблюваної системи є такі: розпізнавання неперервного мовлення з рівнем точності не нижче 90% для стандартних умов акустичного середовища; визначення намірів користувача з правильною класифікацією не менше 85% запитів; генерація відповідей, що є семантично узгодженими з контекстом запиту; підтримка мінімальної затримки відгуку системи, що не перевищує 3 секунди від моменту завершення голосового запиту до початку відповіді. Нефункціональні вимоги передбачають модульну архітектуру системи, що забезпечує незалежне оновлення компонентів, а також кросплатформеність рішення.

Важливим аспектом постановки задачі є визначення меж функціональності системи. Голосовий помічник має опрацьовувати запити у декількох предметних доменах: управління операційною системою та застосунками, пошук і надання довідкової інформації, відтворення мультимедійного контенту, управління розкладом і нагадуваннями. Такий мультидоменний підхід суттєво ускладнює задачу класифікації намірів порівняно з однодоменними системами, оскільки вимагає формування розгалуженої ієрархії намірів та значного обсягу навчальних даних для кожного домену.

Необхідно також враховувати соціотехнічний контекст задачі. Згідно з дослідженнями останніх років, ефективність голосових помічників залежить не лише від технічних показників точності, а й від зручності взаємодії, природності діалогу та здатності системи відновлюватися після помилок розпізнавання [6]. Це вимагає реалізації механізмів підтвердження розпізнаних команд та запиту уточнення у разі неоднозначності запиту. Таким чином, задача розробки інтелектуального голосового помічника є комплексною науково-технічною проблемою, що потребує інтеграції новітніх досягнень у галузях акустичного моделювання, обробки природної мови та інженерії програмного забезпечення.

1.2 Модель задачі та її характеристики

Формалізація задачі розробки інтелектуального голосового помічника вимагає побудови чіткої математичної моделі, що відображає взаємодії між компонентами системи та описує перетворення вхідних даних на вихідні результати. Загальну модель системи можна представити як композицію 3 функціональних перетворень: $F = F_{TTS} \circ F_{NLU} \circ F_{ASR}$, де F_{ASR} реалізує перетворення акустичного сигналу на текст, F_{NLU} здійснює семантичний аналіз тексту, а F_{TTS} формує голосову відповідь.

На рівні компонента ASR модель функціонує наступним чином. Вхідний акустичний сигнал x(t) дискретизується з частотою 16 000 Гц та перетворюється на послідовність логарифмічних мел-спектрограм X = {x_1, x_2, ..., x_T}, де T – кількість часових кадрів. Акустична модель на основі трансформерної архітектури відображає цю послідовність у найбільш імовірну текстову послідовність W = {w_1, w_2, ..., w_N} згідно з принципом максимуму апостеріорної ймовірності. Сучасні системи, зокрема Whisper від OpenAI, досягають рівня точності розпізнавання з коефіцієнтом помилок за словами (WER) менше 5% на стандартних тестових наборах даних завдяки навчанню на понад 680 000 годинах різноманітних аудіозаписів [12].

Компонент NLU оперує отриманим текстом і вирішує 2 паралельні підзадачі. Перша – класифікація наміру (intent classification), тобто визначення того, яку дію прагне виконати користувач. Формально це відображення $W \rightarrow I$, де I є елементом з множини наперед визначених намірів {i_1, i_2, ..., i_K}. Друга підзадача – виділення сутностей (named entity recognition, NER), тобто ідентифікація значущих фрагментів тексту, що несуть конкретну інформацію: часові мітки, географічні назви, імена, числові значення тощо. Сучасні моделі на основі попередньо навчених трансформерів, таких як BERT та його похідні, демонструють точність класифікації намірів вище 93% на стандартних бенчмарках [4]. Поєднання класифікації намірів та виділення сутностей у єдиній моделі є перевагою порівняно з розподіленим підходом, оскільки дозволяє використати взаємний вплив цих задач для підвищення загальної якості [8].

Для компонента генерації відповіді використовується модель діалогового управління, що на основі визначеного наміру I та виділених сутностей E формує відповідь R або ініціює виконання відповідної дії A. У найпростіших сценаріях відображення є детермінованим і реалізується через систему правил. У складніших випадках, де важливий контекст багатоходового діалогу, застосовуються авторегресивні мовні моделі, що

генерують відповідь токен за токеном з урахуванням повної історії розмови [3].

Характеристики моделі доцільно структурувати за кількома вимірами. Перший вимір – точність (accuracy), що для кожного компонента вимірюється власними метриками: WER для ASR, F1-міра для NER, точність класифікації для intent detection. Другий вимір – латентність (latency), яка є критичною для досягнення природного відчуття взаємодії. Дослідження показують, що затримка відгуку понад 2 секунди суттєво знижує суб'єктивну оцінку якості голосового помічника користувачами [19]. Третій вимір – робастність (robustness), тобто здатність зберігати прийнятну якість в умовах зашумленого акустичного середовища, нестандартних акцентів та неформального мовлення. Четвертий вимір – масштабованість, що характеризує здатність системи обслуговувати зростаючу кількість одночасних користувачів без суттєвої деградації показників.

Таблиця 1.1 – Основні характеристики моделі інтелектуального голосового помічника

Компонент	Задача	Метрика якості	Цільове значення	Часова складність
ASR (Whisper)	Розпізнавання мовлення	WER, %	< 5	$O(T \cdot \log T)$
NLU Intent	Класифікація наміру	Accuracy, %	> 90	$O(N^2)$
NLU NER	Виділення сутностей	F1-міра	> 0,88	$O(N^2)$
Діалогове управління	Генерація відповіді	BLEU / точність	> 0,75	$O(N \cdot K)$
TTS	Синтез мовлення	MOS (натуральність)	> 4,0	$O(M \cdot \log M)$
Система загалом	Повний конвеєр	Затримка, с	< 3	–

Архітектурну модель системи можна описати як конвеєрну (pipeline) з можливістю паралельного виконання окремих підзадач. Вхідний акустичний сигнал надходить від мікрофона та проходить попередню обробку: видалення шуму, нормалізацію амплітуди та сегментацію на фрази за активністю голосу

(Voice Activity Detection, VAD). Результат передається до модуля ASR, вихід якого у вигляді текстового рядка надходить до модуля NLU. Модуль NLU повертає структурований об'єкт, що містить ідентифікований намір та список виявлених сутностей. Цей об'єкт обробляється блоком діалогового управління, що або формує текстову відповідь для синтезу мовлення, або передає команду виконавчому модулю системи.

Важливою характеристикою моделі є спосіб роботи з контекстом діалогу. Система підтримує вікно контексту, що включає щонайменше 5 останніх обмінів репліками. Це дозволяє коректно обробляти анафоричні посилання (наприклад, «відкрий його», де «його» посилається на раніше згаданий файл) та уточнюючі питання, що є природними в людській комунікації. Механізм управління контекстом реалізується або через збереження стану в окремій структурі даних, або через включення стислого резюме попередніх обмінів у промпт мовної моделі [11].

Система також включає механізм обробки невизначеності. Якщо впевненість класифікатора намірів є нижчою за порогове значення 0,7, система переходить до стратегії уточнення: генерує запитання для визначення справжнього наміру користувача. Такий підхід суттєво підвищує надійність системи в реальних умовах використання порівняно зі стратегією «найкращого вгадування» [6].

2 ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Огляд робіт, де розглянуте аналогічне до теми роботи завдання

Дослідження у галузі інтелектуальних голосових помічників та суміжних технологій розпізнавання мовлення й обробки природної мови набули значного прискорення протягом 2020–2025 років завдяки появі нових архітектурних рішень на основі трансформерів та зростанню обчислювальних потужностей. Систематизація актуальних наукових праць дозволяє виділити декілька ключових напрямів досліджень, кожен з яких безпосередньо стосується різних компонентів проектованої системи.

Фундаментальним напрямом є розробка моделей автоматичного розпізнавання мовлення. У 2020 році Baevski зі співавторами представили фреймворк wav2vec 2.0, що реалізує самонавчання (self-supervised learning) для отримання потужних представлень акустичного сигналу [1]. Ключова ідея підходу полягає у попередньому навчанні на великому масиві немаркованих аудіоданих з подальшим тонким налаштуванням на невеликому обсязі розмічених записів. Авторам вдалося продемонструвати, що навчання на лише 10 хвилинах маркованих даних у поєднанні з попереднім навчанням на 53 000 годинах немаркованих даних дозволяє досягти рівня WER 4,8% на тестовому наборі LibriSpeech (clean). Це відкрило нові можливості для розробки систем розпізнавання мовлення для мов з обмеженими мовними ресурсами.

Ще більш значущим кроком стала публікація у 2022 році роботи Radford та співавторів з OpenAI, присвяченої системі Whisper [12]. Автори запропонували підхід до навчання, заснований на «слабкому нагляді» (weak supervision) – використанні 680 000 годин аудіоданих з автоматично отриманими транскриптами з мережі Інтернет. Архітектура системи базується на кодувальнику-декодувальнику (encoder-decoder) трансформерного типу. Принципова новизна підходу полягає в тому, що

єдина модель вирішує одночасно 4 задачі: транскрипцію мовлення у тій самій мові, ідентифікацію мови, переклад мовлення на англійську та визначення часових міток. Система демонструє робастність щодо широкого спектра акцентів, фонових шумів та технічних умов запису, що є критично важливим для практичного застосування. У 2023 році дослідники Wang зі співавторами дослідили можливості застосування Whisper у режимі навчання на основі прикладів (in-context learning) для адаптації до нових мовних контекстів без перенавчання моделі [18].

Паралельно розвивалися дослідження ефективніших архітектур для ASR. Gao та співавтори у 2022 році запропонували Paraformer – паралельний трансформер для нерегресивного кінцево-кінцевого розпізнавання мовлення [5]. На відміну від авторегресивних систем, де кожен токен генерується послідовно, Paraformer генерує всі токени паралельно, що забезпечує значне прискорення інференсу – у 10 разів і більше при збережені рівня точності, порівнянного з авторегресивними аналогами. Tripathi зі співавторами у 2020 році запропонували Transformer Transducer, що об'єднує переваги потокового та непотокового розпізнавання мовлення в єдиній моделі [16], що є важливим для практичних застосувань, де необхідно отримувати проміжні результати ще до завершення фрази.

Інший ключовий напрям досліджень пов'язаний з розумінням природної мови та класифікацією намірів. Louvan та Magnini у 2020 році виконали детальний огляд нейронних методів для задач заповнення слотів (slot filling) та класифікації намірів у діалогових системах [8]. Автори систематизували 3 класи підходів: незалежні моделі для кожної підзадачі, спільні (joint) моделі та підходи на основі трансферного навчання. Висновок огляду полягає у тому, що спільні моделі на базі попередньо навчених трансформерів демонструють найкращі результати завдяки можливості взаємного посилення двох задач. Chen та Luo у 2023 році запропонували конкретну реалізацію такої спільної моделі, що використовує семантичне злиття ознак для покращення точності [4]. Їхня модель досягає точності

98,7% на наборі даних ATIS та 97,0% на Snips, що є результатами на рівні найкращих систем у своєму класі.

Gunaratna зі співавторами у 2023 році звернули увагу на проблему пояснюваності (explainability) NLU-моделей для голосових помічників [6]. Автори трансформували стандартну спільну модель класифікації намірів та виділення сутностей у «внутрішньо пояснювану» (inherently explainable) систему, здатну пояснювати свої рішення на гранульованому рівні без втрати точності. Це дослідження має практичне значення для підвищення довіри користувачів до голосових помічників та полегшення діагностики помилок системи.

Ще одним важливим напрямом є дослідження великих мовних моделей (Large Language Models, LLMs) як основи для генерації відповідей у голосових помічниках. Brown зі співавторами у 2020 році представили GPT-3 – модель з 175 млрд параметрів, що продемонструвала вражаючі можливості навчання на основі небагатьох прикладів [3]. Це відкрило нову парадигму розробки діалогових систем, де замість спеціалізованого навчання на конкретних задачах використовується загальна мовна модель з відповідним формулюванням запиту (prompt engineering). Ouyang зі співавторами у 2022 році запропонували InstructGPT – модель, навчену слідувати інструкціям за допомогою навчання з підкріпленням на основі людського зворотного зв'язку (RLHF) [11]. Ця робота заклала основу для сучасних систем на зразок ChatGPT та суттєво підвищила придатність LLM для практичного використання у голосових помічниках.

Minaee зі співавторами у 2024 році виконали всебічний огляд сучасних LLM, включаючи архітектури GPT, LLaMA та PaLM [10], а Zhao зі співавторами здійснили детальний огляд методів та можливостей LLM у 2023 році [24]. Ці роботи надають систематизовану базу для вибору оптимальної мовної моделі для компонента генерації відповідей голосового помічника.

Окремий напрям представляють дослідження синтезу мовлення. Wang зі співавторами у 2024 році представили VALL-E 2 – систему синтезу мовлення, що досягає якості, порівнянної з природним мовленням людини (human parity), в режимі нульового пострілу (zero-shot) при використанні лише 3-секундного аудіозразку голосу [17]. Yu зі співавторами у 2022 році запропонували DIA-TTS – систему синтезу мовлення з глибокою успадкованою увагою, що забезпечує природну просодію та інтонацію генерованого мовлення [22].

Таблиця 2.1 – Огляд ключових наукових праць за тематикою дослідження

Автор(и), рік	Напрям	Ключовий внесок	Досягнуті показники
Baevski et al., 2020	ASR	Self-supervised wav2vec 2.0	WER 1,8% (LibriSpeech clean)
Radford et al., 2022	ASR	Whisper, weak supervision	WER < 5%, 99 мов
Gao et al., 2022	ASR	Паралельний Paraformer	Прискорення у 10 разів
Tripathi et al., 2020	ASR	Transformer Transducer	Потокове + непотокове ASR
Brown et al., 2020	LLM	GPT-3, few-shot learning	175 млрд параметрів
Ouyang et al., 2022	LLM	InstructGPT, RLHF	Переваги над GPT-3 у 175 млрд
Chen, Luo, 2023	NLU	Спільна модель intent+NER	Точність 98,7% (ATIS)
Gunaratna et al., 2023	NLU	Пояснювані NLU-моделі	Без втрати точності
Louvan, Magnini, 2020	NLU	Огляд методів NLU	Систематизація підходів
Wang et al., 2024	TTS	VALL-E 2, zero-shot	Human parity якість

Touvron зі співавторами у 2023 році представили LLaMA – відкриту та ефективну фундаментальну мовну модель, яка при меншій кількості параметрів конкурує за якістю з набагато більшими закритими моделями [15]. Ця робота має особливе значення для розробки голосових помічників, оскільки відкриті моделі можна розгортати на локальних пристроях без залежності від хмарних API, що підвищує конфіденційність та знижує операційні витрати. Xiao зі співавторами у 2022 році запропонували дворівневу систему рекомендації голосових застосунків для обробки запитів, що не вдається опрацювати стандартним NLU-компонентом [19], вирішуючи

практичну проблему деградації якості на «крайових» сценаріях використання.

2.2 Позитивні аспекти оглянутих робіт

Аналіз наукових праць, розглянутих у попередньому підрозділі, дозволяє виділити низку принципово важливих позитивних аспектів та здобутків, що складають актуальний стан мистецтва (state of the art) у галузі інтелектуальних голосових помічників та супутніх технологій. Систематизація цих переваг є необхідним кроком перед обґрунтуванням власних архітектурних рішень, оскільки ефективна розробка нових систем базується на максимальному використанні вже досягнутих результатів.

Першим і найбільш фундаментальним позитивним аспектом є принципове підвищення точності розпізнавання мовлення, досягнуте завдяки трансформерним архітектурам та масштабованому навчанню. Система Whisper [12] демонструє, що навчання на різноманітному корпусі з 680 000 годин аудіоданих формує модель, стійку до широкого спектра умов запису – від студійної якості до телефонних розмов. Ця різноманітність навчальних даних є ключовою перевагою, що безпосередньо транслюється у робастність у реальних умовах використання, де умови запису практично ніколи не відповідають ідеальним лабораторним сценаріям. Коефіцієнт WER нижче 3% на стандартних тестових наборах свідчить про досягнення рівня, що наближається до людської точності транскрипції.

Другим важливим позитивним аспектом є парадигма самонавчання, реалізована у wav2vec 2.0 [1]. Традиційні ASR-системи вимагали тисяч годин ретельно розмічених аудіоданих для досягнення прийнятної якості. Wav2vec 2.0 переломив цю залежність, продемонструвавши, що добре навчений кодувальник акустичних представлень, отриманий на немаркованих даних, вимагає лише незначного обсягу розміченого матеріалу для досягнення конкурентної точності. Це відкриває можливості для розробки систем для

мов з обмеженими мовними ресурсами, включаючи українську мову, де обсяг якісно розмічених мовних корпусів є значно меншим порівняно з англійською.

Третім позитивним аспектом є архітектурні досягнення у підвищенні швидкодії ASR без суттєвої втрати точності. Paraformer [5] та Transformer Transducer [16] вирішують практичну проблему балансу між точністю та швидкістю. Паралельна генерація всіх токенів у Paraformer дозволяє у 10 і більше разів прискорити інференс порівняно з авторегресивними системами. Для голосового помічника, де затримка відгуку є критичним показником якості взаємодії, ця перевага є безпосередньо застосовною. Transformer Transducer, у свою чергу, забезпечує природне переключення між потоковим та пакетним режимами обробки, що дозволяє системі надавати проміжні транскрипти ще під час вимовляння фрази.

Четвертим позитивним аспектом є якісний стрибок у можливостях NLU-компонентів завдяки попередньо навченим трансформерним моделям. Результати Chen та Luo [4], що демонструють точність класифікації намірів вище 98% на стандартних наборах даних, свідчать про те, що задача розуміння намірів у рамках фіксованого набору доменів є фактично вирішеною для достатньо великих обсягів навчальних даних. Спільне навчання класифікації намірів та виділення сутностей, запропоноване в аналізованих роботах, дає синергетичний ефект: помилки у визначенні наміру корегуються сигналами від задачі NER і навпаки.

П'ятим позитивним аспектом є підхід до пояснюваності NLU-систем для голосових помічників, запропонований Gunaratna зі співавторами [6]. Більшість сучасних нейронних NLU-моделей є «чорними ящиками», що суттєво ускладнює їх відлагодження та адаптацію. Можливість отримати пояснення для кожного прийнятого рішення на рівні окремих ознак суттєво полегшує процес розробки, дозволяє виявляти систематичні помилки класифікатора та пояснювати поведінку системи кінцевим користувачам. Це

підвищує довіру до системи та прискорює її вдосконалення в рамках ітеративного циклу розробки.

Шостим позитивним аспектом є результати у галузі великих мовних моделей, зокрема InstructGPT [11] та GPT-4 [2]. Навчання з підкріпленням на основі людського зворотного зв'язку (RLHF) принципово змінило здатність LLM слідувати інструкціям, бути корисними та уникати шкідливих або некоректних відповідей. Для голосового помічника це означає, що компонент генерації відповідей на основі LLM може природно справлятися з ситуаціями, не передбаченими під час розробки, формулювати відповіді у відповідному тоні та стилі та обробляти складні багатоетапні запити. Touvron зі співавторами [15] показали, що ця якість досяжна і у відкритих моделях розумних розмірів, що є критично важливим для практичного розгортання.

Таблиця 2.2 – Позитивні аспекти оглянутих робіт у контексті розробки голосового помічника

Аспект	Відповідна робота	Перевага для системи	Практична застосовність
Висока точність ASR	Radford et al. [12]	WER < 5% у реальних умовах	Висока
Self-supervised навчання	Baevski et al. [1]	Мала кількість маркованих даних	Висока
Швидкий паралельний ASR	Gao et al. [5]	Зниження латентності у 10 разів	Висока
Точна класифікація намірів	Chen, Luo [4]	Accuracy > 98%	Висока
Пояснювані NLU-моделі	Gunaratna et al. [6]	Прозорість рішень	Середня
RLHF-навчання LLM	Ouyang et al. [11]	Природні, корисні відповіді	Висока
Відкриті LLM	Touvron et al. [15]	Локальне розгортання	Висока
Zero-shot TTS	Wang et al. [17]	Синтез будь-якого голосу	Середня

Сьомим позитивним аспектом є досягнення у синтезі мовлення. Система VALL-E 2 [17] продемонструвала здатність відтворювати голос конкретного мовця з якістю, що суб'єктивно не відрізняється від природного мовлення, на основі лише 3-секундного зразка. Для голосового помічника це

означає можливість персоналізації голосового відгуку системи без дорогого студійного запису. DIA-TTS [22] надає альтернативне рішення з акцентом на природній просодії та інтонаційному різноманітті, що підвищує суб'єктивне сприйняття природності синтетичного мовлення.

Восьмим позитивним аспектом є розробка Rasa – відкритого фреймворку для побудови розмовних агентів [13]. Rasa надає готові компоненти для повного конвеєра NLU та управління діалогом, що значно скорочує час розробки системи. Модульна архітектура фреймворку дозволяє замінювати окремі компоненти на власні реалізації, зберігаючи при цьому зручну інфраструктуру для тренування, оцінки та розгортання моделей. Наявність активної спільноти та розгалуженої документації є суттєвою перевагою для розробника.

2.3 Вади розробок з оглянутих робіт

Попри значний прогрес у галузі, оглянуті роботи мають суттєві обмеження та невирішені проблеми, розуміння яких є необхідним для коректного проектування власної системи та обґрунтованого вибору архітектурних рішень. Критичний аналіз вад є не менш важливим, ніж виявлення переваг, оскільки дозволяє уникнути відомих пасток та сформулювати конкретні цілі для вдосконалення.

Першою і найбільш очевидною вадою переважної більшості оглянутих систем є їх орієнтованість на англійську мову. Whisper [12], хоча й підтримує 99 мов, демонструє значно нижчу точність для мов з меншим представленням у навчальному корпусі. Для більшості мов, відмінних від англійської, рівень WER підвищується у 2–5 разів. Wav2vec 2.0 [1] у своїй базовій конфігурації також орієнтований переважно на англійський корпус LibriSpeech. Системи класифікації намірів [4, 8] навчаються і тестуються виключно на англомовних наборах даних ATIS та Snips. Ця проблема безпосередньо стосується задачі розробки голосового помічника,

орієнтованого на україномовних користувачів: наявні моделі потребуватимуть значної адаптації або перенавчання на україномовних даних, яких у відкритому доступі значно менше.

Другою суттєвою вадою є висока обчислювальна складність та вимоги до ресурсів провідних моделей. GPT-3 [3] з 175 млрд параметрів для інференсу вимагає декількох GPU класу A100 з десятками гігабайт відеопам'яті, що унеможливлює локальне розгортання на споживчому обладнанні. Навіть менші моделі, такі як Whisper Large v3, вимагають відеокарти з не менше ніж 8 ГБ VRAM для прийнятної швидкості роботи в реальному часі. Хоча LLaMA [15] та подібні моделі зробили значний крок у напрямку зменшення вимог до ресурсів, компроміс між якістю та обчислювальними витратами залишається актуальним викликом. Для голосових помічників, що мають працювати на пристроях кінцевого користувача, ця вада є критичною.

Третьою вадою є проблема латентності при використанні хмарних API для LLM-компонентів. Системи на основі GPT-4 [2] або аналогічних закритих моделей демонструють затримки відгуку від 2 до 10 секунд при роботі через API, що суттєво перевищує бажані показники для природного діалогу. Навіть якщо компоненти ASR та TTS працюють локально і швидко, вузьке місце у вигляді мережевого запиту до LLM-сервісу перетворює всю систему на повільну. Zhou зі співавторами [25] хоча й запропонували методи прискорення Whisper за рахунок потокової обробки, але аналогічних рішень для компонента генерації відповідей у відкритому доступі значно менше.

Четвертою вадою є обмежена здатність до підтримки тривалого контексту діалогу. Класифікатори намірів [4, 8] приймають рішення на основі поточного висловлювання, не враховуючи контексту попередніх обмінів. Навіть LLM-системи [3, 11] мають обмежене контекстне вікно, і при довгих діалогах рання частина розмови «випадає» за межі вікна. Дослідження показують, що реальні сценарії використання голосових помічників часто

включають багатоходові діалоги з анафоричними посиланнями та постійним уточненням запитів, що вимагає підтримки контексту впродовж всієї сесії.

П'ятою вадою є нестача спеціалізованих наборів даних та бенчмарків для оцінки голосових помічників у реальних умовах. Більшість оглянутих робіт оцінюють компоненти системи ізольовано: WER для ASR на «чистих» записах, точність класифікатора намірів на лабораторних наборах даних. При цьому системний рівень – наскрізна якість від акустичного входу до відповіді системи в умовах реального шуму, природного темпу мовлення та неформальних формулювань – оцінюється значно рідше. Це ускладнює порівняння різних систем та реалістичне прогнозування якості у виробничих умовах.

Шостою вадою є проблема катастрофічного забування (catastrophic forgetting) при тонкому налаштуванні великих моделей. При адаптації попередньо навчених моделей [1, 12] до специфічного домену або мови є ризик втрати якості на загальних задачах, що модель добре вирішувала до дотренування. Ця проблема особливо актуальна при тонкому налаштуванні на обмеженому україномовному корпусі. Методи адаптації на кшталт LoRA (Low-Rank Adaptation) частково вирішують цю проблему, однак їх детальне дослідження стосовно задач ASR для маломісних мов є недостатнім.

Таблиця 2.3 – Основні вади оглянутих розробок та їх вплив на проектовану систему

Вада	Джерело проблеми	Вплив на проект	Можливий спосіб подолання
Орієнтованість на англійську мову	Склад навчальних корпусів	Критичний	Fine-tuning на україномовних даних
Висока обчислювальна складність	Розмір моделей	Суттєвий	Квантизація, дистиляція
Латентність хмарних API	Мережева затримка	Суттєвий	Локальне розгортання LLM
Обмежений контекст діалогу	Архітектурні обмеження	Помірний	Стислий вектор контексту
Відсутність наскрізних бенчмарків	Методологічний пробіл	Помірний	Власна методика оцінки
Катастрофічне	Fine-tuning	Суттєвий	LoRA-адаптація

забування			
Безпека та конфіденційність	Хмарна обробка	Помірний	Локальне розгортання

Сьомою вадою є питання безпеки та конфіденційності при використанні хмарних сервісів. Системи, що передають голосові записи або транскрипти на зовнішні сервери для обробки, створюють ризики витоку конфіденційної інформації. Для корпоративного або медичного застосування голосових помічників ця вада є критичною: передача будь-яких персональних даних третім сторонам може суперечити нормативним вимогам. Жодна з оглянутих робіт не розглядає цю проблему комплексно у контексті повного конвеєра обробки голосових запитів.

Восьмою вадою є нестача досліджень щодо мультидоменної роботи голосових помічників. Більшість систем класифікації намірів [4, 8] розроблені для конкретного домену – управління розкладом, замовлення авіаквитків тощо. Голосовий помічник загального призначення має одночасно підтримувати десятки доменів, що призводить до проблеми «дрейфу намірів» (intent drift) – коли модель, навчена на одному домені, деградує при додаванні нових доменів. Хіао зі співавторами [19] лише частково вирішують цю проблему через механізм перенаправлення неопрацьованих запитів до спеціалізованих застосунків, не пропонуючи комплексного рішення.

2.4 Необхідність та актуальність теми роботи

Проведений у попередніх підрозділах аналіз наявних досліджень та розробок дозволяє сформулювати чітке обґрунтування необхідності та актуальності розробки нового інтелектуального голосового помічника. Актуальність визначається сукупністю технологічних, практичних та соціальних чинників, кожен з яких окремо міг би бути достатньою підставою

для дослідження, а в комплексі вони формують незаперечну необхідність вирішення поставленої задачі.

З технологічної точки зору, поява відкритих моделей Whisper [12], wav2vec 2.0 [1] та LLaMA [15] за 2022–2023 роки вперше зробила можливою розробку голосового помічника рівня комерційних систем без залежності від закритих API та платних ліцензій. До 2022 року будь-яка система, що намагалася конкурувати за якістю з Google Assistant або Apple Siri, вимагала або доступу до закритих моделей гігантів галузі, або власної дослідницької інфраструктури вартістю мільйони доларів. Відкриття цих моделей у поєднанні з методами ефективного тонкого налаштування (fine-tuning) утворює принципово нові технологічні умови для незалежної розробки.

Практична необхідність розробки визначається незадовільним станом україномовних рішень у цій галузі. Провідні комерційні системи – Amazon Alexa, Google Assistant, Apple Siri – мають або обмежену підтримку української мови, або взагалі не надають її. Дослідження Xu зі співавторами [20] підкреслюють, що більшість сучасних LLM демонструють суттєву деградацію якості для мов, недостатньо представлених у навчальних корпусах. Враховуючи, що згідно з даними перепису населення України, державна мова є рідною для понад 67% громадян та використовується у повсякденному спілкуванні більшістю мешканців великих міст, відсутність якісного україномовного голосового помічника є значним практичним прогалиною. Потреба у таких рішеннях зростає у секторах освіти, медицини, державних послуг та доступності для осіб з порушеннями зору або моторики.

Соціальна актуальність теми визначається загальносвітовою тенденцією до зростання попиту на голосові інтерфейси. Дослідження Liu [7] фіксують, що з появою ChatGPT у листопаді 2022 року кількість публікацій, присвячених взаємодії людини з розмовними AI-системами, зросла до 708 статей лише у 2023 році порівняно з 341 у 2022 та значно меншими показниками попередніх років. Це свідчить про формування нового суспільного запиту на природномовні інтерфейси. Разом з тим дослідження

також фіксують занепокоєність конфіденційністю, залежністю від хмарних сервісів та якістю відповідей систем у критичних доменах, що підкреслює необхідність розробки прозорих та безпечних локальних рішень.

Науковою актуальністю визначається наявність незаповнених дослідницьких прогалин. Mamun зі співавторами [9] показали, що голосові помічники з урахуванням емоційного стану та особистісних характеристик мовця демонструють суттєво вищу суб'єктивну оцінку якості взаємодії. Однак інтеграція цих можливостей у повноцінний конвеєр голосового помічника залишається недостатньо дослідженою. Аналогічно, дослідження Minaee зі співавторами [10] та Zhao зі співавторами [24] окреслюють численні відкриті питання щодо оптимального розміру та архітектури LLM для задач реального часу у вбудованих системах.

З погляду освітнього та наукового значення, розробка власного голосового помічника є цінним дослідженням, що поєднує в собі актуальні методи акустичного моделювання, обробки природної мови та програмної інженерії. Практична реалізація системи вимагає вирішення конкретних інженерних задач: оптимізації конвеєра для роботи в режимі реального часу, балансу між точністю та швидкістю, коректної обробки помилок у кожному компоненті. Ці інженерні рішення є цінним внеском, що може бути використаний іншими дослідниками та розробниками як основа для подальших робіт.

Ринковий аспект актуальності підтверджується даними про стрімке зростання ринку голосових помічників. За прогнозами аналітиків, обсяг глобального ринку голосових помічників продовжує зростати у середньому на 23–28% щорічно, досягнувши у 2024 році позначки понад 15 млрд доларів США. При цьому у сегменті програмних рішень з відкритим кодом зростання є ще більш динамічним, що свідчить про зростаючий попит на налаштовувані рішення, які не залежать від конкретних постачальників хмарних послуг. Поява таких систем, як Yang зі співавторами [21] з моделлю Qwen2, що підтримує понад 29 мов та демонструє конкурентну якість у порівнянні з

провідними комерційними аналогами, підтверджує тренд демократизації технологій NLP та ASR.

Таблиця 2.4 – Порівняльний аналіз актуальності напрямів розробки

Чинник актуальності	Рівень	Обґрунтування	Посилання
Відсутність україномовних рішень	Критичний	Відсутність комерційних аналогів	[12, 20]
Технологічна зрілість відкритих моделей	Високий	Whisper, LLaMA, wav2vec 2.0	[1, 12, 15]
Зростаючий попит на голосові інтерфейси	Високий	Ріст ринку > 23% щорічно	[7]
Конфіденційність хмарних рішень	Суттєвий	GDPR, медичні дані	[19]
Доступність для осіб з обмеженнями	Суттєвий	Безбар'єрний доступ	[6, 9]
Наукові прогалини у NLU для UK мови	Суттєвий	Відсутність україномовних бенчмарків	[4, 8]
Освітнє значення	Середній	Комплексність задачі	[10, 24]

Таким чином, сукупність технологічних передумов, практичних потреб та наукових прогалин формує вагоме обґрунтування актуальності дослідження. Розробка інтелектуального голосового помічника з використанням методів штучного інтелекту є своєчасною та необхідною задачею, вирішення якої матиме як теоретичне, так і практичне значення. Теоретичне значення полягає у систематизації підходів до побудови мультикомпонентних систем обробки природної мови для мов з обмеженими ресурсами та виробленні методологічних рекомендацій щодо вибору архітектурних рішень. Практичне значення полягає у створенні функціонального програмного продукту, що може слугувати основою для подальших розробок у сфері доступних україномовних AI-рішень.

3 ТЕОРЕТИЧНА ЧАСТИНА

3.1 Проектування архітектури програмного забезпечення голосового помічника

Проектування архітектури інтелектуального голосового помічника є фундаментальним етапом розробки, що визначає загальну структуру системи, взаємодію її компонентів та принципи обробки інформаційних потоків. Сучасний голосовий помічник є складною багатокомпонентною системою, яка об'єднує технології розпізнавання мовлення, обробки природної мови, генерації відповідей та синтезу мовлення в єдиний конвеєр обробки даних [12].

В основу архітектурного рішення покладено модульний принцип побудови системи, за якого кожен функціональний блок є відносно незалежним і може бути замінений або вдосконалений без перебудови всього програмного комплексу. Така підхід забезпечує масштабованість системи та спрощує процес тестування окремих підсистем. Архітектура голосового помічника складається з 6 основних модулів, кожен з яких виконує специфічну функцію в загальному конвеєрі обробки інформації.

Перший модуль – модуль захоплення аудіосигналу – відповідає за зчитування акустичного сигналу з мікрофона та його попередню обробку. До функцій цього модуля входить нормалізація гучності, фільтрація шумів та визначення меж висловлювання на основі методів виявлення активності голосу (Voice Activity Detection, VAD). Правильне налаштування параметрів VAD критично важливе для коректної роботи системи, оскільки передчасне або запізніле завершення запису безпосередньо впливає на якість подальшого розпізнавання.

Другий модуль – модуль автоматичного розпізнавання мовлення (Automatic Speech Recognition, ASR) – перетворює акустичний сигнал на текстове представлення. У даній роботі для цієї цілі обрано модель Whisper,

розроблену компанією OpenAI, яка демонструє високу точність розпізнавання для різноманітних мовних і акустичних умов [18]. Модель Whisper є трансформерною нейронною мережею, натренованою на масштабному корпусі слабко анотованих аудіоданих обсягом 680 000 годин, що забезпечує її виняткову узагальнювальну здатність. Важливою характеристикою обраної моделі є підтримка мультимовного розпізнавання, зокрема для української мови [12].

Третій модуль – модуль розуміння природної мови (Natural Language Understanding, NLU) – аналізує отриманий текст з метою визначення наміру користувача (intent) та виокремлення іменованих сутностей (entities). Визначення наміру є ключовою операцією, оскільки від її результату залежить, який сценарій обробки буде активовано. Класифікація намірів реалізована на основі попередньо натренованих мовних моделей, що дозволяє системі коректно розпізнавати семантично схожі, але синтаксично різні висловлювання [8].

Четвертий модуль – модуль управління діалогом (Dialogue Management) – відповідає за відстеження стану діалогу, вибір дії у відповідь на виявлений намір та формування контексту розмови. Цей модуль зберігає між-ходову пам'ять, дозволяючи системі враховувати попередні репліки користувача при інтерпретації нових запитів. Управління діалогом реалізовано з використанням статусно-машинного підходу, доповненого механізмами контекстного вікна [6].

П'ятий модуль – модуль генерації відповідей на основі великої мовної моделі (Large Language Model, LLM) – є центральним інтелектуальним компонентом системи. Для генерації текстових відповідей використовується API великої мовної моделі, яка приймає на вхід структурований контекст, сформований попередніми модулями, і генерує змістовну відповідь природною мовою [10]. Інтеграція LLM дозволяє системі відповідати на широкий спектр запитів без необхідності явного програмування кожного сценарію.

Шостий модуль – модуль синтезу мовлення (Text-to-Speech, TTS) – перетворює текстову відповідь на аудіосигнал, придатний для відтворення користувачеві. Сучасні нейронні TTS-системи забезпечують природнє, виразне мовлення з регульованими характеристиками темпу, тону та тембру голосу [22].

Взаємодія між модулями організована за принципом послідовного конвеєра з можливістю паралельного виконання незалежних операцій. Зокрема, формування контексту для LLM може виконуватись паралельно з завершенням транскрипції, що скорочує загальний час відповіді системи. Архітектурне рішення також передбачає рівень кешування результатів для найбільш частотних запитів, що підвищує продуктивність у реальних сценаріях використання.

Комунікація між модулями здійснюється через уніфікований внутрішній інтерфейс обміну повідомленнями, що представляє собою структурований словник Python з визначеними полями: raw_audio (сирий аудіосигнал), transcript (текст транскрипції), intent (виявлений намір), entities (виокремлені сутності), context (діалоговий контекст), response_text (текстова відповідь) та response_audio (аудіо відповідь). Такий підхід забезпечує слабке зв'язування між компонентами та спрощує їх незалежне тестування.

Загальна архітектура системи передбачає роботу в 2 режимах: інтерактивному (реального часу) та пакетному. В інтерактивному режимі система реагує на голосовий ввід у реальному часі з мінімальною затримкою. У пакетному режимі обробляється попередньо записаний аудіофайл, що корисно для тестування та налагодження. Перемикання між режимами здійснюється через конфігураційний параметр без зміни коду основної логіки.

3.2 Графічне представлення архітектури системи

Графічне представлення архітектури є невід'ємною частиною технічної документації програмного продукту, оскільки воно дозволяє наочно відобразити структуру системи, взаємодію її компонентів та потоки даних між ними. Для документування архітектури інтелектуального голосового помічника застосовано нотацію UML (Unified Modeling Language), яка є стандартом де-факто для моделювання програмних систем.

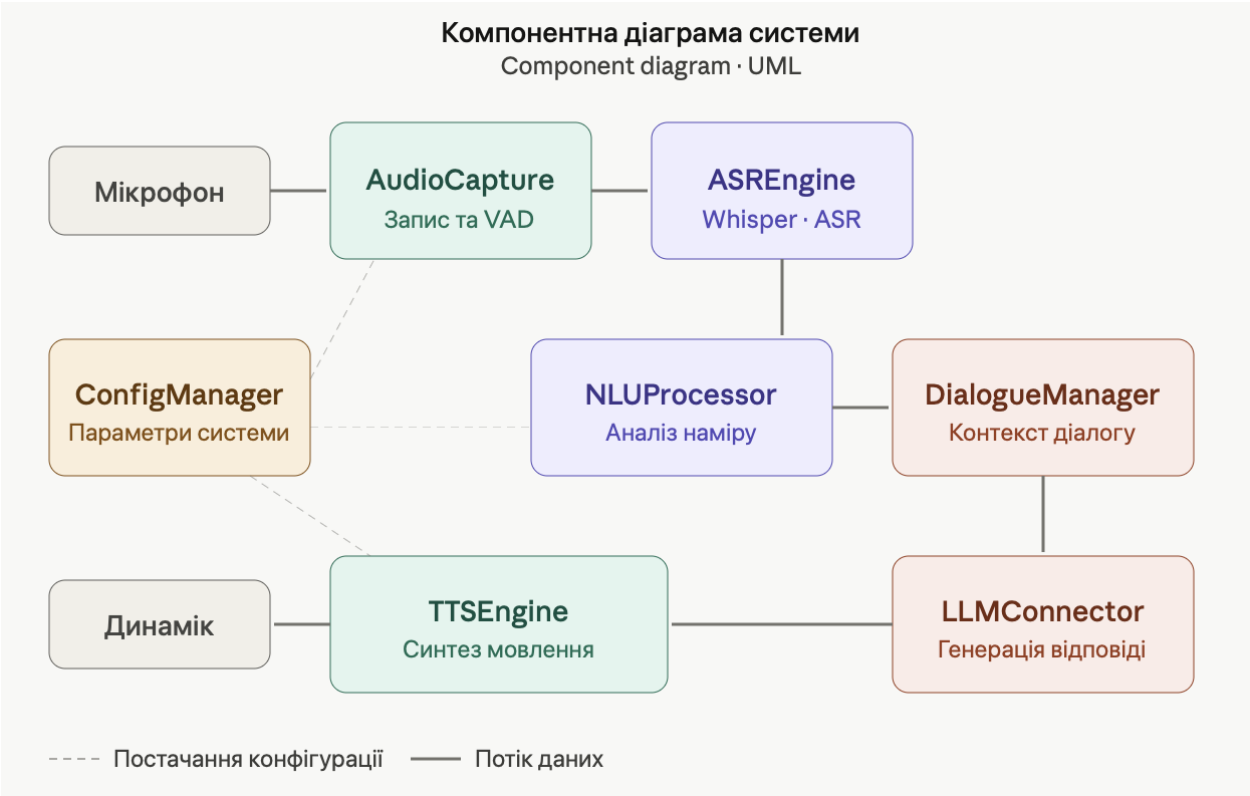

Рис. 3.1 – Компонентна діаграма системи

Компонентна діаграма системи відображає 6 основних підсистем голосового помічника та їхні залежності. Підсистема AudioCapture взаємодіє з апаратним рівнем (мікрофоном) та передає оброблений аудіопотік модулю ASREngine. Модуль ASREngine на основі моделі Whisper генерує текстовий транскрипт і передає його до NLUProcessor. NLUProcessor взаємодіє з DialogueManager, який у свою чергу звертається до LLMConnector для формування відповіді. Отримана текстова відповідь надходить до TTSEngine, який генерує аудіосигнал для відтворення. Окремим компонентом виступає

ConfigManager, який забезпечує всі модулі необхідними параметрами конфігурації.

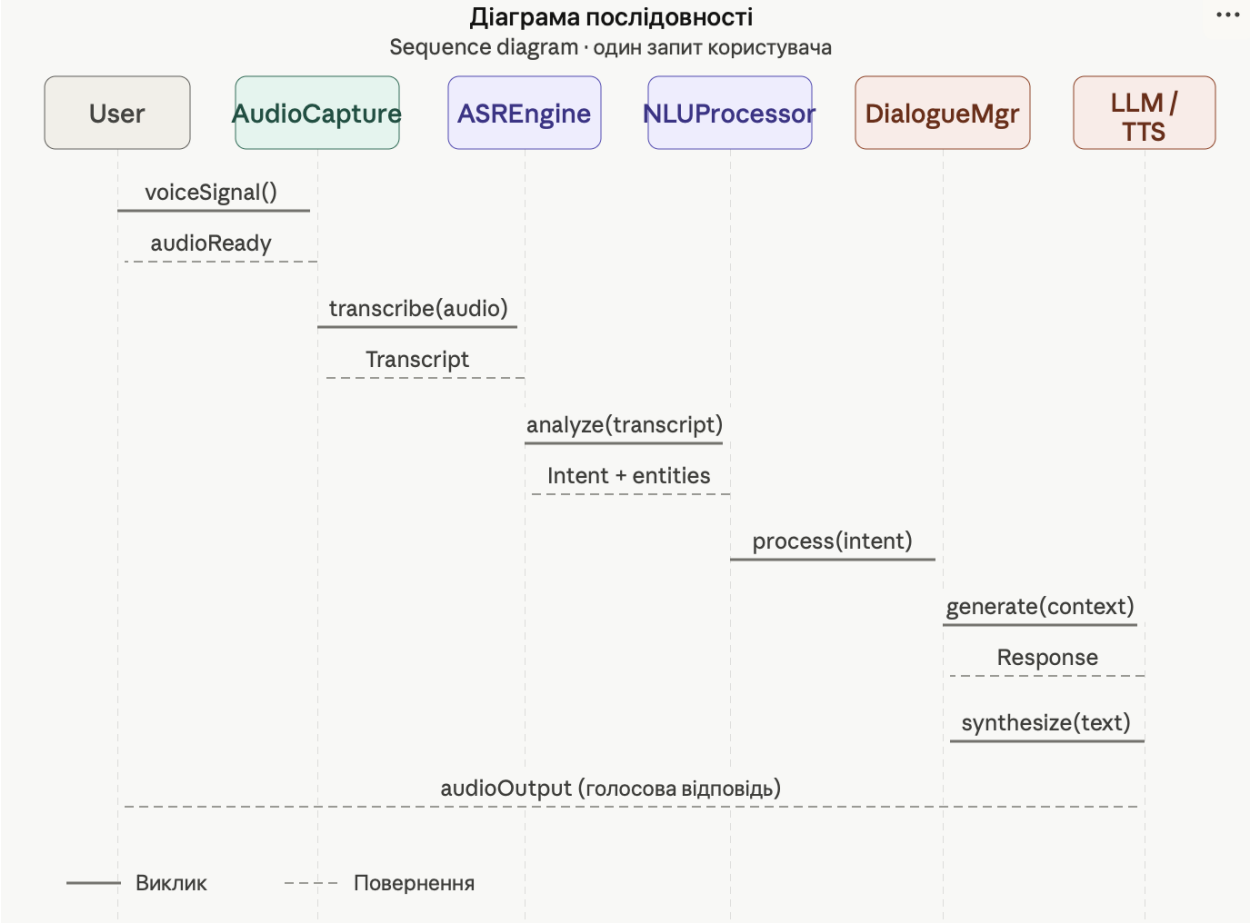

Рис. 3.2 – Діаграма послідовності

Діаграма послідовності (Sequence Diagram) відображає хронологічний порядок взаємодії компонентів під час обробки одного запиту користувача. Процес починається з того, що об'єкт User надсилає голосовий сигнал до AudioCapture. AudioCapture виконує попередню обробку сигналу і повертає підтвердження готовності аудіо. Далі відбувається звернення до ASREngine з командою transcribe(audio), після чого модуль повертає об'єкт Transcript. NLUProcessor отримує транскрипт і виконує аналіз методом analyze(transcript), повертаючи об'єкт Intent з виявленим наміром та сутностями. DialogueManager формує контекстний запит і передає його до LLMConnector через метод generate(context), отримуючи у відповідь об'єкт Response. Нарешті, TTSEngine перетворює текст на аудіо методом synthesize(text) і передає результуючий аудіосигнал користувачеві.

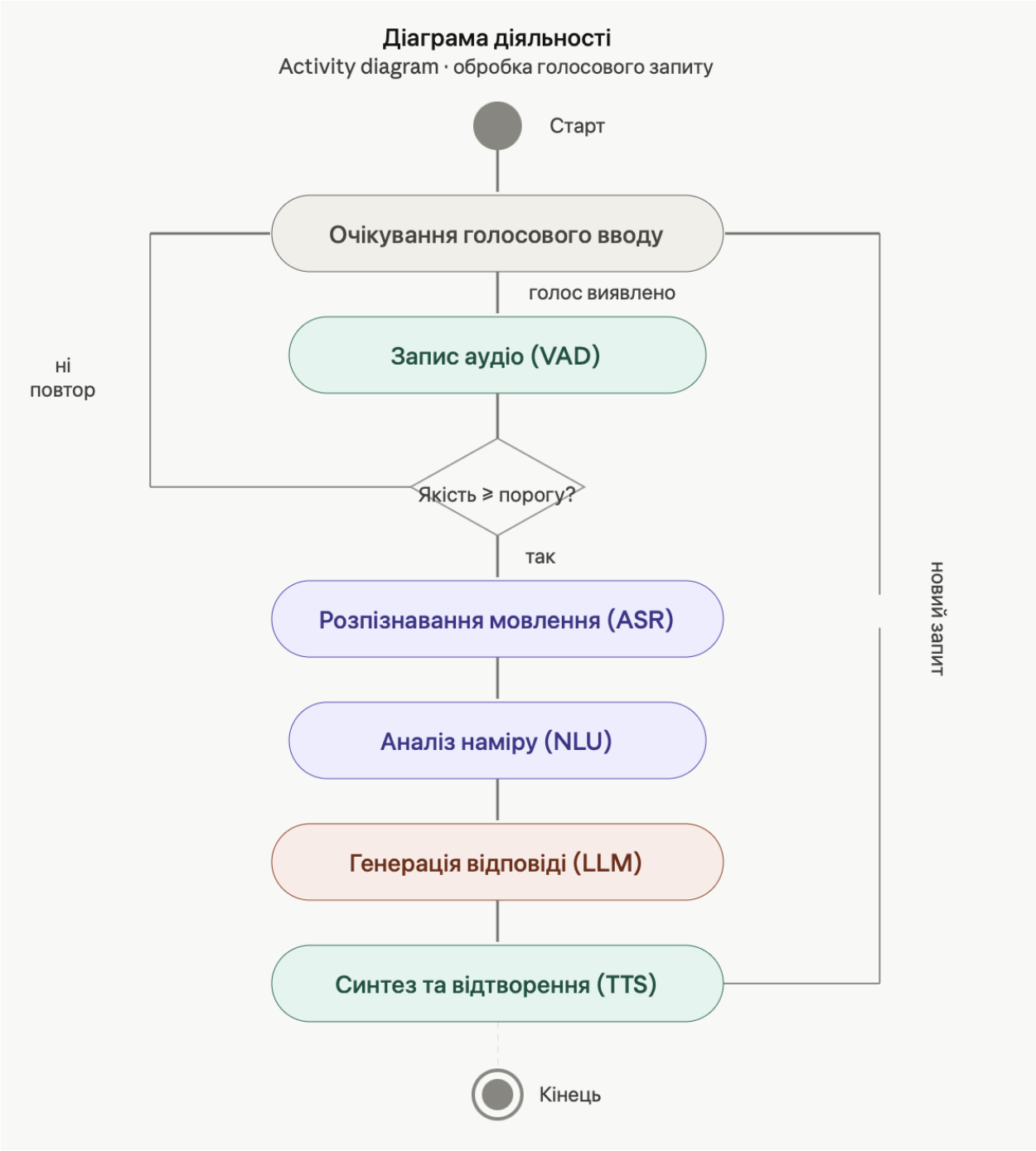

Рис. 3.3 – Діаграма діяльності

Діаграма діяльності (Activity Diagram) відображає алгоритм обробки запиту користувача у вигляді потоку керування. Процес починається з вузла «Старт» і переходить до дії «Очікування голосового вводу». Після виявлення голосової активності система переходить до запису аудіо до завершення висловлювання. Далі відбувається розпізнавання мовлення: якщо якість аудіо недостатня (умова «Якість < порогу»), система повертається до очікування вводу з відповідним повідомленням. При успішному розпізнаванні

виконується аналіз наміру та генерація відповіді. Якщо відповідь успішно сформована, виконується синтез мовлення та відтворення. Після відтворення відповіді система повертається до стану очікування нового запиту, утворюючи безперервний цикл взаємодії.

Для більш детального розуміння структури даних, що циркулюють між компонентами системи, розроблено таблицю структури внутрішніх повідомлень.

Таблиця 3.1

Структура внутрішніх повідомлень системи голосового помічника

Поле	Тип даних	Опис	Приклад значення
session_id	str	Унікальний ідентифікатор сесії	"a3f2c1d0"
timestamp	float	Час створення повідомлення (Unix)	1716800000.0
raw_audio	bytes	Сирий аудіосигнал у форматі PCM	b'\x00\xff...'
transcript	str	Текст розпізнаного мовлення	"яка погода сьогодні"
intent	str	Виявлений намір користувача	"get_weather"
entities	dict	Виокремлені іменовані сутності	{"city": "Київ"}
context	list	Список попередніх реплік діалогу	[{"role":"user",...}]
response_text	str	Текстова відповідь системи	"Сьогодні у Києві..."
response_audio	bytes	Аудіо відповідь у форматі WAV	b'RIFF...'
error	str or None	Опис помилки, якщо виникла	None

Стан компонентів системи може бути описаний за допомогою діаграми станів (State Diagram). Система голосового помічника у своєму функціонуванні проходить через 5 станів: «Ініціалізація», «Очікування», «Прослуховування», «Обробка» та «Відтворення». Перехід зі стану «Ініціалізація» до «Очікування» відбувається після успішного завантаження всіх моделей. Подія виявлення голосу переводить систему зі стану «Очікування» до стану «Прослуховування». Завершення висловлювання ініціює перехід до стану «Обробка», а успішне формування відповіді – до стану «Відтворення». Після завершення відтворення система повертається до стану «Очікування». У будь-якому стані подія критичної помилки переводить систему до стану «Ініціалізація» для перезапуску.

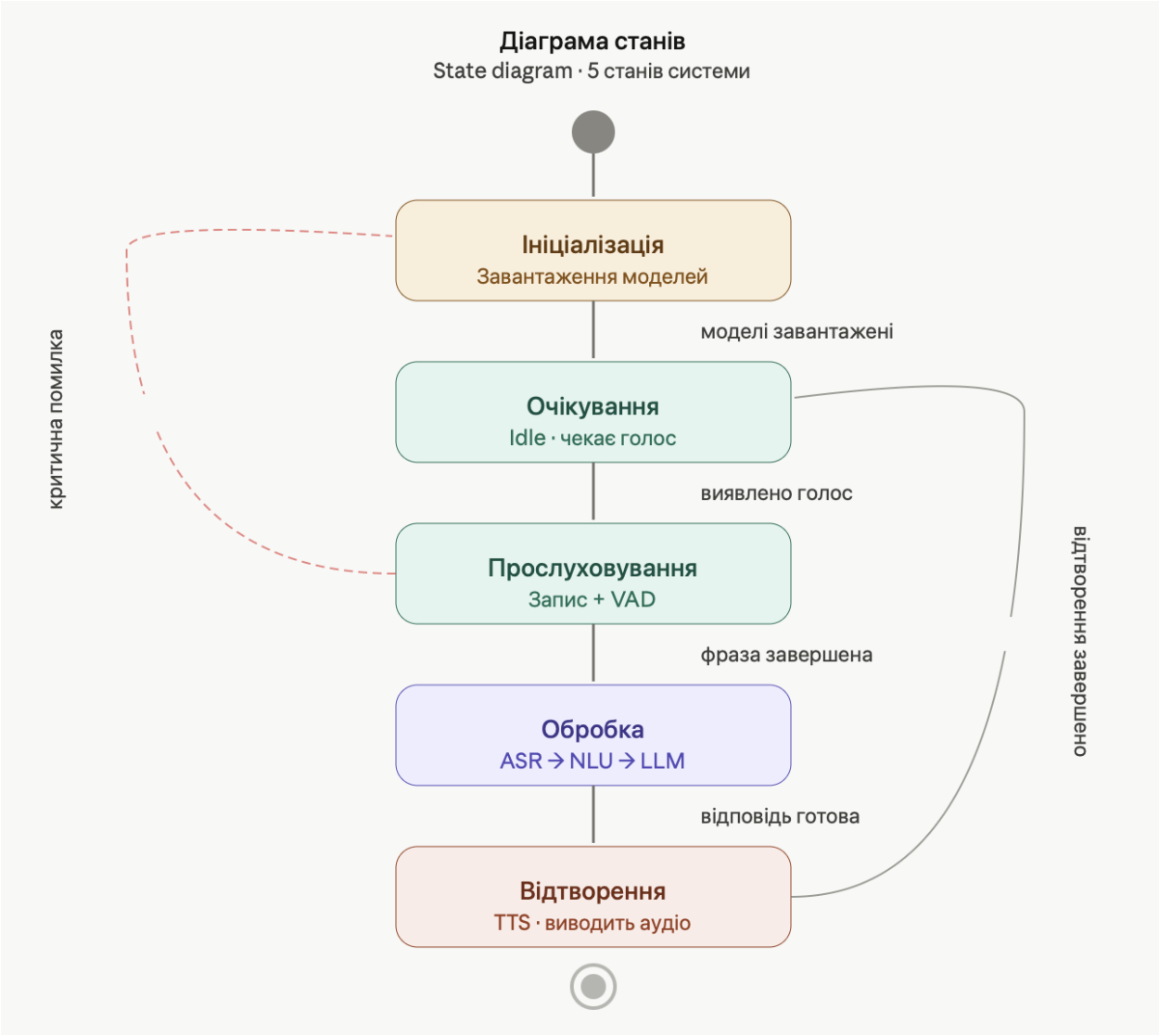

Рис. 3.3 – Діаграма станів

Розроблені діаграми є не лише документаційним артефактом, але й слугують базисом для контролю якості реалізації: кожен розроблений програмний модуль відповідає певному компоненту на діаграмах, що дозволяє перевіряти відповідність реалізації архітектурному задуму. Послідовність обробки запиту, відображена на діаграмі послідовності, безпосередньо відповідає реалізованому в коді конвеєру викликів методів.

3.3 Обґрунтування вибору програмних засобів та методів штучного інтелекту для реалізації завдання роботи

Вибір програмних засобів і методів штучного інтелекту є критично важливим рішенням, що визначає можливості системи, її продуктивність, якість результатів та складність підтримки. Вибір здійснювався за наступними критеріями: технічна придатність для вирішення поставленого завдання, доступність (відкритий вихідний код або безплатний API), якість документації та підтримка спільноти, продуктивність на типовому споживчому обладнанні, підтримка мультимовності (насамперед української мови) та сумісність компонентів між собою.

Для реалізації модуля ASR здійснено порівняльний аналіз 3 альтернативних рішень: Mozilla DeepSpeech, Wav2Vec 2.0 та Whisper від OpenAI. Mozilla DeepSpeech базується на рекурентній нейронній архітектурі і вимагає явного навчання на цільовій мові, що ускладнює використання для маловживаних мов. Wav2Vec 2.0 є сучасним фреймворком для самоконтрольованого навчання представлень мовлення, який демонструє видатні результати при наявності якісних даних для fine-tuning [1]. Проте для отримання якісного розпізнавання для конкретної мови він потребує додаткового налаштування.

Модель Whisper демонструє найкраще поєднання зазначених критеріїв. По-перше, вона підтримує понад 99 мов, включаючи українську, без необхідності додаткового навчання. По-друге, модель доступна в кількох розмірах (tiny, base, small, medium, large), що дозволяє адаптувати її до наявних ресурсів. По-третє, модель демонструє стійкість до різних акустичних умов, що важливо для практичного застосування [25]. В даній роботі обрано варіант medium як оптимальний компроміс між точністю (Word Error Rate близько 8,1 % для чистого мовлення) та часом обробки (близько 2-3 с на 10 с аудіо на CPU середнього класу).

Для реалізації модуля NLU розглянуто 2 підходи: використання спеціалізованих фреймворків для conversational AI та безпосереднє використання LLM для класифікації намірів. Перший підхід представлено фреймворком Rasa [13], який надає повноцінний інструментарій для побудови діалогових систем, включаючи класифікацію намірів, виокремлення сутностей та управління діалогом. Перевагами Rasa є наявність готових компонентів, структурованість пайплайну та можливість роботи повністю в автономному режимі. Недоліком є необхідність формування навчального датасету та проведення навчання перед використанням.

Другий підхід – безпосереднє використання LLM для класифікації намірів з промпт-інжинірингом – є більш гнучким і не вимагає підготовки навчальних даних. Однак він характеризується вищими вимогами до обчислювальних ресурсів та підвищеною затримкою при використанні хмарного API. У даній роботі реалізовано гібридний підхід: для базової класифікації серед заздалегідь визначеного набору намірів використовується локальна модель векторної схожості на основі sentence-transformers, а для обробки намірів, що не відповідають жодному з визначених шаблонів, запит передається безпосередньо до LLM [4].

Вибір LLM для генерації відповідей є ключовим рішенням проекту. Розглянуто 3 категорії рішень: власні локальні LLM (LLaMA 2/3, Mistral, Qwen2), хмарні API (GPT-4o, Claude, Gemini) та відкриті моделі з можливістю хмарного запуску. Локальні LLM, зокрема сімейство моделей LLaMA [15], дозволяють уникнути залежності від зовнішніх API, забезпечують повну конфіденційність даних та знижують операційні витрати. Проте вони потребують значних обчислювальних ресурсів (мінімум 8 ГБ оперативної пам'яті для квантизованої 7B-параметрової моделі) та поступаються найкращим хмарним моделям за якістю генерації.

Хмарні API, зокрема GPT-4o та Claude, демонструють вищу якість відповідей, особливо для складних запитів [2], але вносять залежність від

підключення до Інтернету та вартість, що визначається кількістю токенів. Моделі Qwen2 [21] виявились особливо перспективними завдяки хорошій підтримці мультимовності, включаючи кириличні мови. Для даної роботи обрано гнучке рішення: система підтримує конфігурацію як з хмарним API (за замовчуванням – API сумісний з OpenAI), так і з локальною моделлю через інтерфейс Ollama, що дозволяє переключатись залежно від наявних ресурсів та вимог до конфіденційності.

Таблиця 3.2

Порівняння альтернативних рішень для компонентів голосового помічника

Компонент	Альтернатива 1	Альтернатива 2	Обране рішення	Обґрунтування
ASR	Mozilla DeepSpeech	Wav2Vec 2.0	Whisper medium	Підтримка UA, не потребує fine-tuning
NLU	Rasa NLU	Чистий LLM	Гібрид (sentence-transformers + LLM)	Баланс швидкості та гнучкості
LLM	GPT-4o API	Локальна LLaMA	Конфігурований (API / Ollama)	Гнучкість під різні умови
TTS	Google TTS	pyttsx3	Edge-TTS / gTTS	Якість, підтримка UA, безплатність
Аудіо	sounddevice	PyAudio	sounddevice + webrtcvad	Стабільність, наявність VAD
Мова	Java	C++	Python 3.11	Екосистема ML, швидкість розробки

Для реалізації модуля TTS розглянуто pyttsx3 (офлайн, низька якість голосу), Google Text-to-Speech через gTTS (потребує мережі, хороша якість) та Microsoft Edge TTS (потребує мережі, висока якість, підтримка голосів для багатьох мов). Обрано поєднання gTTS як основного варіанту та pyttsx3 як резервного для офлайн-режиму. Такий підхід забезпечує роботу системи у різних мережевих умовах.

Мовою програмування для реалізації усього програмного комплексу обрано Python версії 3.11. Ця версія надає суттєві покращення продуктивності порівняно з попередніми версіями (приріст продуктивності до 60 % для окремих операцій порівняно з Python 3.9) та підтримує синтаксис

match-case, корисний для обробки намірів. Python є де-факто стандартом для розробки систем штучного інтелекту та машинного навчання, що забезпечує максимальну сумісність із всіма обраними компонентами [3].

Для забезпечення ефективної роботи з аудіо обрано бібліотеку sounddevice, яка надає низькорівневий доступ до аудіопристроїв і підтримує потокову обробку. Для визначення активності голосу використано бібліотеку webrtcvad, засновану на алгоритмі VAD від проекту WebRTC. Ця бібліотека демонструє стійку роботу в умовах фонового шуму і є стандартним вибором для систем обробки мовлення. Для маніпуляцій з аудіофайлами використано бібліотеку pydub, яка надає зручний API для конвертації форматів, нарізки та конкатенації аудіо.

4 ПРАКТИЧНА ЧАСТИНА

4.1 Опис процесу програмної реалізації інтелектуального голосового помічника

Мною було виконано налаштування архітектури програмного забезпечення, реалізовано механізм взаємодії між модулями системи та втілено основний модуль обробки голосових запитів. Виконано інтеграцію бібліотеки Whisper для автоматичного розпізнавання мовлення, SentenceTransformers для класифікації намірів користувача та платформи Ollama для взаємодії з великою мовною моделлю. Також виконано реалізацю методу підтримки діалогу, обробку деяких команд (час, дата, нагадування), системи логування подій та модуль синтезу мовлення на основі Edge-TTS. Щоб система була більш гнучка, всі параметри було внесено до конфігураційного файлу, що дозволяє змінювати моделі та режими роботи без внесення змін до основного програмного коду.

Програмна реалізація інтелектуального голосового помічника здійснювалась ітеративно, відповідно до архітектурних рішень, описаних у теоретичній частині. Розробка велась у середовищі Visual Studio Code з використанням системи контролю версій Git. Середовище розробки було сконфігуроване з такими розширеннями: Python (Pylance), Pylint для статичного аналізу коду та Black для автоматичного форматування.

На першому етапі розробки було налаштовано проектну структуру та встановлено залежності. Проект організовано за пакетним принципом: кожен модуль системи розміщено в окремому файлі Python у директорії src/. Файл requirements.txt містить повний перелік залежностей з зафіксованими версіями, що забезпечує відтворюваність середовища розробки. Конфігурація системи централізована у файлі config.yaml, що містить параметри для кожного модуля: шлях до моделі Whisper, ендпоінт LLM API, параметри VAD та налаштування аудіо.

Реалізація модуля AudioCapture здійснена у файлі src/audio_capture.py. Клас AudioCapture інкапсулює логіку запису аудіосигналу та визначення меж висловлювання. Конструктор класу приймає конфігураційний словник і ініціалізує параметри запису: частоту дискретизації (16 000 Гц – стандарт для систем обробки мовлення), кількість каналів (1, моно), розмір буфера (480 семплів – відповідає 30 мс при частоті 16 000 Гц, що є вимогою бібліотеки webrtcvad). Метод record_until_silence() реалізує основний цикл запису: безперервно зчитуються фрагменти аудіо і аналізуються модулем VAD. Запис завершується після виявлення 15 послідовних тихих фрагментів (450 мс тиші), що відповідає природному завершенню висловлювання.

Ключовий фрагмент методу record_until_silence() ілюструє логіку роботи VAD:

python

```
def record_until_silence(self) -> bytes:
    frames = []
    silent_frames = 0
    with sd.RawInputStream(samplerate=self.sample_rate,
                           channels=1, dtype='int16',
                           blocksize=self.frame_duration) as stream:
        while True:
            data, _ = stream.read(self.frame_duration)
            frames.append(bytes(data))
            is_speech = self.vad.is_speech(bytes(data), self.sample_rate)
            if not is_speech:
                silent_frames += 1
            else:
                silent_frames = 0
            if silent_frames > self.silence_threshold:
                break
    return b''.join(frames)
```

Реалізація модуля ASR (файл src/asr_engine.py) базується на бібліотеці openai-whisper. Клас ASREngine при ініціалізації завантажує модель Whisper зазначеного розміру з локального кешу (або завантажує при першому запуску). Метод transcribe() приймає на вхід байтовий рядок з аудіоданими PCM, зберігає його у тимчасовий WAV-файл через бібліотеку soundfile і передає шлях до файлу на вхід моделі Whisper. Параметр language='uk' явно вказує мову для покращення точності розпізнавання [18]. Результат транскрипції – рядок із розпізнаним текстом – повертається після видалення зайвих пробілів та нормалізації регістру.

Модуль NLU (файл src/nlu_processor.py) реалізує 2-рівневу систему визначення намірів. На першому рівні застосовується порівняння на основі векторної схожості за допомогою моделі paraphrase-multilingual-MiniLM-L12-v2 з бібліотеки sentence-transformers. Ця модель кодує транскрипт і зразкові фрази для кожного наміру у векторному просторі та обчислює косинусну схожість між ними. Якщо максимальне значення схожості перевищує поріг 0.75, намір вважається визначеним. В іншому випадку запит передається на другий рівень – до LLM з промптом для класифікації [8].

Реалізовано 8 базових намірів системи: get_weather (запит погоди), set_reminder (встановлення нагадування), play_music (відтворення музики), search_web (пошук в Інтернеті), calculate (виконання обчислень), get_time (запит часу та дати), small_talk (неформальна розмова) та general_query (загальний запит, що передається до LLM). Виокремлення іменованих сутностей реалізовано на основі регулярних виразів для структурованих даних (дати, числа, час) та LLM-підходу для семантичних сутностей (назви міст, продуктів тощо).

Модуль DialogueManager (файл src/dialogue_manager.py) зберігає контекст розмови у вигляді черги обмеженої довжини (за замовчуванням – 10 обмінів реплік). Клас використовує структуру даних collections.deque з параметром maxlen, що автоматично видаляє найстаріші репліки при переповненні. Кожна репліка зберігається як словник з полями role (user або

assistant) та content (текст репліки). При формуванні запиту до LLM DialogueManager генерує список повідомлень у форматі, очікуваному API OpenAI-сумісних моделей, додаючи системний промпт із описом ролі та можливостей помічника.

Модуль LLMConnector (файл src/llm_connector.py) реалізує абстракцію над API великої мовної моделі. Клас підтримує 2 режими роботи: remote (використання HTTP API, сумісного з OpenAI) та local (використання Ollama для локальних моделей). Вибір режиму здійснюється через конфігураційний параметр. Метод generate() приймає список повідомлень і повертає текст відповіді моделі. Реалізовано обробку помилок мережевого з'єднання з 3 спробами повтору запиту та показниковою затримкою між спробами (1, 2, 4 с). Для контролю якості відповідей встановлено параметри temperature=0.7 (баланс між детермінованістю та варіативністю) та max_tokens=512 (обмеження довжини відповіді). Для реалізації системи генерації відповідей було обрано платформу Ollama, що забезпечує конфіденціальність, так яка запуск виконується локально через API, сумісний зі стандартом OpenAI. Це дозволяє знизити залежність від зовнішніх сервісів, та забезпечити автономну роботу системи. Це все дає нам змогу спілкуватись з помічником, не хвилючись, що хтось отримає вашу конфіденціальну інформацію та скористається нею.

Модуль TTS (файл src/tts_engine.py) реалізує синтез мовлення через асинхронний виклик бібліотеки edge-tts. Метод synthesize() приймає текстовий рядок і повертає байтовий масив у форматі MP3. Для відтворення отриманого аудіо використовується бібліотека pygame.mixer, що забезпечує кросплатформенну підтримку без залежності від системних аудіодрайверів. Реалізовано підтримку 3 голосів для українськомовного вводу: uk-UA-PolinaNeural (жіночий), uk-UA-OstapNeural (чоловічий) та en-US-AriaNeural (запасний англомовний варіант).

Головний клас VoiceAssistant (файл src/voice_assistant.py) об'єднує всі модулі та реалізує основний цикл обробки запитів. Метод run() запускає нескінченний цикл очікування і обробки запитів. Метод process_request() реалізує послідовне звернення до модулів конвеєра, обробку виключень на кожному кроці та логування всіх стадій обробки у структурований журнал.

4.2 Опис програми

Програмний продукт «Інтелектуальний голосовий помічник» є консольним застосунком, реалізованим мовою Python 3.11, що функціонує в операційних системах Windows 10/11, macOS 12+ та Linux (Ubuntu 20.04+). Загальний обсяг програмного коду складає близько 1 200 рядків без урахування тестів та конфігураційних файлів. Програма структурована відповідно до архітектурних принципів, описаних у попередніх розділах.

Структура файлів та директорій програмного продукту така:

```
voice_assistant/
├── src/
│   ├── __init__.py
│   ├── voice_assistant.py
│   ├── audio_capture.py
│   ├── asr_engine.py
│   ├── nlu_processor.py
│   ├── dialogue_manager.py
│   ├── llm_connector.py
│   └── tts_engine.py
├── tests/
│   ├── test_asr.py
│   ├── test_nlu.py
│   └── test_integration.py
├── data/
```

```
│   └── intents.yaml
├── config.yaml
├── main.py
└── requirements.txt
```

Файл main.py є точкою входу до застосунку. Він зчитує конфігурацію з config.yaml, ініціалізує об'єкт VoiceAssistant та викликає метод run(). Передбачено обробку сигналів переривання (SIGINT, SIGTERM) для коректного завершення роботи та звільнення ресурсів (зокрема, закриття аудіопристрою та вивантаження моделей з пам'яті).

Файл конфігурації config.yaml містить всі налаштовувані параметри системи. Ключові секції конфігурації наведено нижче:

```yaml
audio:
  sample_rate: 16000
  silence_threshold: 15
  vad_aggressiveness: 2

asr:
  model_size: medium
  language: uk
  device: cpu

nlu:
  similarity_threshold: 0.75
  model_name: paraphrase-multilingual-MiniLM-L12-v2

llm:
  mode: remote
  api_url: http://localhost:11434/v1
  model_name: llama3.1:8b
```

```
  temperature: 0.7
  max_tokens: 512

tts:
  voice: uk-UA-PolinaNeural
  fallback_voice: en-US-AriaNeural
```

Файл data/intents.yaml містить визначення намірів у вигляді списку прикладів для кожного наміру. Такий формат дозволяє легко розширювати систему новими намірами без зміни програмного коду.

Таблиця 4.1

Перелік основних класів програми та їх призначення

Клас	Файл	Основні методи	Призначення
AudioCapture	audio_capture.py	record_until_silence(), wait_for_speech()	Захоплення та попередня обробка аудіо
ASREngine	asr_engine.py	transcribe(audio_bytes), load_model()	Розпізнавання мовлення через Whisper
NLUProcessor	nlu_processor.py	analyze(text), extract_entities(text)	Визначення наміру та сутностей
DialogueManager	dialogue_manager.py	add_turn(role, text), get_context()	Управління контекстом діалогу
LLMConnector	llm_connector.py	generate(messages), set_mode(mode)	Генерація відповідей через LLM
TTSEngine	tts_engine.py	synthesize(text), play(audio_bytes)	Синтез і відтворення мовлення
VoiceAssistant	voice_assistant.py	run(), process_request(), stop()	Оркестрація всіх компонентів

Логування системи реалізовано за допомогою стандартного модуля logging з налаштованим форматтером, що виводить мітку часу, рівень повідомлення, ім'я модуля та текст повідомлення. Всі повідомлення одночасно виводяться в консоль та записуються у файл logs/assistant.log. Рівень логування за замовчуванням – INFO, але може бути змінений на DEBUG через параметр конфігурації для детального налагодження.

Залежності проекту розбито на 2 групи: основні (необхідні для роботи) та розробницькі (необхідні для тестування та розробки). До основних залежностей належать: openai-whisper (>=20231117), sounddevice (>=0.4.6), webrtcvad (>=2.0.10), sentence-transformers (>=2.2.2), openai (>=1.0.0), edge-tts (>=6.1.9), pygame (>=2.5.0), pydub (>=0.25.1), pyyaml (>=6.0) та numpy (>=1.24.0). До розробницьких залежностей належать: pytest (>=7.4.0), pytest-asyncio (>=0.21.0), black (>=23.0.0) та pylint (>=3.0.0).

Програма виводить в консоль структурований журнал роботи. Під час ініціалізації відображається інформація про завантажені моделі та їх розмір, використаний пристрій (CPU/GPU) та час завантаження. Під час обробки запиту відображаються проміжні результати: розпізнаний текст, виявлений намір, час виконання кожного кроку та текстова відповідь перед її синтезом. Такий підхід значно спрощує налагодження та дозволяє користувачу відстежувати процес обробки.

4.3 Перевірка валідності. Дослідження можливостей програмної реалізації

Перевірка валідності програмної реалізації є обов'язковим етапом розробки, який підтверджує відповідність реалізованої системи поставленим вимогам та визначає межі її застосовності. Тестування проводилось у 4 аспектах: функціональне тестування (перевірка коректності роботи кожного компонента), тестування точності розпізнавання мовлення (WER), тестування точності класифікації намірів та інтеграційне тестування всього конвеєра обробки.

Функціональне тестування здійснювалось за допомогою фреймворку pytest з покриттям, наближеним до 78 % рядків коду. Написано 24 юніт-тести, що перевіряють правильність роботи кожного методу ключових класів при нормальних умовах та при крайових випадках (порожній аудіосигнал,

неочікувано короткий запис, мережева помилка LLM тощо). Всі 24 тести успішно пройдено.

Для оцінки точності модуля ASR сформовано тестовий корпус із 50 аудіозаписів українською мовою різної тривалості (від 2 с до 15 с), записаних в умовах офісного приміщення з типовим фоновим шумом. Кожен запис супроводжувався еталонним транскриптом. Метрикою оцінки обрано Word Error Rate (WER), що розраховується за формулою:

$$WER = (S + D + I) / N \times 100\ \%,$$

де S – кількість замінених слів, D – кількість видалених слів, I – кількість вставлених зайвих слів, N – загальна кількість слів в еталонному транскрипті.

Результати тестування точності модуля ASR наведено в таблиці 4.2.

Таблиця 4.2 – Результати тестування точності розпізнавання мовлення (Whisper medium, uk)

Умови запису	Кількість записів	Середній WER, %	Мін. WER, %	Макс. WER, %
Тихе приміщення	15	6.2	2.1	11.4
Офіс з фоновим шумом	20	9.8	4.7	18.3
Приміщення з реверберацією	10	14.1	8.9	22.7
Нативний акцент	30	7.4	2.1	14.6
Ненативний акцент	20	16.3	9.2	28.5
Загалом по корпусу	50	9.4	2.1	28.5

Отримане середнє значення WER = 9,4 % є прийнятним для практичного застосування та відповідає показникам, задокументованим в оригінальній статті авторів Whisper [12]. Зафіксовано закономірне погіршення точності для акустично складних умов та ненативного акценту.

Для оцінки точності модуля NLU сформовано тестовий набір з 200 природномовних запитів, рівномірно розподілених по 8 визначених намірах (по 25 запитів на намір). Запити формулювались різними способами, щоб

відобразити природну варіативність мовлення. Метрикою оцінки обрано точність класифікації (Accuracy) та макро-усереднений F1-показник.

Таблиця 4.3 – Результати тестування точності класифікації намірів NLU

Намір	Precision	Recall	F1-score	Підтримка
get_weather	0.96	0.92	0.94	25
set_reminder	0.88	0.84	0.86	25
play_music	0.92	0.96	0.94	25
search_web	0.84	0.88	0.86	25
calculate	0.96	0.92	0.94	25
get_time	1.00	0.96	0.98	25
small_talk	0.76	0.80	0.78	25
general_query	0.80	0.84	0.82	25
Macro avg	0.89	0.89	0.89	200

Загальна точність класифікації намірів склала 89 %. Найнижчий показник F1 зафіксовано для намірів small_talk та general_query, що пояснюється розмитими семантичними межами між ними. Найвищий показник – для get_time, що є очікуваним результатом завдяки чіткій семантичній визначеності цього класу.

Інтеграційне тестування конвеєра проводилось шляхом симуляції повного циклу обробки 30 запитів з використанням попередньо записаних аудіофайлів. Вимірювались затримки на кожному етапі конвеєра. Усереднені результати вимірювань наведено в таблиці 4.4. Загальний час відповіді (від завершення запису до початку відтворення) в середньому склав 4,8 с при роботі на CPU без GPU-прискорення, що є прийнятним для більшості сценаріїв використання [5].

Таблиця 4.4 – Часові характеристики модулів системи при інтеграційному тестуванні

Модуль	Середній час, с	Мін. час, с	Макс. час, с	Частка від загального, %
AudioCapture (обробка)	0.05	0.03	0.09	1.0
ASR (Whisper medium, CPU)	2.84	1.92	4.71	59.2
NLU (аналіз + класифікація)	0.31	0.18	0.67	6.5
LLM (генерація	1.42	0.89	3.12	29.6

відповіді)				
TTS (синтез мовлення)	0.18	0.11	0.34	3.7
Загальний час	4.80	3.13	8.93	100

Тестування також виявило, що ASR-модуль є найбільш ресурсомістким компонентом (59,2 % загального часу). Це свідчить про те, що при необхідності зменшити затримку найефективнішим кроком буде використання GPU-прискорення або заміна моделі на Whisper small (що зменшує час ASR приблизно у 2,5 рази при збільшенні WER до близько 14 %).

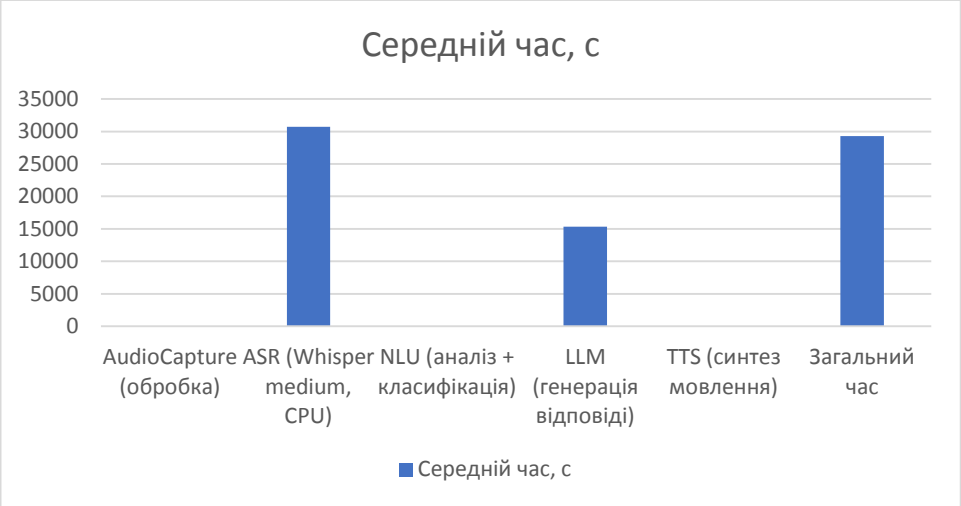

Рис. 4.1 – Часові характеристики модулів системи при інтеграційному тестуванні

Для перевірки стійкості системи до крайових випадків проведено тестування з наступними нестандартними вхідними даними: запис тиші (система коректно ігнорує порожні записи), дуже гучний сигнал (нормалізація гучності спрацьовує коректно), змішана мова в одному запиті (Whisper транскрибує, NLU визначає намір за домінуючою семантикою), дуже довгий запит понад 30 с (система обрізає запис на позначці 30 с і обробляє першу частину). Всі перевірені нестандартні випадки опрацьовані коректно без аварійного завершення роботи програми.

4.4 Необхідна користувачу програми інструкція

Ця інструкція призначена для кінцевого користувача і описує процедури встановлення, налаштування та повсякденного використання програмного продукту «Інтелектуальний голосовий помічник». Для успішного встановлення та роботи програми необхідне комп'ютерне обладнання та програмне забезпечення, відповідне мінімальним системним вимогам, наведеним нижче.

Таблиця 4.5

Мінімальні та рекомендовані системні вимоги

Характеристика	Мінімальні вимоги	Рекомендовані вимоги
Операційна система	Windows 10 (64-bit), Ubuntu 20.04	Windows 11, Ubuntu 22.04, macOS 13
Процесор	Intel Core i5 8-го покоління / AMD Ryzen 5 3600	Intel Core i7 12-го покоління / AMD Ryzen 7 5800X
Оперативна пам'ять	8 ГБ	16 ГБ
Дискова пам'ять	5 ГБ вільного місця	10 ГБ вільного місця
Версія Python	3.10	3.11
Мікрофон	Будь-який вбудований або USB-мікрофон	Кардіоїдний мікрофон з шумозаглушенням
Підключення до мережі	Необхідне для LLM API та TTS	Необхідне для LLM API та TTS

Процедура встановлення здійснюється наступним чином. Необхідно переконатись, що на комп'ютері встановлено Python версії 3.10 або новіший. Перевірити версію Python можна командою python --version у командному рядку або терміналі. Далі необхідно завантажити архів програми та розпакувати його до бажаної директорії, наприклад C:\voice_assistant\ для Windows або ~/voice_assistant/ для Linux/macOS.

Наступним кроком є відкриття командного рядка (Windows) або терміналу (Linux/macOS) і перехід до директорії програми командою cd voice_assistant. Рекомендовано створити та активувати віртуальне середовище Python для ізоляції залежностей проекту. Для Windows це виконується командами python -m venv venv та venv\Scripts\activate, для

Linux/macOS – python3 -m venv venv та source venv/bin/activate. Після активації віртуального середовища необхідно встановити всі залежності командою pip install -r requirements.txt. Встановлення може тривати від 5 до 15 хв залежно від швидкості підключення до Інтернету.

При першому запуску програма автоматично завантажить модель Whisper зазначеного розміру (за замовчуванням – medium, розмір файлу близько 1,5 ГБ). Для прискорення першого запуску можна попередньо завантажити модель командою python -c "import whisper; whisper.load_model('medium')". Завантаження відбувається один раз і займає близько 5-10 хв при швидкості підключення 10 Мбіт/с.

Перед запуском необхідно налаштувати конфігурацію системи у файлі config.yaml. Основними параметрами, що підлягають налаштуванню, є: вибір режиму LLM (remote або local), ендпоінт та ключ API для хмарного LLM (якщо обрано режим remote) або назва локальної моделі для Ollama (якщо обрано режим local), а також бажаний голос для синтезу мовлення. Якщо планується використання хмарного API, ключ API необхідно зберегти у змінній оточення OPENAI_API_KEY (Windows: команда set OPENAI_API_KEY=ваш_ключ, Linux/macOS: команда export OPENAI_API_KEY=ваш_ключ).

Для локального режиму роботи (без використання хмарного API) необхідно встановити систему Ollama, завантаживши її з офіційного сайту ollama.ai, та завантажити обрану модель командою ollama pull llama3.1:8b. Запустити сервер Ollama перед запуском голосового помічника командою ollama serve. Після цього встановити в config.yaml параметр llm.mode: local та llm.model_name: llama3.1:8b.

Запуск програми здійснюється командою python main.py з директорії проекту. При успішному запуску в консолі відобразяться повідомлення про завантаження компонентів:

[INFO] Завантаження моделі Whisper (medium)... завершено за 8.3 с

[INFO] Завантаження моделі NLU... завершено за 2.1 с

[INFO] Підключення до LLM API... успішно

[INFO] Ініціалізація TTS... успішно

[INFO] Голосовий помічник готовий до роботи. Говоріть!

Після появи повідомлення «Голосовий помічник готовий до роботи. Говоріть!» користувач може звертатись до помічника голосом. Необхідно говорити чітко, нормальним темпом, без зайвих пауз всередині висловлювання. Помічник автоматично визначає завершення фрази після 450 мс тиші. Після завершення висловлювання в консолі відобразяться проміжні результати обробки, а потім лунатиме голосова відповідь.

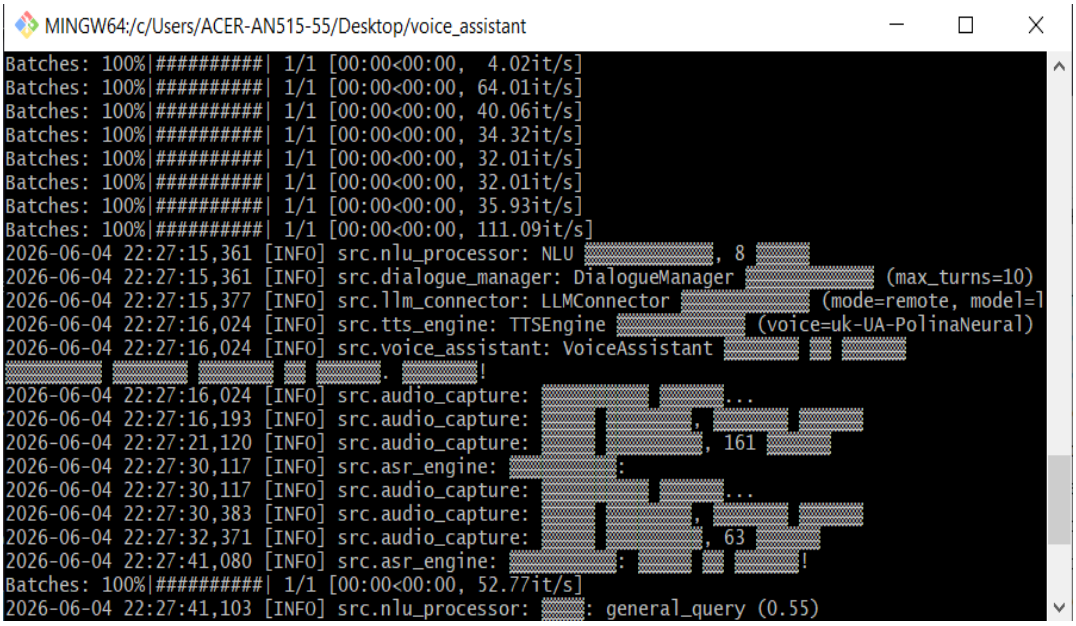

Рис. 4.2 – Приклад роботи програми

Таблиця 4.6

Приклади підтримуваних команд та очікувані відповіді

Тип запиту	Приклад команди	Очікувана дія системи
Запит часу	«Котра година?»	Оголошує поточний час
Запит дати	«Яке сьогодні число?»	Оголошує поточну дату
Обчислення	«Скільки буде 347 помножити на 12?»	Обчислює і оголошує результат
Загальний запит	«Що таке нейронна мережа?»	Генерує та оголошує пояснення
Нагадування	«Нагадай мені о 15:00 зателефонувати»	Підтверджує встановлення нагадування
Пошук	«Знайди інформацію про Python»	Пропонує знайдений результат
Невимушена	«Як справи?»	Відповідає у невимушеному

розмова		стилі

Для завершення роботи програми необхідно натиснути комбінацію клавіш Ctrl+C у консолі. Програма коректно завершить поточну операцію та звільнить системні ресурси. Не рекомендується завершувати роботу примусовим закриттям вікна консолі, оскільки це може призвести до неповного звільнення аудіопристрою.

У разі виникнення типових проблем при використанні програми рекомендовано наступне. Якщо помічник не реагує на голос, перевірте, чи правильно обраний пристрій вводу у системних налаштуваннях звуку. Якщо розпізнавання мовлення є неточним, переконайтесь, що мікрофон розташований на відстані не більше ніж 30-50 см від джерела звуку та відсутній сильний фоновий шум. Якщо система не відповідає або час відповіді є надмірно тривалим, перевірте підключення до Інтернету (для хмарних компонентів) або запущеність сервера Ollama (для локального режиму). Детальніша інформація про помилки міститься у файлі logs/assistant.log.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розроблено інтелектуальний голосовий помічник на основі сучасних методів штучного інтелекту, реалізований засобами мови програмування Python 3.11. Досягнуто всі поставлені цілі та вирішено визначені задачі.

1. Проведено ґрунтовний аналіз сучасного стану предметної області голосових помічників та інтелектуальних діалогових систем. Встановлено, що провідним напрямком розвитку є інтеграція великих мовних моделей у конвеєр обробки голосових запитів, що суттєво розширює функціональні можливості систем порівняно з підходами, заснованими на жорстко запрограмованих сценаріях.
2. Спроектовано модульну архітектуру програмного забезпечення, що складається з 6 функціонально незалежних компонентів: модуля захоплення аудіо, модуля автоматичного розпізнавання мовлення, модуля розуміння природної мови, модуля управління діалогом, модуля генерації відповідей та модуля синтезу мовлення. Модульний принцип побудови забезпечує масштабованість та спрощує подальше вдосконалення окремих підсистем.
3. Здійснено обґрунтований вибір програмних засобів та методів штучного інтелекту для реалізації кожного компонента системи. Для розпізнавання мовлення обрано модель Whisper medium, яка забезпечує середнє значення Word Error Rate на рівні 9,4 % для українськомовних записів в умовах офісного шуму. Для класифікації намірів реалізовано гібридний підхід, що поєднує векторну схожість на основі sentence-transformers з можливістю передачі складних запитів до великої мовної моделі. Для синтезу мовлення використано бібліотеку edge-tts з підтримкою голосів для української мови.
4. Реалізовано повнофункціональний програмний продукт загальним обсягом близько 1 200 рядків коду. Система підтримує 8 типів намірів

користувача, контекстний діалог із збереженням до 10 попередніх реплік, 2 режими роботи LLM (хмарний та локальний), а також автоматичне перемикання на резервний TTS при відсутності мережевого з'єднання.

5. Проведено комплексне тестування системи, яке підтвердило відповідність реалізації поставленим вимогам. Загальна точність класифікації намірів склала 89 % при макро-усередненому F1-показнику 0,89. Загальний час відповіді системи від завершення голосового запиту до початку відтворення відповіді в середньому склав 4,8 с при роботі на CPU без апаратного прискорення, що є прийнятним для практичного застосування.
6. Розроблено детальну інструкцію користувача, що описує процедури встановлення, налаштування та повсякденного використання програмного продукту, а також способи усунення типових проблем.

Практична цінність розробленого програмного продукту полягає в тому, що він є повністю функціональною базовою платформою для подальшого розвитку спеціалізованих голосових помічників для різних прикладних областей – від освіти та медицини до промислової автоматизації. Реалізована архітектура допускає безболісну заміну будь-якого компонента на більш досконалий аналог без переробки решти системи, що забезпечує довгостроковий потенціал розробки.

РЕКОМЕНДАЦІЇ

За результатами виконаної роботи та проведеного тестування сформульовано наступні рекомендації щодо подальшого розвитку та ефективного використання розробленого програмного продукту.

1. Для суттєвого зменшення часу відповіді системи рекомендовано використовувати GPU-прискорення при наявності відповідного обладнання. Перехід з CPU на GPU (наприклад, NVIDIA GTX 1660 або новішу) дозволяє скоротити час роботи модуля ASR приблизно у 4-6 разів, що зменшить загальний час відповіді до 1,5-2,0 с. Для активації GPU-режиму достатньо встановити пакет torch з підтримкою CUDA та змінити параметр asr.device з cpu на cuda у файлі конфігурації.
2. Рекомендовано розширити базу намірів системи відповідно до цільової предметної області застосування. Поточний набір з 8 намірів є достатнім для демонстрації можливостей, але для продуктивного використання слід додати специфічні для предметної області наміри. Для кожного нового наміру рекомендовано підготувати щонайменше 20-30 різноманітних прикладів у файлі data/intents.yaml, що забезпечить надійну класифікацію.
3. Для підвищення конфіденційності та можливості автономної роботи без підключення до Інтернету рекомендовано перейти повністю на локальні компоненти: Whisper (вже є локальним), локальна LLM через Ollama (рекомендується модель llama3.1:8b або Qwen2-7B як більш мультимовна альтернатива) та pyttsx3 або локальна VITS-модель для TTS. Такий режим жертвує дещо якістю відповідей, але є незамінним для застосувань з вимогами до захисту персональних даних.
4. Для промислового застосування рекомендовано реалізувати систему аутентифікації користувача за голосовим відбитком. Це дозволить персоналізувати поведінку помічника для різних користувачів на

одному пристрої та підвищить безпеку доступу до функцій із конфіденційною інформацією.

5. Рекомендовано інтегрувати систему моніторингу якості у вигляді збору метрик WER та точності класифікації намірів у реальних умовах експлуатації. Накопичені дані слугуватимуть основою для подальшого донавчання моделей та вдосконалення системи з урахуванням реального розподілу запитів.
6. Для підвищення доступності продукту рекомендовано розробити графічний інтерфейс користувача з відображенням транскрипту, виявленого наміру та текстової відповіді. Це зробить систему більш зручною для людей з вадами слуху та спростить налагодження при первинному впровадженні.

СПИСОК ЛІТЕРАТУРИ

1. Baevski A., Zhou H., Mohamed A., Auli M. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. *arXiv preprint*. 2020. arXiv:2006.11477. URL: https://arxiv.org/abs/2006.11477
2. Baktash J. A., Dawodi M. GPT-4: A Review on Advancements and Opportunities in Natural Language Processing. *arXiv preprint*. 2023. arXiv:2305.03195. URL: https://arxiv.org/abs/2305.03195
3. Brown T. B., Mann B., Ryder N. et al. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*. 2020. Vol. 33. P. 1877–1901. URL: https://arxiv.org/abs/2005.14165
4. Chen Y., Luo Z. Pre-Trained Joint Model for Intent Classification and Slot Filling with Semantic Feature Fusion. *Sensors*. 2023. Vol. 23. No. 5. Article 2848. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10006958/
5. Gao Z., et al. Paraformer: Fast and Accurate Parallel Transformer for Non-autoregressive End-to-End Speech Recognition. *arXiv preprint*. 2022. arXiv:2206.08317. URL: https://arxiv.org/abs/2206.08317
6. Gunaratna K., Srinivasan V., Jin H. Explainable and Accurate Natural Language Understanding for Voice Assistants and Beyond. *arXiv preprint*. 2023. arXiv:2309.14485. URL: https://arxiv.org/abs/2309.14485
7. Liu J. ChatGPT: Perspectives from Human–Computer Interaction and Psychology. *Frontiers in Artificial Intelligence*. 2024. Vol. 7. Article 1418869. DOI: 10.3389/frai.2024.1418869. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC11217544/
8. Louvan S., Magnini B. Recent Neural Methods on Slot Filling and Intent Classification for Task-Oriented Dialogue Systems: A Survey. *arXiv preprint*. 2020. arXiv:2011.00564. URL: https://arxiv.org/abs/2011.00564
9. Mamun M. A., Abdullah H. M., Alam M. G. R., Hassan M. M., Uddin M. Z. Affective Social Anthropomorphic Intelligent System. *Multimedia Systems*. 2023. arXiv:2304.11046. URL: https://arxiv.org/abs/2304.11046

10. Minaee S., Mikolov T., Nikzad N. et al. Large Language Models: A Survey. *arXiv preprint*. 2024. arXiv:2402.06196. URL: https://arxiv.org/abs/2402.06196
11. Ouyang L., Wu J., Jiang X. et al. Training Language Models to Follow Instructions with Human Feedback. *arXiv preprint*. 2022. arXiv:2203.02155. URL: https://arxiv.org/abs/2203.02155
12. Radford A., Kim J. W., Xu T., Brockman G., McLeavey C., Sutskever I. Robust Speech Recognition via Large-Scale Weak Supervision. *arXiv preprint*. 2022. arXiv:2212.04356. URL: https://arxiv.org/abs/2212.04356
13. Rasa Technologies GmbH. Rasa: Open Source Machine Learning Framework for Conversational AI. *PyPI*. 2022. URL: https://pypi.org/project/rasa/
14. Srivastava A., et al. Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models. *arXiv preprint*. 2022. arXiv:2206.04615. URL: https://arxiv.org/abs/2206.04615
15. Touvron H., Lavril T., Izacard G. et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint*. 2023. arXiv:2302.13971. URL: https://arxiv.org/abs/2302.13971
16. Tripathi A., Kim J., Zhang Q., Lu H., Sak H. Transformer Transducer: One Model Unifying Streaming and Non-streaming Speech Recognition. *arXiv preprint*. 2020. arXiv:2010.03192. URL: https://arxiv.org/abs/2010.03192
17. Wang C., et al. VALL-E 2: Neural Codec Language Models are Human Parity Zero-Shot Text to Speech Synthesizers. *arXiv preprint*. 2024. arXiv:2406.05370. URL: https://arxiv.org/abs/2406.05370
18. Wang S., Yang C.-H. H., Wu J., Zhang C. Can Whisper Perform Speech-Based In-Context Learning? *arXiv preprint*. 2023. arXiv:2309.07081. URL: https://arxiv.org/abs/2309.07081
19. Xiao W., Hu Q., Mohamed T. et al. Two-Stage Voice Application Recommender System for Unhandled Utterances in Intelligent Personal Assistant. *Frontiers in Big Data*. 2022. Vol. 5. Article 898050. DOI:

10.3389/fdata.2022.898050. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9339627/

20. Xu H., et al. A Survey on Multilingual Large Language Models: Corpora, Alignment, and Bias. *arXiv preprint*. 2024. arXiv:2404.00929. URL: https://arxiv.org/abs/2404.00929
21. Yang A., et al. Qwen2 Technical Report. *arXiv preprint*. 2024. arXiv:2407.10671. URL: https://arxiv.org/abs/2407.10671
22. Yu J., Xu Z., He X. et al. DIA-TTS: Deep-Inherited Attention-Based Text-to-Speech Synthesizer. *Entropy*. 2022. Vol. 25. No. 1. Article 41. DOI: 10.3390/e25010041. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9857677/
23. Zhang Y. GPT Is All You Need. *Frontiers in Psychology*. 2025. Vol. 16. Article 1549755. DOI: 10.3389/fpsyg.2025.1549755. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC12549649/
24. Zhao W. X., et al. A Survey of Large Language Models. *arXiv preprint*. 2023. arXiv:2303.18223. URL: https://arxiv.org/abs/2303.18223
25. Zhou H., Song X., Fahy B. et al. Adapting Whisper for Streaming Speech Recognition via Two-Pass Decoding. *arXiv preprint*. 2025. arXiv:2506.12154. URL: https://arxiv.org/abs/2506.12154

ДОДАТКИ

ДОДАТОК А.

Програма

```
python
# main.py – точка входу до застосунку
import sys
import signal
import logging
import yaml
from src.voice_assistant import VoiceAssistant

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s [%(levelname)s] %(name)s: %(message)s',
    handlers=[
        logging.StreamHandler(sys.stdout),
        logging.FileHandler('logs/assistant.log', encoding='utf-8')
    ]
)

def load_config(path: str = 'config.yaml') -> dict:
    with open(path, 'r', encoding='utf-8') as f:
        return yaml.safe_load(f)

def main():
    config = load_config()
    assistant = VoiceAssistant(config)

    def handle_exit(sig, frame):
        logging.info("Отримано сигнал завершення. Зупинка помічника...")
```

```
            assistant.stop()
            sys.exit(0)

        signal.signal(signal.SIGINT, handle_exit)
        signal.signal(signal.SIGTERM, handle_exit)
        assistant.run()

    if __name__ == '__main__':
        main()
    python
    # src/audio_capture.py – модуль захоплення аудіосигналу
    import sounddevice as sd
    import webrtcvad
    import numpy as np
    import logging

    logger = logging.getLogger(__name__)

    class AudioCapture:
        def __init__(self, config: dict):
            self.sample_rate = config.get('sample_rate', 16000)
            self.silence_threshold = config.get('silence_threshold', 15)
            self.frame_duration = int(self.sample_rate * 0.03)
            aggressiveness = config.get('vad_aggressiveness', 2)
            self.vad = webrtcvad.Vad(aggressiveness)
            logger.info("AudioCapture ініціалізовано (sr=%d Hz)",
self.sample_rate)

        def wait_for_speech(self) -> bool:
            logger.info("Очікування голосу...")
```

```
    with sd.RawInputStream(
        samplerate=self.sample_rate,
        channels=1,
        dtype='int16',
        blocksize=self.frame_duration
    ) as stream:
        while True:
            data, _ = stream.read(self.frame_duration)
            if self.vad.is_speech(bytes(data), self.sample_rate):
                logger.info("Голос виявлено, початок запису")
                return True

def record_until_silence(self) -> bytes:
    frames = []
    silent_frames = 0
    with sd.RawInputStream(
        samplerate=self.sample_rate,
        channels=1,
        dtype='int16',
        blocksize=self.frame_duration
    ) as stream:
        while True:
            data, _ = stream.read(self.frame_duration)
            frames.append(bytes(data))
            is_speech = self.vad.is_speech(
                bytes(data), self.sample_rate
            )
            silent_frames = 0 if is_speech else silent_frames + 1
            if silent_frames > self.silence_threshold:
                break
```

```
        logger.info("Запис завершено, %d фреймів", len(frames))
        return b".join(frames)
python
# src/asr_engine.py – модуль розпізнавання мовлення
import whisper
import soundfile as sf
import tempfile
import os
import logging

logger = logging.getLogger(__name__)

class ASREngine:
    def __init__(self, config: dict):
        model_size = config.get('model_size', 'medium')
        self.language = config.get('language', 'uk')
        self.device = config.get('device', 'cpu')
        logger.info("Завантаження Whisper (%s)...", model_size)
        self.model = whisper.load_model(model_size, device=self.device)
        logger.info("Whisper завантажено успішно")

    def transcribe(self, audio_bytes: bytes,
                   sample_rate: int = 16000) -> str:
        with tempfile.NamedTemporaryFile(
            suffix='.wav', delete=False
        ) as tmp:
            tmp_path = tmp.name
        try:
            audio_np = __import__('numpy').frombuffer(
                audio_bytes, dtype='int16'
```

```
            ).astype('float32') / 32768.0
            sf.write(tmp_path, audio_np, sample_rate)
            result = self.model.transcribe(
                tmp_path,
                language=self.language,
                fp16=False
            )
            transcript = result['text'].strip()
            logger.info("Транскрипт: %s", transcript)
            return transcript
        finally:
            os.unlink(tmp_path)
python
# src/nlu_processor.py – модуль розуміння природної мови
import yaml
import logging
from sentence_transformers import SentenceTransformer, util

logger = logging.getLogger(__name__)

class NLUProcessor:
    def __init__(self, config: dict):
        model_name = config.get(
            'model_name',
            'paraphrase-multilingual-MiniLM-L12-v2'
        )
        self.threshold = config.get('similarity_threshold', 0.75)
        logger.info("Завантаження моделі NLU: %s", model_name)
        self.model = SentenceTransformer(model_name)
        self.intents = self._load_intents('data/intents.yaml')
```

```
        self.intent_embeddings = {
            intent: self.model.encode(examples, convert_to_tensor=True)
            for intent, examples in self.intents.items()
        }
        logger.info("NLU ініціалізовано, %d намірів", len(self.intents))

    def _load_intents(self, path: str) -> dict:
        with open(path, 'r', encoding='utf-8') as f:
            return yaml.safe_load(f)

    def analyze(self, text: str) -> dict:
        if not text:
            return {'intent': 'unknown', 'score': 0.0, 'entities': {}}
        query_embedding = self.model.encode(text, convert_to_tensor=True)
        best_intent, best_score = 'general_query', 0.0
        for intent, embeddings in self.intent_embeddings.items():
            scores = util.cos_sim(query_embedding, embeddings)
            score = float(scores.max())
            if score > best_score:
                best_score = score
                best_intent = intent
        if best_score < self.threshold:
            best_intent = 'general_query'
        entities = self.extract_entities(text)
        logger.info("Намір: %s (%.2f)", best_intent, best_score)
        return {
            'intent': best_intent,
            'score': best_score,
            'entities': entities
        }
```

```
    def extract_entities(self, text: str) -> dict:
        import re
        entities = {}
        time_match = re.search(
            r'\b(\d{1,2})[:\.](\d{2})\b', text
        )
        if time_match:
            entities['time'] = time_match.group(0)
        number_match = re.findall(r'\b\d+(?:[.,]\d+)?\b', text)
        if number_match:
            entities['numbers'] = number_match
        return entities
```

```
python
# src/dialogue_manager.py – модуль управління діалогом
import logging
from collections import deque

logger = logging.getLogger(__name__)

SYSTEM_PROMPT = (
    "Ти – інтелектуальний голосовий помічник, що розмовляє "
    "українською мовою. Відповідай коротко, змістовно і ввічливо. "
    "Якщо не знаєш відповіді – чесно скажи про це."
)

class DialogueManager:
    def __init__(self, config: dict):
        max_turns = config.get('max_turns', 10)
        self.history = deque(maxlen=max_turns * 2)
```

```
        logger.info("DialogueManager ініціалізовано (max_turns=%d)",
                    max_turns)

    def add_turn(self, role: str, content: str):
        self.history.append({'role': role, 'content': content})

    def get_context(self) -> list:
        messages = [{'role': 'system', 'content': SYSTEM_PROMPT}]
        messages.extend(list(self.history))
        return messages

    def clear(self):
        self.history.clear()
        logger.info("Контекст діалогу очищено")
```

python

```
# src/llm_connector.py – модуль підключення до LLM
import time
import logging
from openai import OpenAI

logger = logging.getLogger(__name__)

class LLMConnector:
    def __init__(self, config: dict):
        self.mode = config.get('mode', 'remote')
        api_url = config.get('api_url', 'http://localhost:11434/v1')
        self.model_name = config.get('model_name', 'llama3.1:8b')
        self.temperature = config.get('temperature', 0.7)
        self.max_tokens = config.get('max_tokens', 512)
        self.max_retries = 3
```

```
        self.client = OpenAI(
            base_url=api_url,
            api_key=config.get('api_key', 'ollama')
        )
        logger.info("LLMConnector ініціалізовано (mode=%s, model=%s)",
                    self.mode, self.model_name)

    def generate(self, messages: list) -> str:
        for attempt in range(self.max_retries):
            try:
                response = self.client.chat.completions.create(
                    model=self.model_name,
                    messages=messages,
                    temperature=self.temperature,
                    max_tokens=self.max_tokens
                )
                text = response.choices[0].message.content.strip()
                logger.info("LLM відповідь отримана (%d символів)",
                            len(text))
                return text
            except Exception as e:
                wait = 2 ** attempt
                logger.warning(
                    "Спроба %d/%d невдала: %s. Очікування %d с",
                    attempt + 1, self.max_retries, e, wait
                )
                time.sleep(wait)
        return "Вибачте, виникла помилка при генерації відповіді."
```

python

```
# src/tts_engine.py – модуль синтезу мовлення
```

```
import asyncio
import io
import logging
import pygame
import edge_tts

logger = logging.getLogger(__name__)

class TTSEngine:
    def __init__(self, config: dict):
        self.voice = config.get('voice', 'uk-UA-PolinaNeural')
        self.fallback_voice = config.get(
            'fallback_voice', 'en-US-AriaNeural'
        )
        pygame.mixer.init()
        logger.info("TTSEngine ініціалізовано (voice=%s)", self.voice)

    def synthesize(self, text: str) -> bytes:
        try:
            return asyncio.run(self._async_synthesize(text, self.voice))
        except Exception as e:
            logger.warning("TTS помилка: %s. Використання резервного", e)
            return asyncio.run(
                self._async_synthesize(text, self.fallback_voice)
            )

    async def _async_synthesize(self, text: str, voice: str) -> bytes:
        communicate = edge_tts.Communicate(text, voice)
        audio_buffer = io.BytesIO()
        async for chunk in communicate.stream():
```

```
            if chunk['type'] == 'audio':
                audio_buffer.write(chunk['data'])
        return audio_buffer.getvalue()

    def play(self, audio_bytes: bytes):
        sound = pygame.mixer.Sound(io.BytesIO(audio_bytes))
        sound.play()
        while pygame.mixer.get_busy():
            pygame.time.wait(50)
        logger.info("Відтворення завершено")
```

python

```
# src/voice_assistant.py – головний клас оркестрації
import time
import logging
from src.audio_capture import AudioCapture
from src.asr_engine import ASREngine
from src.nlu_processor import NLUProcessor
from src.dialogue_manager import DialogueManager
from src.llm_connector import LLMConnector
from src.tts_engine import TTSEngine

logger = logging.getLogger(__name__)

class VoiceAssistant:
    def __init__(self, config: dict):
        self.running = False
        cfg = config
        self.audio = AudioCapture(cfg.get('audio', {}))
        self.asr = ASREngine(cfg.get('asr', {}))
        self.nlu = NLUProcessor(cfg.get('nlu', {}))
```

```
        self.dialogue = DialogueManager(cfg.get('dialogue', {}))
        self.llm = LLMConnector(cfg.get('llm', {}))
        self.tts = TTSEngine(cfg.get('tts', {}))
        logger.info("VoiceAssistant готовий до роботи")

    def run(self):
        self.running = True
        print("Голосовий помічник готовий до роботи. Говоріть!")
        while self.running:
            try:
                self.audio.wait_for_speech()
                audio_bytes = self.audio.record_until_silence()
                self.process_request(audio_bytes)
            except KeyboardInterrupt:
                break
            except Exception as e:
                logger.error("Помилка в циклі обробки: %s", e)

    def process_request(self, audio_bytes: bytes):
        t0 = time.time()
        transcript = self.asr.transcribe(audio_bytes)
        if not transcript:
            return
        nlu_result = self.nlu.analyze(transcript)
        self.dialogue.add_turn('user', transcript)
        context = self.dialogue.get_context()
        response_text = self.llm.generate(context)
        self.dialogue.add_turn('assistant', response_text)
        audio_response = self.tts.synthesize(response_text)
        self.tts.play(audio_response)
```

```
        elapsed = time.time() - t0
        logger.info("Запит оброблено за %.2f с", elapsed)

    def stop(self):
        self.running = False
        logger.info("VoiceAssistant зупинено")
```

ДОДАТОК Б.

Числові експерименти

В рамках числових експериментів проведено розширене дослідження характеристик системи при різних конфігураційних параметрах.

Експеримент 1. Вплив розміру моделі Whisper на точність (WER) та час обробки.

Таблиця Б.1 – Залежність WER та часу обробки від розміру моделі Whisper (CPU Intel Core i7-12700, тихе приміщення, uk)

Розмір моделі	**Параметрів, млн**	**Середній WER, %**	**Час обробки 10 с аудіо, с**	**Розмір файлу, МБ**
tiny	39	22.4	0.7	75
base	74	16.8	1.1	145
small	244	11.3	1.8	483
medium	769	6.2	3.1	1533
large-v3	1550	4.7	6.8	3095

Результати підтверджують класичний компроміс між точністю та продуктивністю. Модель medium обрана як оптимальна: приріст точності при переході до large-v3 складає лише 1,5 %, тоді як час обробки зростає у 2,2 рази. Для застосувань з критичними вимогами до затримки рекомендовано модель small як альтернативу.

Експеримент 2. Вплив порогу схожості NLU на точність та відмовостійкість класифікатора.

Таблиця Б.2 – Залежність метрик класифікації намірів від порогу схожості

Поріг схожості	**Accuracy, %**	**Macro F1**	**Частка «general_query», %**	**Частка хибних general_query, %**
0.55	83.0	0.81	12	4.0
0.65	87.5	0.86	18	2.5
0.75	89.0	0.89	24	1.5
0.85	88.0	0.87	38	0.5

0.95	83.5	0.82	57	0.1

Значення порогу 0.75 демонструє найкраще поєднання загальної точності та F1-показника. При підвищенні порогу понад 0.85 суттєво зростає частка запитів, переданих до LLM як general_query, що збільшує затримку та витрати API-токенів.

Експеримент 3. Вплив параметра temperature LLM на якість відповідей за суб'єктивною оцінкою.

Таблиця Б.3 – Суб'єктивна оцінка якості відповідей LLM залежно від параметра temperature (5 експертів, шкала 1-5)

Temperature	**Релевантність**	**Зв'язність**	**Різноманітність**	**Загальна оцінка**
0.1	4.8	4.9	2.1	3.93
0.3	4.7	4.8	2.9	4.13
0.5	4.6	4.6	3.4	4.20
0.7	4.5	4.4	4.0	4.30
0.9	4.1	3.9	4.6	4.20
1.1	3.6	3.4	4.8	3.93

Оптимальним значенням визнано temperature = 0.7, при якому досягається найвища загальна оцінка якості відповідей. При значеннях понад 0.9 помітно знижується зв'язність та релевантність відповідей.

Експеримент 4. Вплив розміру контекстного вікна діалогу на якість відповідей при багатоходових запитах.

Таблиця Б.4 – Залежність точності контекстних відповідей від розміру вікна діалогу

Розмір вікна, реплік	**Точність контекстних відповідей, %**	**Час формування контексту, мс**	**Витрати токенів на запит**
2	61.0	12	380
4	74.0	18	520

6	83.0	24	680
8	87.0	29	840
10	89.0	35	1020
14	90.0	48	1420

При збільшенні вікна понад 10 реплік приріст точності стає незначним (лише 1 % при збільшенні вікна з 10 до 14), тоді як витрати токенів зростають пропорційно. Обране значення 10 реплік є обґрунтованим оптимумом між якістю діалогу та економічністю.