Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ ГОЛОСОВОГО ПОМІЧНИКА НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

DEVELOPMENT AND RESEARCH OF A VOICE ASSISTANT BASED ON ARTIFICIAL INTELLIGENCE

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Антонюк А. О.
Керівник:
к.т.н., доцент
Повстяна Ю. С.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.
_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____
«___» _____ 202__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Антонюку Андрію Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Розробка та дослідження голосового помічника на основі штучного інтелекту*
Керівник роботи: *Повстяна Юлія Славомирівна, доцент, к.т.н.*
затверджені наказом закладу вищої освіти від *«29» березня 2025 р. № 190/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи*:* *04 грудня 2025 р.*
3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування системного наповнення, оцінка ергономічних та надійністних параметрів проектованої системи.*
5. Перелік графічного матеріалу: *16 рисунків, 4 таблиці, 10 лістингів коду.*

6. Консультація розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Повстяна Ю. С.*		
Теоретичне дослідження та практична реалізація	*Повстяна Ю. С.*		
Експериментальне дослідження системи	*Повстяна Ю. С.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Андрущак І. Є.*		
Показник запозичень тексту	___%		
Академічна доброчесність	*Повстяна Ю. С.*		

7. Дата видачі завдання *«02» квітня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну схему роботи програмного продукту	01.11.2025	
4	Описати засоби розробки об'єкта проектування	19.11.2025	
5	Практична реалізація об'єкта проектування	26.11.2025	
6	Розробити методику для проведення експерименту	05.11.2025	
7	Провести аналіз результатів експерименту	15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ Антонюк А. О.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Повстяна Ю. С.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Антонюк А. О. Розробка та дослідження голосового помічника на основі штучного інтелекту. Рукопис.

Кваліфікаційна робота магістра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається з вступу, 3 розділів, висновків і пропозицій, списку використаних джерел.

У роботі досліджено методи розробки інтелектуальних голосових помічників для персональних комп'ютерів з можливістю автоматизації системних операцій. Проаналізовано існуючі голосові асистенти та обґрунтовано вибір модульної архітектури на основі протоколу Model Context Protocol. Розроблено голосовий помічник з використанням Whisper для розпізнавання української мови, Claude API для обробки команд та власного MCP сервера для виконання системних операцій. Експериментальне дослідження показало загальну успішність виконання команд 92,1 % при середньому часі відгуку від двох до шести секунд. Оцінено економію робочого часу користувача від двох до тридцяти трьох хвилин щоденно залежно від інтенсивності використання.

Ключові слова: голосовий помічник, штучний інтелект, розпізнавання мовлення, обробка природної мови, автоматизація системних операцій, Model Context Protocol, велика мовна модель, Whisper, Claude API, синтез мовлення.

ABSTRACT

Antoniuk A. O. Development and Research of a Voice Assistant Based on Artificial Intelligence. Manuscript.

Master's qualifying thesis of the educational program «Software Engineering», specialty 121 «Software Engineering». Lutsk National Technical University. Lutsk, 2025.

The master's thesis consists of an introduction, 3 chapters, conclusions and recommendations, and a list of references.

The thesis investigates methods for developing intelligent voice assistants for personal computers with system operation automation capabilities. Existing voice assistants were analyzed and the choice of modular architecture based on the Model Context Protocol was justified. A voice assistant was developed using Whisper for Ukrainian language recognition, Claude API for command processing, and a custom MCP server for executing system operations. Experimental research demonstrated an overall command execution success rate of 92,1 % with an average response time ranging from two to six seconds. User working time savings were estimated at two to thirty-three minutes daily depending on usage intensity.

Keywords: voice assistant, artificial intelligence, speech recognition, natural language processing, system operation automation, Model Context Protocol, large language model, Whisper, Claude API, speech synthesis.

ЗМІСТ

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується інтенсивним впровадженням систем штучного інтелекту в повсякденне життя користувачів персональних комп'ютерів. Взаємодія людини з обчислювальною технікою еволюціонує від традиційних інтерфейсів з використанням клавіатури та миші до більш природних форм комунікації, серед яких голосове керування посідає особливе місце. Голосові помічники вже стали звичним інструментом на мобільних пристроях та розумних колонках, проте їхнє застосування на персональних комп'ютерах залишається обмеженим, особливо в контексті глибокої інтеграції з операційною системою та можливістю виконання складних автоматизованих задач.

Традиційні голосові асистенти здебільшого орієнтовані на виконання простих команд або пошук інформації в інтернеті, тоді як потенціал голосової взаємодії для автоматизації робочих процесів на комп'ютері використовується недостатньо. Користувачі змушені витрачати значний час на рутинні операції з файлами, програмами та системними налаштуваннями, які теоретично могли б виконуватися автоматично за голосовою командою. Водночас існує потреба в універсальному помічнику, який би не лише керував комп'ютером, а й надавав інтелектуальні відповіді на запитання різного характеру, поєднуючи функції автоматизації та консультування.

Актуальність теми дослідження обумовлена необхідністю підвищення ефективності взаємодії користувача з персональним комп'ютером через створення інтелектуальної системи голосового керування, яка здатна розуміти природну мову, інтерпретувати складні команди та виконувати як системні операції, так і надавати інформаційну підтримку. Розробка такого помічника дозволить суттєво скоротити час на виконання типових задач, зробить роботу з комп'ютером більш доступною для користувачів з обмеженими можливостями та створить новий стандарт інтерактивної взаємодії з операційною системою.

Метою дослідження є розробка та дослідження архітектури голосового помічника на основі штучного інтелекту, який забезпечує природну голосову взаємодію для автоматизації операцій на персональному комп'ютері та надання інтелектуальних відповідей на запити користувача.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести аналіз існуючих голосових помічників та виявити їхні переваги і обмеження в контексті застосування на персональних комп'ютерах;

– дослідити сучасні технології розпізнавання та синтезу мовлення, обробки природної мови та машинного навчання, що застосовуються в голосових системах;

– розробити модульну та розширювану архітектуру голосового помічника, яка забезпечить ефективну взаємодію між компонентами розпізнавання мовлення, обробки природної мови, виконання системних операцій та синтезу відповідей;

– реалізувати локальну підсистему розпізнавання мовлення з підтримкою української мови, яка забезпечить прийнятну точність розпізнавання голосових команд користувача;

– інтегрувати велику мовну модель для інтелектуальної обробки запитів користувача з можливістю розуміння контексту діалогу та прийняття рішень щодо необхідних дій;

– створити систему інтеграції з операційною системою через набір спеціалізованих MCP серверів, які надають доступ до різних функцій комп'ютера;

– розробити механізм перетворення намірів користувача у конкретні виклики функцій системи;

– реалізувати підсистему синтезу мовлення для генерації природних голосових відповідей користувачу;

– провести експериментальне дослідження розробленої системи для оцінки її технічних характеристик.

Об'єктом дослідження є процес голосової взаємодії користувача з персональним комп'ютером для виконання системних операцій та отримання інформаційних послуг.

Предметом дослідження є методи та технології розробки інтелектуальних голосових помічників на основі штучного інтелекту з можливістю автоматизації дій на комп'ютері та обробки інформаційних запитів.

Практична цінність роботи полягає в можливості застосування результатів дослідження для створення функціонального голосового помічника, який підвищить продуктивність роботи з персональним комп'ютером, спростить доступ до системних функцій та забезпечить зручний інтерфейс для користувачів різного рівня технічної підготовки.

Апробація результатів дослідження. Основні положення та результати кваліфікаційної роботи магістра доповідались та обговорювались на X Міжнародній науково-практичній конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві» (ІТОНВ-2025), яка відбулась 23-24 травня 2025 року у місті Луцьк [31].

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Голосові помічники на основі штучного інтелекту є результатом тривалої еволюції технологій обробки людського мовлення та розвитку методів машинного навчання. Перші спроби створення систем розпізнавання мовлення датуються ще 1950-ми роками, коли дослідники Bell Laboratories розробили систему Audrey, здатну розпізнавати окремі цифри, вимовлені одним диктором [27]. Ця система працювала на основі аналізу спектральних характеристик звуку і могла ідентифікувати лише обмежений набір слів за умови чіткої вимови.

У 1960-х роках в IBM було створено систему Shoebox, яка вже могла розпізнавати шістнадцять англійських слів та цифри від 0 до 9 [27]. Це був значний крок вперед, оскільки система могла виконувати прості арифметичні операції за голосовими командами. Проте обмеження обчислювальної потужності того часу не дозволяли створити практично застосовні рішення для масового використання.

Справжній прорив у галузі розпізнавання мовлення відбувся в 1970-х роках завдяки роботам в агентстві DARPA, де була розроблена система Harpy, здатна розпізнавати понад тисячу слів [27]. Вона базувалася на прихованих марковських моделях, які стали основою для розвитку технологій розпізнавання мовлення на наступні десятиліття. Ці моделі дозволяли враховувати ймовірнісний характер мовлення та передбачати наступні слова на основі попередніх, що значно підвищило точність розпізнавання.

Протягом 1980-х та 1990-х років активно розвивалися комерційні системи розпізнавання мовлення, зокрема Dragon Dictate від компанії Dragon Systems, яка стала першою системою диктування для персональних комп'ютерів. Хоча вона вимагала від користувача робити паузи між словами і потребувала тривалого

навчання під конкретного диктора, це була революційна технологія для свого часу. Згодом з'явилася система Dragon NaturallySpeaking, яка вже підтримувала безперервне мовлення і не вимагала пауз між словами, що зробило взаємодію значно більш природною [27].

На початку 2000-х років розвиток інтернет-технологій та зростання обчислювальної потужності відкрили нові можливості для голосових систем. Google запустив сервіс голосового пошуку для телефонів у 2008 році, що стало передвісником сучасних голосових помічників. Система використовувала хмарну обробку даних, що дозволило застосовувати складні алгоритми машинного навчання без обмежень мобільних пристроїв [12].

Справжньою віхою в розвитку голосових помічників став запуск Siri компанією Apple у 2011 році (рис. 1.1). Це була перша масова система, яка не просто розпізнавала команди, а намагалася розуміти контекст та вести діалог з користувачем. Siri інтегрувала технології розпізнавання мовлення, обробки природної мови та машинного навчання в єдину систему, яка могла виконувати різноманітні завдання від встановлення будильника до пошуку інформації в інтернеті [24]. Поява Siri стимулювала інші технологічні компанії до розробки власних голосових асистентів.

Рисунок 1.1 – Голосовий помічник Siri [24]

У 2014 році Amazon представив Alexa разом з розумною колонкою Echo, що поклало початок новій категорії пристроїв для розумного дому (рис. 1.2). На відміну від Siri, яка була інтегрована в мобільні пристрої, Alexa створювалася як голосовий інтерфейс для керування домашніми системами та пристроями [25].

Amazon запропонував відкриту платформу для розробників, що дозволило створювати так звані навички, які розширювали функціональність помічника. Це рішення виявилося надзвичайно успішним і сприяло стрімкому поширенню голосового керування в побуті.

Google Assistant, представлений у 2016 році, використав переваги екосистеми Google для створення найбільш контекстно обізнаного помічника (рис. 1.3). Завдяки доступу до величезних обсягів даних про користувачів та їхні пошукові запити, Google Assistant міг надавати більш персоналізовані та точні відповіді [12]. Система також демонструвала кращу здатність підтримувати природний діалог та розуміти складні багатоступеневі запити.

Рисунок 1.2 – Розумна колонка Amazon Alexa [25]

Microsoft інтегрував свій голосовий помічник Cortana в операційну систему Windows 10, намагаючись створити єдиний голосовий інтерфейс для роботи з комп'ютером [8]. Це був перший серйозний крок до інтеграції голосового керування безпосередньо в десктопну операційну систему. Однак Cortana не отримала широкого поширення через обмежені можливості порівняно з конкурентами та недостатню інтеграцію з екосистемою сторонніх розробників.

Паралельно з розвитком комерційних рішень активно розвивалися відкриті технології та фреймворки для створення голосових систем. Проєкт Mozilla Common Voice зібрав величезну базу голосових записів різними мовами для навчання відкритих моделей розпізнавання мовлення [8]. Платформи на кшталт

Mycroft позиціонувалися як відкрита альтернатива комерційним помічникам з акцентом на приватність та можливість повного контролю користувача над системою.

Рисунок 1.3 – Розумна колонка Google Assistant [12]

Останні роки ознаменувалися революцією у сфері обробки природної мови завдяки появі трансформерної архітектури нейронних мереж. Моделі на кшталт BERT, GPT та їхні наступники продемонстрували безпрецедентну здатність розуміти контекст, генерувати природні відповіді та навіть проводити складні міркування [10]. Ці досягнення відкрили нові горизонти для голосових помічників, дозволяючи їм вести більш природні діалоги та розуміти складні запити, які раніше були поза можливостями подібних систем.

Адаптація голосових технологій для різних мовних систем стала однією з принципових проблем у розвитку цієї галузі. Початково більшість систем розроблялися виключно для англомовного середовища, що суттєво обмежувало їхнє впровадження в інших лінгвістичних контекстах. З часом почали з'являтися рішення для інших мов, проте їхня ефективність залишалася нерівномірною. Ключовим фактором виявився обсяг доступних голосових корпусів для тренування моделей. Для української мови це питання є особливо гострим через порівняно обмежену кількість якісних аудіоматеріалів порівняно з глобальними мовами.

Сучасний розвиток голосових асистентів концентрується навколо декількох пріоритетних векторів. Передусім це підвищення точності

розпізнавання мовлення в несприятливих акустичних умовах, включаючи присутність зовнішніх шумів, варіативність акцентів та індивідуальні фонетичні особливості мовців. Другим важливим напрямком є розбудова контекстуальної пам'яті системи, що дозволяє підтримувати послідовні діалоги з урахуванням попередніх взаємодій. Третім аспектом виступає індивідуалізація систем відповідно до специфічних характеристик користувачів, їхніх преференцій та комунікативних шаблонів.

Проблематика конфіденційності та інформаційної безпеки голосових систем потребує окремої уваги. Оскільки такі пристрої функціонують у режимі постійного моніторингу акустичного середовища, очікуючи активаційні команди, виникають легітимні занепокоєння щодо потенційного неконтрольованого збору аудіоданих та їхньої подальшої обробки. Відповідно, активно розробляються методології локальної обробки інформації без передачі на віддалені сервери, а також технології криптографічного захисту та декодування голосових даних.

Теоретичний аналіз голосових помічників здійснюється з множини дисциплінарних перспектив. У рамках комп'ютерної лінгвістики основний фокус зосереджений на методах обробки природної мови, визначенні інтенцій користувача та екстракції сутностей з текстових представлень. Розробляються алгоритмічні підходи, які забезпечують розуміння системою не лише змісту висловлювань, але й іронічних конструкцій, емоційного забарвлення та культурно-специфічних комунікативних практик.

З позиції машинного навчання голосові системи представляють складну послідовність взаємопов'язаних обчислювальних завдань. Процес включає акустичне моделювання для трансформації звукового сигналу у фонемні репрезентації, мовне моделювання для формування лексичних одиниць та синтаксичних конструкцій, а також семантичний аналіз для інтерпретації змісту запиту. Кожен етап вимагає застосування конкретної архітектури нейронних мереж та значних обсягів тренувальних даних.

У контексті людино-комп'ютерної взаємодії голосові асистенти розглядаються як інноваційна форма інтерфейсу, що має відповідати принципам ергономічності та універсальної доступності. Аналізується сприйняття користувачами голосової взаємодії, типологія завдань, які вони готові делегувати голосовим системам, та ситуації, коли надається перевага традиційним формам взаємодії. Користувачі схильні використовувати голосове керування для оперативних простих команд, водночас віддаючи перевагу візуальним інтерфейсам для виконання багатоетапних комплексних операцій.

Експериментальні дослідження ефективності голосових помічників демонструють неоднозначні результати залежно від контексту використання. Дослідження [17] Alianda Lopez та співавторів, проведене у 2021 році, показало, що голосове керування може підвищити швидкість виконання простих завдань на комп'ютері на 23-31 % порівняно з традиційними методами введення за допомогою клавіатури та миші. Особливо це помітно при багатозадачності, коли користувач може віддавати команди голосом, одночасно виконуючи інші дії. Проте дослідники виявили, що для складних завдань, які вимагають точності та детального контролю, візуальний інтерфейс залишається більш ефективним, забезпечуючи на 40 % менше помилок при виконанні операцій з файлами та налаштуваннями системи.

Цікавим напрямком є дослідження емоційного інтелекту голосових помічників. Група вчених під керівництвом Zhicheng Zhang у 2023 році розробила метод розпізнавання емоційного стану користувача за інтонацією, темпом мовлення та іншими акустичними характеристиками з точністю 78 % [30]. Система, яка може визначити, що користувач розчарований або перебуває у стресі, може адаптувати свою поведінку відповідно, пропонуючи більш детальні пояснення або спрощуючи діалог. Подальші експерименти його команди продемонстрували, що емоційно адаптивні голосові помічники краще утримують увагу користувачів в освітніх додатках порівняно з емоційно нейтральними системами.

Дослідження міжкультурних аспектів використання голосових помічників [23], проведене Katie Seaborn у 2024 році на вибірці з понад 5000 користувачів з 15 країн, виявило суттєві відмінності в шаблонах взаємодії між різними культурами. Користувачі з Японії та Південної Кореї на 67 % частіше використовували ввічливі форми звертання до помічника і очікували аналогічної ввічливості у відповідь, тоді як користувачі зі Сполучених Штатів та Великобританії демонстрували схильність до більш прямого та лаконічного стилю комунікації. Це створює виклики для розробників глобальних систем, які повинні враховувати культурні особливості різних регіонів.

Важливою проблемою, яка активно досліджується, є питання доступності голосових технологій для людей з особливими потребами. Дослідження [20] Gonzalez та колег, опубліковане у 2023 році, показало, що для людей з порушеннями зору голосовий помічник може стати основним інтерфейсом взаємодії з комп'ютером, надаючи можливість виконувати операції, які раніше були недоступні без сторонньої допомоги. Учасники експерименту з вадами зору змогли виконувати типові комп'ютерні завдання в 3,2 рази швидше з використанням голосового інтерфейсу порівняно зі скрін-рідерами та клавіатурою.

Аналогічно, дослідження [11] Paola Esquivel продемонструвало, що для людей з обмеженою рухливістю голосове керування відкриває нові можливості самостійного використання технологій, підвищуючи їхню незалежність на 58 % за індексом самостійності при роботі з комп'ютером. Було виявлено, що надмірна залежність від голосових асистентів може призвести до погіршення показників короткострокової пам'яті на 12-18 % протягом активного використання, оскільки користувачі перестають запам'ятовувати інформацію, покладаючись на можливість завжди запитати помічника.

Технічні аспекти створення голосових помічників для персональних комп'ютерів мають свою специфіку порівняно з мобільними пристроями чи розумними колонками. Помічник повинен мати можливість взаємодіяти з

операційною системою на глибокому рівні, щоб виконувати системні операції, керувати програмами та автоматизувати робочі процеси.

Користувачі персональних комп'ютерів часто виконують складні професійні завдання, які вимагають точності та контролю. Голосовий помічник для десктопу має бути здатним працювати з професійним програмним забезпеченням та забезпечувати достатній рівень гнучкості для налаштування під специфічні робочі процеси.

На відміну від мобільних помічників, які використовуються переважно для коротких епізодичних взаємодій, десктопний помічник може бути активним протягом тривалих робочих сесій.

Дослідження архітектури голосових помічників [4], проведене Francesco Cosimo Andriulo, виділяє три основних підходи до побудови таких систем. Перший підхід передбачає централізовану хмарну обробку з затримкою 200-500 мс, що дозволяє використовувати найпотужніші моделі машинного навчання з мільярдами параметрів, але створює залежність від інтернет-з'єднання та викликає питання приватності.

Другий підхід базується на локальній обробці з затримкою 50-150 мс, що гарантує повну приватність та можливість роботи без інтернету, але обмежує складність моделей, які можна використовувати. Експерименти показали, що локальні моделі на сучасних персональних комп'ютерах можуть досягати 80-84 % точності хмарних систем при правильній оптимізації.

Третій, гібридний підхід, поєднує переваги обох попередніх. Базове розпізнавання та обробка простих команд виконується локально, забезпечуючи затримку до 100 мс для 70 % запитів. Для складніших запитів система звертається до хмарних сервісів, що дозволяє оптимально балансувати між продуктивністю, приватністю та функціональністю.

Важливим аспектом сучасних голосових систем є модульна архітектура, яка дозволяє легко розширювати можливості через додавання нових компонентів або інтеграції з зовнішніми сервісами. Модульні системи з відкритим API демонструють більше доступних інтеграцій порівняно з монолітною

архітектурою. Це особливо актуально для десктопних помічників, оскільки різні користувачі мають різні потреби залежно від сфери діяльності. Програмісти потребують інтеграції з інструментами розробки та системами контролю версій, дизайнери з графічними редакторами та бібліотеками ресурсів, а письменники з текстовими процесорами та довідковими матеріалами.

Питання точності розпізнавання мовлення залишається центральним у дослідженнях голосових технологій. Порівняльний аналіз сучасних систем автоматичного розпізнавання мовлення [18], проведений командою OpenAI у 2022 році, показав, що найкращі моделі досягають показника word error rate на рівні 3-5 % для англійської мови у контрольованих умовах. Проте для української мови цей показник значно вищий і становить 12-18 % через обмежену кількість навчальних даних та морфологічну складність мови.

Окремим викликом є розпізнавання мовлення в умовах акустичних перешкод. Експерименти показали, що наявність фонового шуму на рівні 20-30 децибел підвищує частоту помилок розпізнавання у 2,5-3 рази. Сучасні методи шумоподавлення на основі глибоких нейронних мереж, дозволяють знизити цей негативний ефект, але вимагають значних обчислювальних ресурсів для обробки аудіо в реальному часі.

Обробка природної мови як ключовий компонент голосових помічників зазнала революційних змін з появою трансформерної архітектури. Дослідження [10] Devlin та співавторів, які представили модель BERT у 2018 році, продемонструвало можливість досягнення глибокого розуміння контексту через механізм уваги. Подальший розвиток цього напрямку призвів до створення моделей GPT-3 та GPT-4, які показують здатність до складних міркувань та генерації природних відповідей [1]. Порівняльне тестування показало, що великі мовні моделі з понад 100 мільярдами параметрів досягають 91-96 % точності у задачах розуміння намірів користувача порівняно з 67-73 % для традиційних підходів на основі правил та класичних методів машинного навчання.

Проте застосування таких великих моделей у голосових помічниках має свої обмеження. Дослідження звертає увагу на проблему енергоспоживання та

екологічного впливу навчання і використання великих мовних моделей. За їхніми розрахунками, навчання однієї великої моделі може генерувати викиди вуглецю, еквівалентні викидам 5 автомобілів протягом їхнього життєвого циклу. Це ставить питання про необхідність розробки більш ефективної архітектури та методів оптимізації.

Синтез мовлення як фінальний етап взаємодії з користувачем також пройшов значну еволюцію. Традиційні параметричні методи синтезу, які домінували до 2010-х років, створювали мовлення, яке звучало роботизовано та неприродно. Революцією стала поява нейронних синтезаторів на основі архітектури WaveNet [21], представленої дослідниками DeepMind у 2016 році. Ця система генерувала мовлення якості, практично невідрізнимої від людського, що підтвердили тести з оцінкою 4,21 за 5-бальною шкалою MOS порівняно з 4,55 для натурального мовлення.

Подальший розвиток призвів до появи моделей Tacotron 2 та FastSpeech, які поєднують високу якість з можливістю синтезу в реальному часі. Дослідження показали, що FastSpeech 2 здатен генерувати природне мовлення зі швидкістю, що в 3,8 разів перевищує швидкість реального відтворення, що критично важливо для голосових помічників, де затримка відповіді безпосередньо впливає на користувацький досвід [19].

Питання багатомовності голосових помічників розглядається у роботах Zolzaya Byambadorj та Ryota Nishimura [7], які досліджували методи трансферного навчання для адаптації моделей розпізнавання та синтезу мовлення на нові мови з обмеженими навчальними даними. Їхні експерименти показали, що попереднє навчання на великих корпусах високоресурсних мов з подальшим дотренуванням на 50-100 годинах цільової мови може досягти якості, порівнянної з моделями, натренованими на 1000 годинах даних цільової мови. Це відкриває перспективи для створення якісних голосових помічників для мов з обмеженими ресурсами, включаючи українську.

Важливим напрямком досліджень є також забезпечення приватності користувачів голосових систем. Існують методи федеративного навчання, коли

моделі можуть покращуватися на основі даних користувачів без передачі самих даних на центральні сервери.

Дослідження [16] Jingjin Li та команди, опубліковане у 2023 році, виявило потенційні ризики для приватності навіть у зашифрованому трафіку голосових помічників. Аналіз метаданих мережевого трафіку дозволяв з точністю 89 % визначити, які команди виконував користувач, навіть без доступу до самого вмісту повідомлень. Це підкреслює необхідність комплексного підходу до забезпечення приватності, включаючи локальну обробку чутливих даних та маскування шаблонів мережевої активності.

Важливо зазначити питання інтеграції голосових помічників з операційними системами. Виділяють три рівні інтеграції: поверхневий, коли помічник може лише запускати програми та імітувати натискання клавіш; середній, з використанням API операційної системи для доступу до файлової системи та системних налаштувань; та глибокий, який передбачає прямий доступ до ядра системи та можливість низькорівневого керування процесами. Глибока інтеграція підвищує функціональність, але вимагає суттєво більших зусиль на розробку та тестування безпеки.

Аналіз користувацького досвіду взаємодії з голосовими помічниками [29], проведений Ya-Ling Wang на основі інтерв'ю з 200 користувачами різних систем, виявив кілька ключових факторів задоволеності. Найважливішими виявилися швидкість відповіді, де затримка понад 2 секунди призводила до різкого зниження задоволеності на 45 %, та точність розуміння команд, де користувачі очікували успішного виконання щонайменше 90 % своїх запитів. Цікаво, що природність голосу помічника мала менший вплив, ніж очікувалося, і користувачі були готові прийняти менш природний синтез за умови високої точності та швидкості виконання команд.

Користувачі найчастіше автоматизують повторювані послідовності дій, які виконують щонайменше 3-4 рази на день. Найпопулярнішими сценаріями автоматизації виявилися операції з файлами (копіювання, переміщення, перейменування), запуск груп програм для специфічних робочих сценаріїв, та

виконання послідовностей команд у термінальних додатках. У середньому автоматизація через голосові команди економила користувачам 40-60 хвилин робочого часу на день.

Огляд існуючих теоретичних та експериментальних досліджень показує, що галузь голосових помічників на основі штучного інтелекту перебуває у стадії активного розвитку. Досягнуто значного прогресу в точності розпізнавання мовлення, природності синтезу голосу та можливостях обробки природної мови. Водночас залишається низка невирішених проблем, особливо для мов з обмеженими ресурсами, забезпечення приватності користувачів та створення глибоко інтегрованих систем для персональних комп'ютерів. Розробка універсального голосового помічника, який поєднує автоматизацію системних операцій з можливістю надання інтелектуальних відповідей на різноманітні запити, залишається актуальним завданням, що вимагає подальших досліджень та розробок.

1.2 Огляд і аналіз методів та засобів розробки голосового помічника для вирішення проблеми дослідження

Розробка голосового помічника на основі штучного інтелекту є комплексним завданням, що вимагає інтеграції різноманітних технологій та методів. Кожен компонент системи, від розпізнавання мовлення до виконання команд в операційній системі, потребує ретельного вибору оптимальних підходів та інструментів. У цьому розділі проаналізовано сучасні методи та засоби, які можуть бути застосовані для створення функціонального голосового помічника для персональних комп'ютерів.

Розпізнавання мовлення є першим та одним з найкритичніших етапів роботи голосового помічника. Перетворення звукового сигналу в текстове представлення безпосередньо впливає на здатність системи правильно розуміти команди користувача. Сучасні методи автоматичного розпізнавання мовлення

можна розділити на декілька основних категорій залежно від архітектури та принципів роботи.

Традиційні підходи до розпізнавання мовлення базувалися на гібридних системах, що поєднують приховані Марковські моделі з Гаусовими сумішами для акустичного моделювання. Такі системи домінували в галузі протягом кількох десятиліть і досі використовуються в деяких комерційних продуктах. Основна ідея полягає в моделюванні мовлення як послідовності станів, де кожен стан відповідає певній фонемі або її частині. Ймовірність переходів між станами та ймовірність спостереження певних акустичних характеристик у кожному стані визначаються на основі навчальних даних.

Глибокі нейронні мережі виявилися набагато ефективнішими за традиційні Гаусові суміші в акустичному моделюванні. Це спричинило хвилю розробок гібридних систем, які поєднували нейронні мережі з прихованими Марковськими моделями. Результати виявилися вражаючими – частота помилок суттєво зменшилася порівняно з попередніми методами. Втім, навіть ці покращені системи залишалися досить громіздкими у використанні. Вони й далі потребували ретельного вирівнювання фонем з акустичними сигналами, а мовну модель доводилося тренувати окремо, що ускладнювало весь процес розробки.

Революційним кроком стала поява end-to-end моделей розпізнавання мовлення, які навчаються безпосередньо перетворювати послідовність акустичних ознак у послідовність символів без проміжних представлень. Архітектура Connectionist Temporal Classification (CTC) дозволила навчати рекурентні нейронні мережі для розпізнавання мовлення без потреби в точному вирівнюванні між аудіо та текстом [2]. Це спростило процес навчання та зробило системи більш гнучкими. Архітектура зображена на рисунку 1.4.

Подальший розвиток призвів до появи моделей на основі механізму уваги, де енкодер обробляє акустичні ознаки, а декодер генерує текст, динамічно фокусуючись на різних частинах вхідної послідовності. Такі моделі продемонстрували можливість досягнення високої точності розпізнавання без складних компонентів традиційних систем. Експериментальні результати

показали, що вони можуть досягати точності, порівнянної або навіть кращої за гібридні системи, особливо при наявності великих обсягів навчальних даних.

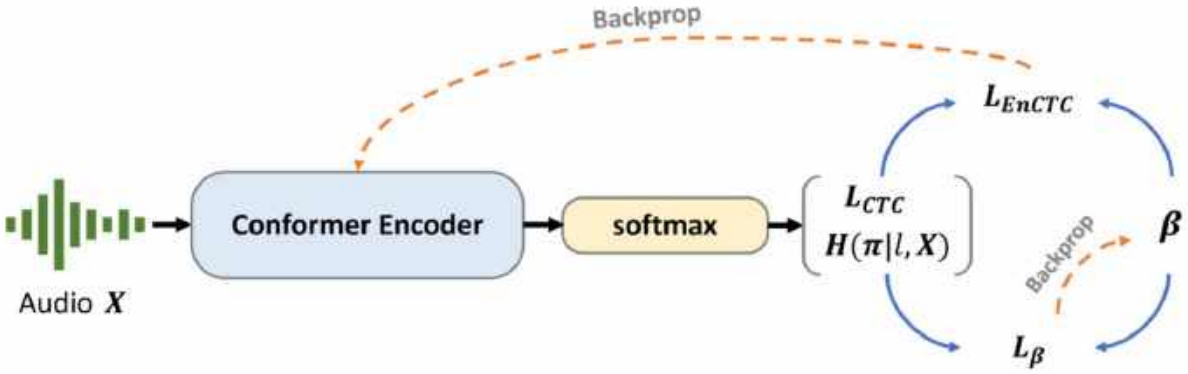

Рисунок 1.4 – Архітектура CTC [1]

Трансформерні архітектури, які спочатку були розроблені для задач машинного перекладу, згодом знайшли застосування і в розпізнаванні мовлення. Дослідження [13] Gulati та команди представило архітектуру Conformer, яка поєднує переваги згорткових та трансформерних шарів для ефективної обробки аудіо. Ця модель демонструє на 10-15 % нижчу частоту помилок порівняно з класичними трансформерами при однаковій кількості параметрів.

Окремої уваги заслуговує модель Whisper, розроблена компанією OpenAI [18]. Це мультимовна та мультизадачна система розпізнавання мовлення, навчена на 680000 годинах слабко анотованих даних з інтернету. Whisper використовує архітектуру енкодер-декодер трансформера і демонструє надзвичайну стійкість до акустичних перешкод та варіативності мовлення. Важливою особливістю є здатність моделі працювати з різними мовами, включаючи мови з обмеженими ресурсами, завдяки великому обсягу багатомовних навчальних даних.

Порівняльне тестування показало, що Whisper досягає результатів, близьких до комерційних систем на англійській мові, а для багатьох інших мов навіть перевершує їх. Особливо важливо, що модель добре справляється з транскрипцією мовлення з акцентами, технічною термінологією та в умовах фонового шуму. Для української мови Whisper демонструє word error rate на рівні

8-12 %, що є значно кращим результатом порівняно з більшістю інших доступних систем.

З практичної точки зору Whisper має кілька переваг для розробки голосового помічника. По-перше, модель є відкритою та може використовуватися локально без необхідності відправлення аудіо на зовнішні сервери, що критично важливо для забезпечення приватності та швидкості. По-друге, існують оптимізовані версії моделі різних розмірів, від tiny з 39 мільйонами параметрів до large з 1550 мільйонами параметрів, що дозволяє вибрати оптимальний баланс між точністю та швидкістю залежно від доступних обчислювальних ресурсів. По-третє, модель підтримує автоматичне визначення мови, що робить можливою багатомовну взаємодію без явного перемикання користувачем.

Альтернативними рішеннями для розпізнавання мовлення є комерційні API від великих технологічних компаній. Google Cloud Speech-to-Text пропонує високу точність розпізнавання для понад 125 мов, включаючи українську [6]. Система використовує найновіші досягнення Google в галузі машинного навчання і постійно оновлюється. Перевагою є можливість налаштування моделі під специфічну термінологію та контекст через механізм адаптації. Недоліками є необхідність постійного інтернет-з'єднання, залежність від зовнішнього сервісу та вартість використання при великих обсягах транскрипції.

Amazon Transcribe пропонує подібні можливості з додатковими функціями, такими як автоматична ідентифікація спікерів та фільтрація нецензурної лексики [25]. Microsoft Azure Speech Service виділяється можливістю створення кастомних голосових моделей, навчених на специфічних даних замовника [8]. Проте всі ці рішення мають спільний недолік для застосування в персональному помічнику – відправлення голосових даних користувача на зовнішні сервери, що створює ризики для приватності та збільшує затримку обробки.

Існують також відкриті рішення на кшталт Mozilla DeepSpeech, яке базується на архітектурі, запропонованій дослідниками Baidu [9]. DeepSpeech використовує рекурентні нейронні мережі з CTC loss та мовну модель на основі n-грам для покращення результатів. Модель є відкритою і може бути дотренована

на власних даних, що корисно для адаптації під специфічну термінологію чи акценти. Проте якість розпізнавання DeepSpeech для української мови поступається Whisper через менший обсяг навчальних даних.

Важливим аспектом вибору методу розпізнавання є також швидкість обробки. Локальні моделі на сучасному апаратному забезпеченні можуть забезпечити затримку 100-300 мс від завершення мовлення до отримання тексту, тоді як хмарні рішення зазвичай мають затримку 500-1500 мс залежно від якості інтернет-з'єднання. Для голосового помічника, де швидкість реакції критично впливає на користувацький досвід, локальна обробка має суттєву перевагу.

Після перетворення звуку в текст наступним критичним етапом є розуміння намірів користувача та визначення відповідних дій. Обробка природної мови включає широкий спектр технологій від простого розбору команд до глибокого семантичного аналізу складних запитів. Вибір методів обробки природної мови безпосередньо впливає на здатність помічника розуміти різноманітні формулювання команд та підтримувати природний діалог.

Традиційний підхід до обробки команд базується на правилах та шаблонах. Система містить набір заздалегідь визначених шаблонів, які зіставляються з введеним текстом для визначення намірів та витягування параметрів. Наприклад, фрази «відкрий файл document.txt», «покажи мені document.txt» та «запусти document.txt» можуть бути розпізнані як намір «відкрити файл» з параметром імені файлу. Перевагою такого підходу є передбачуваність поведінки та можливість точного контролю розробником. Недоліком є негнучкість та необхідність вручну створювати правила для кожного можливого варіанту формулювання команди.

Існують методи для розбору природномовних команд на основі регулярних виразів та граматик. Такі системи добре працюють для обмеженого домену з чітко визначеними командами, але швидко стають неефективними при розширенні функціональності. Для підтримки навіть 50 різних типів команд потрібно створити сотні правил для покриття різних варіантів їх формулювання.

Революцію в обробці природної мови спричинило застосування глибоких нейронних мереж. Рекурентні нейронні мережі, зокрема LSTM та GRU, показали здатність ефективно обробляти послідовності слів з урахуванням контексту. Архітектура двонаправлених LSTM, дозволяє враховувати як попередній, так і наступний контекст кожного слова, що критично важливо для точного розуміння значення [3]. Модель на основі двонаправлених LSTM може досягати 92-94 % точності одночасної класифікації намірів та виділення слотів на датасеті ATIS.

Згорткові нейронні мережі, традиційно використовувані для обробки зображень, також знайшли застосування в задачах класифікації тексту. Відносно проста CNN архітектура може досягати результатів, порівнянних з більш складними рекурентними моделями, при значно меншому часі навчання. Це робить CNN привабливим вибором для задач, де швидкість обробки є критичною.

Найбільший прорив відбувся з появою трансформерної архітектури та моделей попереднього навчання. BERT, представлений Devlin та командою Google, змінив парадигму обробки природної мови [10]. Замість навчання моделі з нуля на специфічному датасеті, BERT спочатку навчається на величезних корпусах тексту для розуміння загальних закономірностей мови, а потім дотреновується на конкретній задачі (рис. 1.5). BERT досягає 92-94 % точності на задачах класифікації намірів, суттєво перевершуючи попередні підходи.

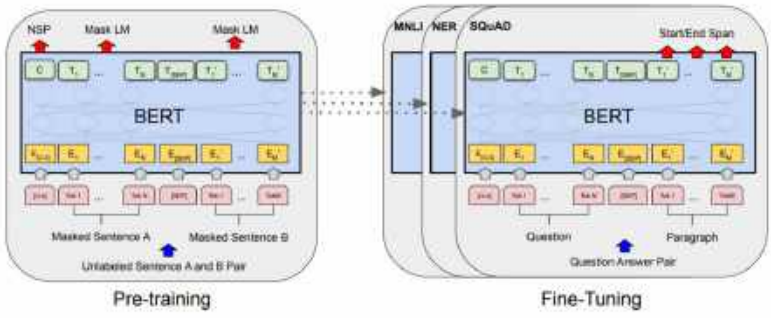

Рисунок 1.5 – Архітектура BERT [10]

Подальший розвиток призвів до появи сімейства моделей GPT від OpenAI. GPT-3 та GPT-4 демонструють здатність розуміти складні інструкції та генерувати природні відповіді на основі промптів без додаткового дотренування [1]. Це відкриває нові можливості для голосових помічників, дозволяючи обробляти не лише команди, а й довільні запитання користувача. Великі мовні моделі з відповідним промптингом можуть досягати точності 93-97 % на складних задачах без жодного додаткового навчання.

Для розробки голосового помічника особливо цікавою є можливість використання API великих мовних моделей через механізм викликів функцій. Claude API від Anthropic надає розвинені можливості для структурованої взаємодії через tool use [28]. Модель може аналізувати запит користувача, визначати необхідні дії та генерувати структуровані виклики відповідних функцій з правильними параметрами. Це дозволяє створити гнучку систему, де розуміння природної мови делегується потужній мовній моделі, а розробник фокусується на реалізації конкретних функцій та інтеграції з операційною системою.

Переваги використання великих мовних моделей через API включають відсутність необхідності збирати навчальні дані та тренувати власні моделі, можливість обробки складних багатокрокових інструкцій, здатність підтримувати природний діалог з урахуванням контексту попередніх повідомлень, та можливість надавати розгорнуті відповіді на інформаційні запитання користувача.

Недоліками підходу з використанням API є залежність від зовнішнього сервісу, необхідність інтернет-з'єднання, та вартість використання при великій кількості запитів. Проте для голосового помічника, який виконує команди на комп'ютері та відповідає на питання, кількість токенів на один запит зазвичай є відносно невеликою, що робить економічну складову прийнятною. Затримка API запитів становить зазвичай 500-2000 мс, що є цілком прийнятним для більшості сценаріїв використання.

Альтернативним підходом є використання локальних відкритих мовних моделей. Моделі сімейства Llama від Meta, Mistral та інші демонструють вражаючі результати при значно меншій кількості параметрів порівняно з закритими комерційними моделями. Llama 3 з 70 мільярдами параметрів досягає результатів, близьких до GPT-3.5, на багатьох бенчмарках [14]. Правильно навчені моделі середнього розміру можуть ефективно конкурувати з набагато більшими моделями на специфічних задачах.

Використання локальних моделей забезпечує повну приватність, оскільки всі дані обробляються на пристрої користувача, відсутність залежності від інтернету та зовнішніх сервісів, та відсутність витрат на API при використанні. Недоліками є необхідність потужного апаратного забезпечення для запуску великих моделей, складність налаштування та оптимізації, та потенційно нижча якість порівняно з найкращими комерційними рішеннями. Для ефективного запуску моделі з 70 мільярдами параметрів потрібно щонайменше 48 гігабайт GPU пам'яті, що доступно лише на високопродуктивних робочих станціях.

Компромісним рішенням може бути гібридний підхід, де прості команди обробляються локально спеціалізованою моделлю класифікації намірів, а складні запитання та багатокрокові інструкції передаються на обробку через API великої мовної моделі. Такий підхід дозволяє значно зменшити затримку для простих запитів при збереженні здатності обробляти складні сценарії через хмарні моделі.

Важливим аспектом обробки природної мови для голосового помічника є підтримка контексту діалогу. Користувач може віддавати послідовні команди, які семантично пов'язані, наприклад «відкрий папку Документи», а потім «знайди там файли Excel». Система повинна розуміти, що «там» посилається на попередньо згадану папку.

Сучасні великі мовні моделі природно підтримують контекст через механізм вікна контексту, куди включається історія діалогу. Claude API підтримує контекстні вікна до 200000 токенів, що дозволяє включити десятки попередніх повідомлень до запиту. Проте необхідно враховувати, що збільшення контексту

підвищує затримку та вартість обробки запиту, тому важливо оптимізувати, які саме частини історії включати в кожен запит.

Після розпізнавання мовлення та визначення наміру, критичним етапом є безпосереднє виконання команд в операційній системі. Інтеграція голосового помічника з операційною системою вимагає надійних механізмів для керування файлами, запуску програм, автоматизації дій та доступу до системних ресурсів. Вибір методів інтеграції безпосередньо впливає на функціональність та безпеку системи.

Базовий рівень інтеграції передбачає використання стандартних можливостей операційної системи через виклики системних команд. Для Unix-подібних систем, включаючи Linux та macOS, це означає виконання команд через shell, а для Windows використання командного рядка або PowerShell. Такий підхід дозволяє реалізувати широкий спектр функціональності з мінімальними зусиллями на розробку, оскільки більшість операційних систем надають потужні інструменти командного рядка.

Проте виконання довільних системних команд створює серйозні ризики безпеки. Якщо система неправильно обробляє введення користувача, зловмисник може через голосові команди виконати небажані дії в системі. Саме тому є необхідність ретельної валідації всіх параметрів перед їх використанням в системних викликах.

Більш безпечним та структурованим підходом є використання API операційної системи через відповідні бібліотеки. Для Python, який є популярною мовою для розробки систем на основі ШІ, існують потужні бібліотеки для взаємодії з ОС. Модуль os та pathlib надають кросплатформні інтерфейси для операцій з файловою системою. Бібліотека subprocess дозволяє безпечно запускати зовнішні програми з контролем введення та виведення. Модуль psutil надає можливості для моніторингу та керування системними процесами.

Для більш глибокої інтеграції можуть використовуватися платформо-специфічні API. На Windows – це Win32 API, доступне через бібліотеку pywin32, яке надає повний доступ до всіх можливостей операційної

системи від керування вікнами до взаємодії з реєстром. На macOS – Foundation та AppKit frameworks, доступні через PyObjC, дозволяють інтегруватися з нативними можливостями системи, включаючи AppleScript для автоматизації програм.

Особливо цікавим для розробки голосового помічника є концепція Model Context Protocol, представлена Anthropic [15]. MCP є відкритим стандартом для інтеграції мовних моделей з зовнішніми інструментами та джерелами даних. Протокол надає уніфікований інтерфейс, через який мовна модель може викликати зовнішні функції, передавати їм параметри та отримувати результати. Це дозволяє створювати модульну архітектуру, де різні аспекти функціональності реалізовані як окремі MCP сервери.

Архітектура MCP складається з трьох основних компонентів: MCP хост, який управляє підключеннями та маршрутизацією запитів; MCP клієнт, який взаємодіє з мовною моделлю; MCP сервери, які реалізують конкретну функціональність. Для голосового помічника це дає можливість створити окремі сервери для різних аспектів взаємодії з операційною системою – один для файлових операцій, інший для керування програмами, третій для веб-пошуку тощо.

Дослідження архітектури на основі MCP показує кілька переваг для розробки голосового помічника. По-перше, модульність дозволяє легко розширювати функціональність додаванням нових серверів без зміни основної логіки. По-друге, стандартизований протокол забезпечує чітке розділення відповідальності між компонентами системи. По-третє, можливість використання існуючих MCP серверів, розроблених спільнотою, значно прискорює розробку.

Для реалізації файлових операцій MCP сервер може надавати такі інструменти як читання та запис файлів, пошук файлів за критеріями, створення та видалення директорій, копіювання та переміщення файлів. Великі мовні моделі можуть визначати необхідні операції на основі природномовних команд та генерувати відповідні виклики інструментів.

Для керування програмами MCP сервер може надавати можливості запуску додатків, закриття процесів, отримання списку запущених програм, переключення між вікнами. Важливим аспектом є також можливість взаємодії з вже запущеними програмами через API автоматизації. На Windows це реалізується через UI Automation, на macOS через Accessibility API, на Linux через AT-SPI.

Окремим викликом є забезпечення надійності виконання команд. Система повинна правильно обробляти ситуації, коли виконання команди неможливе або завершилося помилкою. Рекомендовано реалізувати механізми відкату для операцій, які змінюють стан системи, щоб користувач міг скасувати небажані дії. Також важливо надавати зрозумілі повідомлення про помилки, які мовна модель може перетворити на природномовні пояснення для користувача.

Синтез мовлення є заключним етапом взаємодії, коли система озвучує відповідь користувачу. Вибір методу синтезу впливає на природність звучання, швидкість генерації та можливість роботи в офлайн режимі. Сучасні технології text-to-speech значно еволюціонували від роботизованого звучання до практично людського мовлення.

Традиційні параметричні синтезатори, такі як Festival або eSpeak, генерують мовлення на основі акустичних правил та конкатенації попередньо записаних фрагментів [26]. Вони працюють швидко та не вимагають багато ресурсів, але звучання є неприродним. Для голосового помічника, де якість взаємодії важлива для користувацького досвіду, такі рішення є недостатніми.

Нейронні синтезатори на основі глибоких мереж забезпечують значно вищу якість. Tacotron 2 генерує мел-спектрограми з тексту, які потім перетворюються в аудіо через вокодер WaveGlow [7]. Система досягає показника MOS 4,53, що практично не відрізняється від природного мовлення з оцінкою 4,58. Проте генерація є відносно повільною, що може створювати помітні затримки у відповідях помічника.

FastSpeech 2 вирішує проблему швидкості через використання feed-forward трансформерної архітектури замість авторегресивної генерації.

Дослідження показало можливість генерації мовлення в 50 разів швидше за реальний час при збереженні високої якості [19]. Це робить FastSpeech привабливим вибором для інтерактивних застосувань.

Для локального синтезу мовлення існують готові рішення на кшталт Piper, який забезпечує високоякісний синтез для багатьох мов, включаючи українську [22]. Модель базується на архітектурі VITS та оптимізована для швидкої генерації на процесорах. Експерименти показують можливість генерації мовлення зі швидкістю понад 100 разів швидше за можливості людини на сучасних процесорах.

Альтернативою є використання хмарних сервісів синтезу мовлення. Google Cloud Text-to-Speech, Amazon Polly та Azure Speech Services надають високоякісний синтез з широким вибором голосів та можливістю налаштування інтонації [6]. Проте це створює залежність від інтернету та збільшує затримки відповідей системи.

Для голосового помічника оптимальним є використання локального синтезу, який забезпечує швидку реакцію та приватність. Користувачі чутливі до затримок у голосових відповідях – чим більше користувач очікує, тим більша його оцінка незадоволеності. Локальний синтез дозволяє мінімізувати затримки та забезпечити стабільний користувацький досвід незалежно від якості інтернет-з'єднання.

Узагальнюючи розглянуті методи та засоби, оптимальна архітектура голосового помічника для персональних комп'ютерів має включати локальне розпізнавання мовлення через Whisper для забезпечення приватності та низької затримки обробки голосу, використання Claude API з механізмом tool use для гнучкої обробки природномовних команд та запитів, модульну систему MCP серверів для реалізації різних інтеграцій з операційною системою, та локальний синтез мовлення для швидких природних відповідей. Такий підхід дозволяє поєднати переваги сучасних великих мовних моделей з вимогами до приватності та продуктивності голосового помічника.

1.3 Постановка завдання на кваліфікаційну роботу магістра

На основі проведеного аналізу предметної області та існуючих методів розробки голосових помічників можна сформулювати основну проблему дослідження. Сучасні голосові асистенти, незважаючи на значний прогрес у розпізнаванні мовлення та обробці природної мови, залишаються обмеженими у можливостях автоматизації складних дій на комп'ютері користувача. Більшість комерційних рішень орієнтовані на виконання простих команд або надання інформаційних відповідей, але не здатні повноцінно взаємодіяти з операційною системою для виконання комплексних завдань.

Існуючі голосові помічники працюють переважно в рамках заздалегідь визначеного набору команд або інтеграцій з обмеженою кількістю сервісів. Користувач змушений адаптувати свої запити під можливості системи, а не навпаки. Крім того, більшість сучасних рішень передають голосові дані на віддалені сервери для обробки, що створює питання конфіденційності та збільшує затримки у відповідях. Для української мови ситуація ускладнюється обмеженою підтримкою в комерційних системах та недостатньою кількістю навчальних даних.

Метою даного дослідження є розробка та дослідження голосового помічника на основі штучного інтелекту, здатного як відповідати на довільні запитання користувача, так і автоматизувати дії на комп'ютері через глибоку інтеграцію з операційною системою. Система повинна поєднувати переваги сучасних великих мовних моделей з можливістю локальної обробки критичних компонентів для забезпечення швидкодії та приватності.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

– провести аналіз існуючих голосових помічників та виявити їхні переваги і обмеження в контексті застосування на персональних комп'ютерах;

– дослідити сучасні технології розпізнавання та синтезу мовлення, обробки природної мови та машинного навчання, що застосовуються в голосових системах;

– розробити модульну та розширювану архітектуру голосового помічника, яка забезпечить ефективну взаємодію між компонентами розпізнавання мовлення, обробки природної мови, виконання системних операцій та синтезу відповідей;

– реалізувати локальну підсистему розпізнавання мовлення з підтримкою української мови, яка забезпечить прийнятну точність розпізнавання голосових команд користувача;

– інтегрувати велику мовну модель для інтелектуальної обробки запитів користувача з можливістю розуміння контексту діалогу та прийняття рішень щодо необхідних дій;

– створити систему інтеграції з операційною системою через набір спеціалізованих MCP серверів, які надають доступ до різних функцій комп'ютера;

– розробити механізм перетворення намірів користувача у конкретні виклики функцій системи;

– реалізувати підсистему синтезу мовлення для генерації природних голосових відповідей користувачу;

– провести експериментальне дослідження розробленої системи для оцінки її технічних характеристик.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ГОЛОСОВОГО ПОМІЧНИКА НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Розробка голосового помічника з можливістю автоматизації дій на комп'ютері вимагає прийняття низки рішень щодо архітектури системи, вибору технологій та алгоритмів для реалізації окремих компонентів. Кожне з цих рішень має бути обґрунтованим з урахуванням технічних вимог, обмежень та поставлених задач дослідження.

Перш за все необхідно визначити загальну архітектуру системи та взаємодію її компонентів. Голосовий помічник має складатися з декількох функціональних блоків: модуля активації та запису голосу, підсистеми розпізнавання мовлення, модуля обробки природної мови та прийняття рішень, системи виконання дій в операційній системі, та підсистеми синтезу мовленнєвих відповідей. Ці компоненти повинні працювати послідовно, передаючи результати один одному, але при цьому залишатися достатньо незалежними для можливості заміни окремих частин системи.

Ключовим архітектурним рішенням є вибір між повністю локальною обробкою, повністю хмарним рішенням або гібридним підходом. Повністю локальна обробка забезпечила б максимальну приватність та мінімальні затримки від мережевої взаємодії, проте вимагала б значних обчислювальних ресурсів для роботи великих мовних моделей. Повністю хмарне рішення спростило б вимоги до апаратного забезпечення, але створило б залежність від інтернет-з'єднання та проблеми конфіденційності. Оптимальним видається гібридний підхід, де критичні компоненти, що працюють з голосовими даними користувача, виконуються локально, а для складної обробки природної мови використовуються потужні хмарні моделі. Це дозволить поєднати переваги обох підходів, зберігаючи баланс між продуктивністю, приватністю та якістю

розуміння команд. На рисунку 2.1 зображена діаграма загальної архітектура системи.

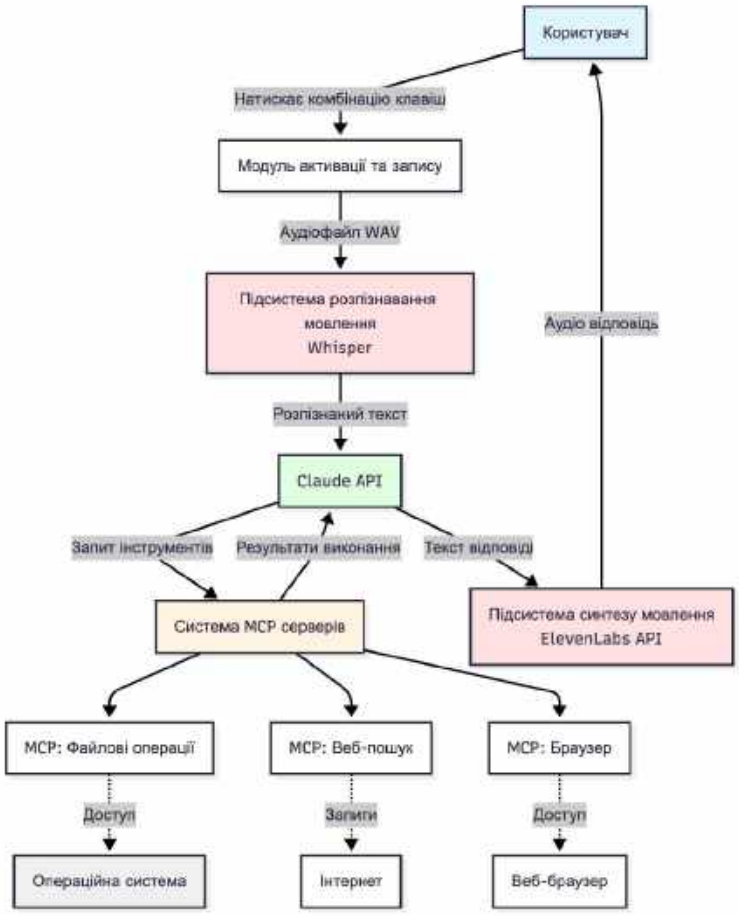

Рисунок 2.1 – Загальна архітектура системи

Важливим аспектом є механізм активації голосового помічника. Існує два основних підходи: постійне прослуховування з активацією за ключовим словом або активація за дією користувача. Перший підхід використовується в більшості комерційних помічників і забезпечує максимальну зручність, оскільки користувач може викликати помічника голосом. Проте цей підхід має істотні недоліки для випадку розробки системи, орієнтованої на складні команди. По-перше, постійне прослуховування створює питання приватності та витрачає обчислювальні ресурси. По-друге, і це найважливіше, автоматичне визначення завершення команди через паузи в мовленні створює серйозні обмеження для користувача.

Під час формулювання складного запиту людина природно робить паузи для обмірковування деталей команди. Наприклад, користувач може сказати «Знайди файл...» і зробити паузу на кілька секунд, намагаючись згадати точну назву. При автоматичному визначенні тиші система інтерпретувала б це як завершення команди і почала б обробку неповного запиту. Це призвело б до помилкових спрацювань та погіршення користувацького досвіду. Тому було прийнято рішення використовувати активацію за допомогою комбінації клавіш на клавіатурі. Користувач натискає призначену комбінацію клавіш для промовляння команди, що явно вказує системі на початок та завершення голосового вводу. Такий підхід надає користувачу повний контроль над процесом запису і дозволяє робити довільні паузи без ризику передчасного завершення команди.

Для розпізнавання мовлення необхідно обрати відповідну модель та підхід до її використання. Серед доступних варіантів можна виділити комерційні API сервіси, такі як Google Cloud Speech-to-Text, Amazon Transcribe або Azure Speech Services, та відкриті моделі для локального розгортання [25]. Комерційні сервіси забезпечують високу точність розпізнавання та підтримку багатьох мов, проте мають суттєві недоліки. Вони вимагають постійного інтернет-з'єднання, передачу голосових даних на сторонні сервери, що створює ризики конфіденційності, та передбачають витрати на використання, які зростають пропорційно до обсягу обробленого аудіо.

Альтернативою є використання відкритих моделей, які можна розгорнути локально. Серед них особливу увагу заслуговує модель Whisper від OpenAI. Ця модель була навчена на величезному датасеті, що включає 680 тисяч годин аудіозаписів різними мовами. Whisper демонструє високу точність розпізнавання для багатьох мов, включаючи українську, де рівень помилок становить приблизно 13 % [18]. Модель доступна у декількох варіантах розміру, від tiny до large, що дозволяє балансувати між точністю та швидкістю обробки залежно від доступних обчислювальних ресурсів. На рисунку 2.2 зображено порівняння показника WER (word error rate) серед різних мов.

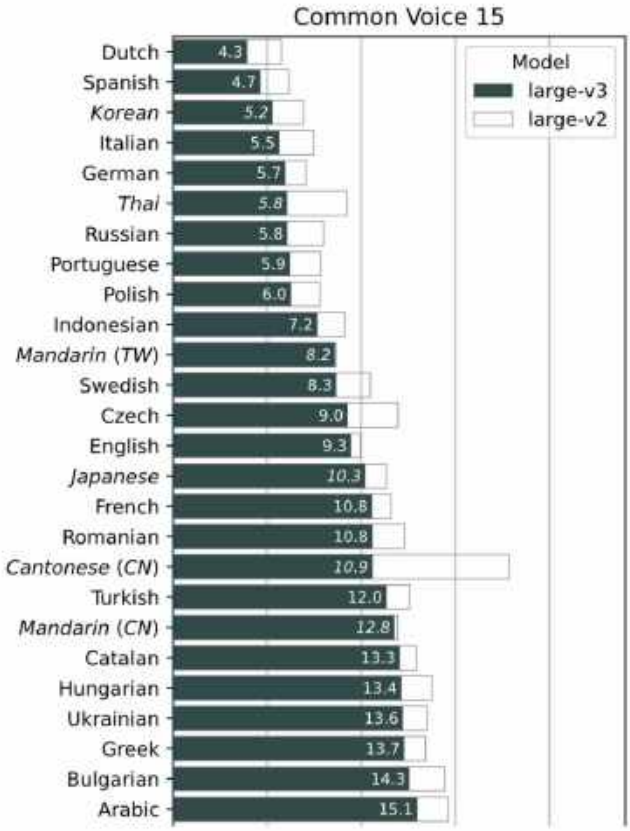

Рисунок 2.2 – Порівняння WER серед різних мов [18]

Перевагами Whisper є відкритий код, що забезпечує прозорість та можливість модифікації, локальна обробка без потреби у мережевому з'єднанні, відсутність витрат на використання, та автоматичне визначення мови вводу. Модель здатна працювати з GPU для прискорення обробки або на CPU для систем без дискретної графічної карти. Для розроблюваного голосового помічника буде використано модель Whisper розміру medium або large, залежно від результатів тестування на цільовому обладнанні. Це забезпечить оптимальний баланс між точністю розпізнавання та швидкістю обробки.

Центральним компонентом системи є модуль обробки природної мови та прийняття рішень. Цей компонент має розуміти намір користувача, визначати, які дії необхідно виконати, і формувати відповідну послідовність команд. Традиційні підходи до обробки команд базувались на розпізнаванні шаблонів та правилах, що вимагало явного програмування кожного можливого типу команди. Такий

підхід є негнучким та не дозволяє обробляти довільні формулювання користувача.

Сучасні великі мовні моделі пропонують принципово інший підхід. Вони здатні розуміти природну мову в широкому контексті, інтерпретувати неоднозначні запити та генерувати структуровані відповіді. Серед доступних моделей можна розглянути як відкриті варіанти, такі як Llama 3 або Mistral, так і комерційні API, зокрема GPT-4, Claude або Gemini.

Відкриті моделі надають повний контроль над даними та не створюють залежності від зовнішніх сервісів. Проте їх розгортання вимагає значних обчислювальних ресурсів. Наприклад, модель Llama 3 з 70 мільярдами параметрів потребує близько 48 гігабайт відеопам'яті для ефективної роботи, що виходить за межі можливостей більшості споживацьких комп'ютерів. Менші версії моделей демонструють гірші результати у складних задачах розуміння та генерації структурованих відповідей.

Комерційні API великих мовних моделей позбавлені цих обмежень. Вони не вимагають потужного локального обладнання, постійно покращуються розробниками та демонструють найвищу якість розуміння природної мови. Серед них особливо виділяється Claude API від Anthropic завдяки механізму tool use. Цей механізм дозволяє моделі не просто генерувати текстові відповіді, а й формувати структуровані виклики зовнішніх функцій на основі опису доступних інструментів.

Принцип роботи tool use полягає в наступному. Системі надається список доступних функцій з описом їх призначення, вхідних параметрів та форматів відповідей. Коли користувач формулює запит, модель аналізує його та визначає, які функції необхідно викликати для виконання завдання. Модель генерує не текстову відповідь, а структурований JSON з назвою функції та її параметрами. Після виконання функції результат повертається моделі, яка може або згенерувати фінальну відповідь користувачу, або ініціювати додаткові виклики функцій для виконання складніших сценаріїв.

Такий підхід ідеально підходить для розроблюваного голосового помічника. Замість програмування логіки розбору команд та їх відображення на системні операції, достатньо описати доступні функції, а модель самостійно визначить необхідні дії. Це забезпечує гнучкість системи та можливість розуміння довільних формулювань користувача. Claude API також демонструє високу точність у задачах розуміння намірів, що критично важливо для коректної інтерпретації команд. Механізм використання tool use зображений на рисунку 2.3.

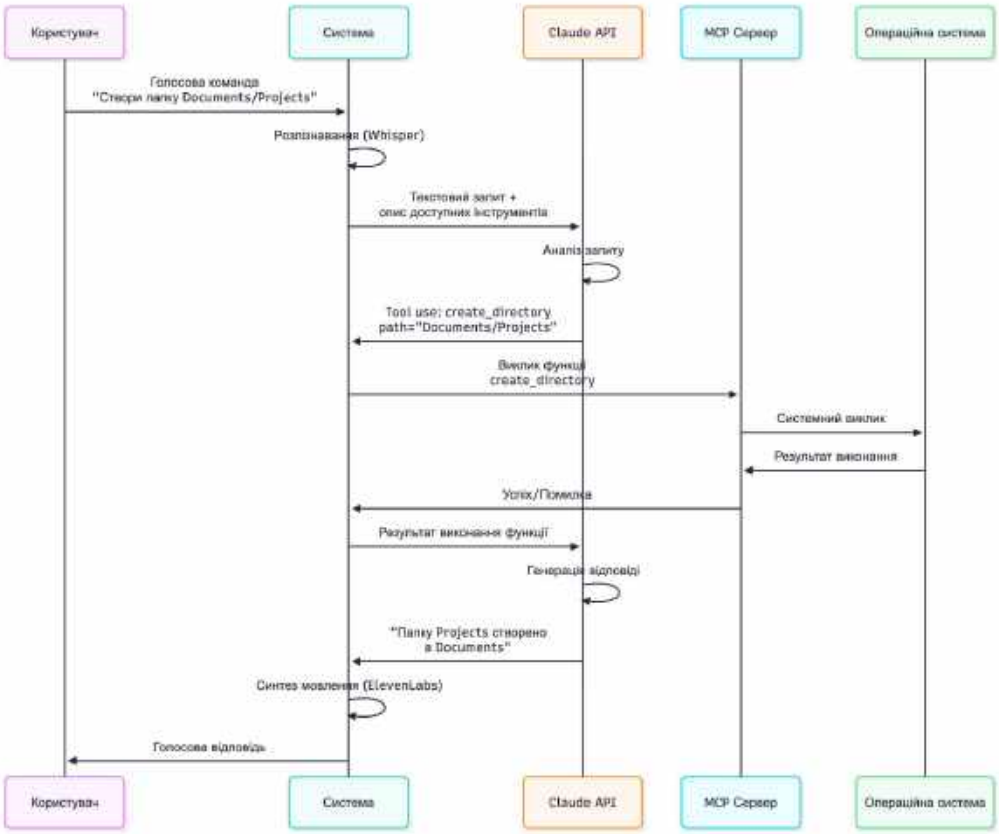

Рисунок 2.3 – Діаграма послідовності використання механізму tool use

Для забезпечення можливості автоматизації дій на комп'ютері необхідно створити систему інтеграції з операційною системою. Існує декілька рівнів такої інтеграції. Найбазовіший підхід передбачає виконання системних команд через shell або PowerShell. Це дозволяє виконувати практично будь-які операції, проте

створює серйозні ризики безпеки та ускладнює обробку результатів, оскільки виведення команд часто є неструктурованим текстом.

Більш безпечний та контрольований підхід полягає у використанні програмних API операційної системи через відповідні бібліотеки. Для Python існують бібліотеки os, subprocess, shutil для базових операцій з файлами та процесами, psutil для отримання системної інформації, та платформо-специфічні бібліотеки для глибокої інтеграції. Цей підхід забезпечує структуровані відповіді, кращу обробку помилок та більший контроль над виконуваними операціями.

Критичним питанням є організація множини різних функцій для взаємодії з системою. Можна реалізувати всі необхідні операції в єдиному монолітному модулі, проте це призведе до складності підтримки та розширення системи. Кращим підходом є модульна архітектура, де різні категорії операцій реалізовані в окремих компонентах, які можна незалежно розробляти, тестувати та оновлювати.

Для реалізації такої модульної архітектури доцільно використати Model Context Protocol від Anthropic. MCP є відкритим стандартом для інтеграції великих мовних моделей з зовнішніми інструментами та джерелами даних. Протокол надає уніфікований інтерфейс для взаємодії між клієнтом мовної моделі та серверами, що надають доступ до різних функцій.

Архітектура MCP включає три основні компоненти. MCP хост – це програма, яка координує взаємодію між моделлю та серверами. MCP клієнт підтримує зв'язок з мовною моделлю та перетворює її запити у виклики функцій. MCP сервери реалізують конкретні набори функцій для різних категорій операцій. Кожен сервер описує доступні інструменти у стандартизованому форматі, що дозволяє моделі динамічно дізнаватись про можливості системи.

Переваги використання MCP є численними. По-перше, модульність – кожен сервер є незалежним компонентом з власною областю відповідальності. По-друге, стандартизація – всі сервери використовують єдиний протокол взаємодії. По-третє, розширюваність – додавання нових можливостей зводиться до створення нового MCP сервера без модифікації існуючого коду.

По-четверте, можливість повторного використання – існують готові MCP сервери для поширених операцій, які можна інтегрувати у систему.

Для розроблюваного голосового помічника планується створити набір спеціалізованих MCP серверів, кожен з яких відповідатиме за певну категорію операцій. Сервер для файлових операцій надаватиме можливості читання, запису, пошуку та маніпуляції файлами. Сервер управління програмами дозволить запускати, зупиняти додатки та взаємодіяти з їх вікнами. Сервер системної інформації надаватиме доступ до даних про стан системи, запущені процеси та використання ресурсів. Сервер для веб-пошуку забезпечить можливість отримання інформації з інтернету для відповідей на запитання користувача. Сервер для взаємодії з браузером дозволить автоматизувати дії у веб-браузері. На рисунку 2.4 зображена архітектура взаємодії системи з MCP серверами.

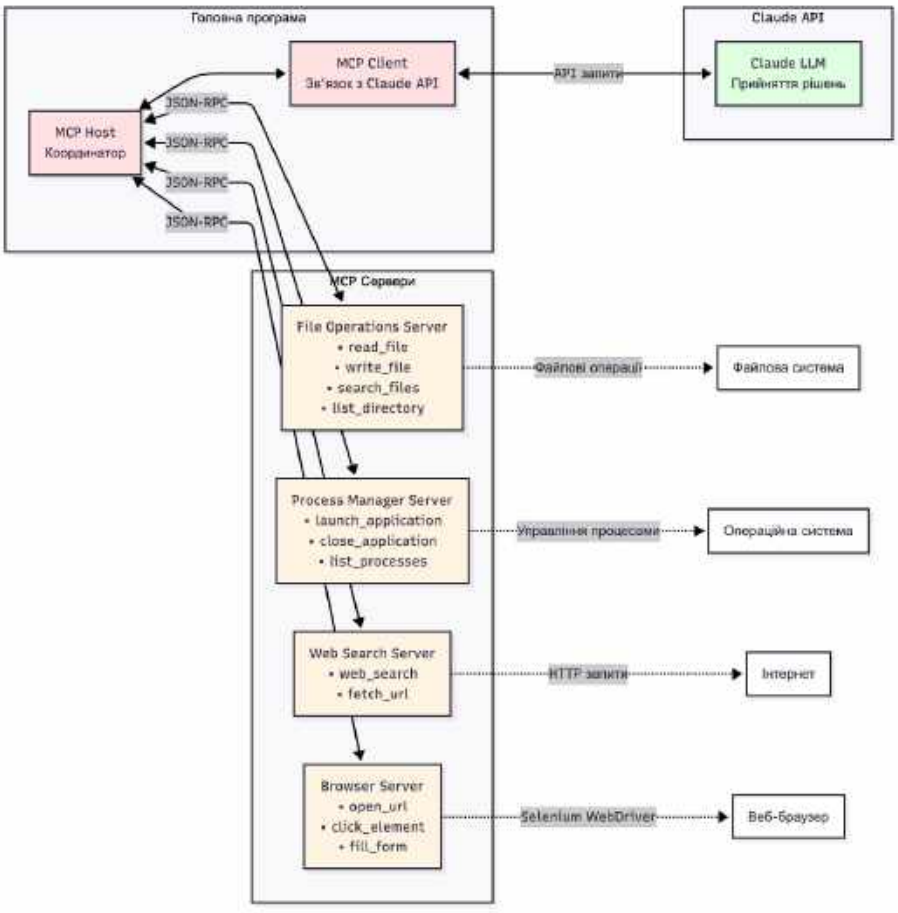

Рисунок 2.4 – Архітектура взаємодії системи з MCP серверами

Кожен MCP сервер має бути реалізований як окремий процес, що комунікує з хостом через стандартні потоки вводу-виводу або через інші транспортні механізми, передбачені протоколом. Сервери описують свої інструменти у форматі JSON Schema, що дозволяє Claude API автоматично розуміти, які параметри приймає кожна функція та яку структуру має її відповідь.

Важливим аспектом є забезпечення безпеки при виконанні операцій в системі. Оскільки голосовий помічник матиме можливість виконувати довільні дії на комп'ютері, необхідно передбачити механізми захисту від потенційно небезпечних операцій. Можливі підходи включають білий список дозволених операцій, запит підтвердження користувача для критичних дій, обмеження доступу до певних директорій та файлів, та логування всіх виконаних операцій для можливості аудиту.

Для розроблюваної системи буде реалізовано комбінований підхід. Безпечні операції, такі як читання файлів, отримання системної інформації або пошук, виконуватимуться автоматично. Потенційно небезпечні операції, такі як видалення файлів, зміна системних налаштувань або встановлення програм, вимагатимуть додаткового підтвердження від користувача. Система також вестиме детальний лог всіх виконаних дій для можливості відстеження проблем.

Останнім компонентом системи є підсистема синтезу мовлення для генерації голосових відповідей користувачу. Існує декілька підходів до синтезу мовлення різного рівня складності та якості. Традиційні параметричні синтезатори, такі як eSpeak, працюють швидко та не вимагають значних ресурсів, проте генерують мовлення, яке звучить механічно та неприродно. Це може негативно впливати на комфорт взаємодії з системою.

Нейронні синтезатори нового покоління, засновані на архітектурі Tacotron або WaveNet, здатні генерувати мовлення, яке за якістю наближається до природного людського голосу. Ці моделі досягають високих оцінок за шкалою Mean Opinion Score, що свідчить про їх сприйняття людьми як природного мовлення. Проте нейронні синтезатори є обчислювально складними та можуть створювати затримки при генерації довгих відповідей.

Для голосового помічника критично важливою є швидкість реакції. Затримка між закінченням команди користувача та початком відповіді системи суттєво впливає на сприйняття інтерактивності. Дослідження показують, що затримки понад дві секунди значно знижують задоволеність користувачів. Тому необхідно обрати рішення для синтезу мовлення, яке забезпечує баланс між якістю звучання та швидкістю генерації.

Серед доступних варіантів можна виділити локальні рішення, такі як Piper на основі VITS, та хмарні сервіси, включаючи ElevenLabs, Google Cloud Text-to-Speech або Amazon Polly. Локальні рішення забезпечують приватність та нульову мережеву затримку, проте можуть поступатися за якістю звучання найкращим хмарним альтернативам, особливо для української мови.

ElevenLabs API представляє сучасний хмарний сервіс синтезу мовлення, що використовує передові нейронні моделі для генерації високоякісних голосів. Сервіс підтримує множину мов, включаючи українську, та пропонує різні голосові профілі з можливістю налаштування інтонації та емоційного забарвлення. Важливою перевагою є висока швидкість генерації – сервіс здатний повертати аудіо практично в реальному часі, що мінімізує затримки у взаємодії.

Для роботи буде використано ElevenLabs API як основне рішення для синтезу мовлення. Це дозволить забезпечити високу якість голосових відповідей та комфортну взаємодію з системою. Вибір хмарного рішення обґрунтовується тим, що затримки на синтез мовлення є менш критичними порівняно з затримками на розпізнавання, оскільки відбуваються після завершення обробки команди. У майбутньому можливе додавання альтернативного локального синтезатора для випадків відсутності інтернет-з'єднання.

Щодо технічної реалізації системи, мовою програмування обрано Python. Цей вибір обумовлений кількома факторами. По-перше, Python має розвинену екосистему бібліотек для машинного навчання та обробки звуку, включаючи PyTorch для роботи з моделлю Whisper, численні HTTP клієнти для взаємодії з API, та бібліотеки для системного програмування. По-друге, Python забезпечує високу швидкість розробки завдяки лаконічному синтаксису та динамічній

типізації. По-третє, більшість документації та прикладів для використовуваних технологій доступні саме для Python.

Для роботи з аудіо будуть використані бібліотеки sounddevice для захоплення звуку з мікрофона та soundfile для збереження аудіоданих у відповідних форматах. Для взаємодії з Claude API застосовуватиметься офіційна бібліотека anthropic, що надає зручний інтерфейс для роботи з механізмом tool use. Глобальне відстеження натискань клавіш для активації помічника реалізовуватиметься через бібліотеку pynput.

Архітектура програми буде організована наступним чином. Головний модуль координуватиме роботу всіх компонентів та забезпечуватиме цикл взаємодії з користувачем. Модуль захоплення звуку відповідатиме за запис голосових команд при активації користувачем. Модуль розпізнавання мовлення виконуватиме транскрибування аудіо через Whisper. Модуль обробки команд взаємодіятиме з Claude API та MCP серверами для інтерпретації та виконання запитів. Модуль синтезу мовлення генеруватиме голосові відповіді та відтворюватиме їх користувачу.

Важливим аспектом є обробка помилок та винятків у роботі системи. Кожен компонент має коректно обробляти можливі проблеми: відсутність інтернет-з'єднання для API запитів, помилки розпізнавання мовлення, невдалі спроби виконання системних операцій, та інші збої. Система має інформувати користувача про виникнення проблем у зрозумілій формі та, де можливо, пропонувати альтернативні шляхи виконання завдання.

Для тестування та налагодження системи передбачається реалізація детального логування роботи всіх компонентів. Логи мають фіксувати час виконання кожного етапу обробки команди, розпізнаний текст запиту, виклики функцій MCP серверів та їх результати, а також будь-які помилки або винятки. Це дозволить аналізувати продуктивність системи та ідентифікувати проблемні місця для подальшого покращення.

Таким чином, обґрунтований вибір технологій та підходів дозволить створити функціональний голосовий помічник, що поєднує переваги сучасних

великих мовних моделей з можливостями глибокої інтеграції з операційною системою через модульну архітектуру MCP серверів. Гібридний підхід до обробки даних забезпечить баланс між продуктивністю, приватністю та якістю розуміння команд користувача.

2.2 Практична реалізація об'єкта проектування

Практична реалізація голосового помічника здійснювалась поетапно, починаючи з базової архітектури та послідовно додаючи функціональні компоненти. Розробка велась мовою програмування Python версії 3.12 з використанням обраних технологій та бібліотек.

Першим кроком була організація структури проекту. Код системи організовано у модульну архітектуру, де кожен компонент відповідає за конкретну функціональність. Основна директорія проєкту містить головний виконуваний модуль, конфігураційні файли та піддиректорії для MCP серверів. Структура директорій проєкту зображена на рисунку 2.5.

Рисунок 2.5 – Структура директорій проєкту

Конфігурація системи винесена в окремий файл, що дозволяє легко змінювати параметри без модифікації коду. Конфігураційний файл містить шляхи до моделі Whisper, налаштування записування аудіо, комбінацію клавіш для активації помічника та параметри логування.

Реалізація захоплення звуку та розпізнавання мовлення є критично важливою для забезпечення якісної роботи системи. Оскільки ці два процеси тісно пов'язані – захоплення аудіо відбувається саме для подальшого розпізнавання – було прийнято рішення об'єднати їх в єдиному компоненті на основі бібліотеки RealtimeSTT.

Для організації взаємодії з різними провайдерами розпізнавання мовлення створено базовий абстрактний клас BaseSTTProvider, який інкапсулює спільну логіку роботи з гарячими клавішами. Цей клас реалізує стандартний цикл роботи голосового вводу: очікування натискання гарячої клавіші для початку запису, захоплення аудіо, очікування повторного натискання для зупинки, та конвертування отриманого запису. Для глобального відстеження натискань клавіш використовується бібліотека pynput, що дозволяє перехоплювати події клавіатури незалежно від того, яка програма має фокус у системі.

В лістингу 2.1 наведений код класу BaseSTTProvider.

Лістинг 2.1 – Код класу BaseSTTProvider

```
class BaseSTTProvider(ABC):
    def __init__(self, hotkey: str = "F12") -> None:
        self._hotkey = hotkey
        self._hotkey_is_pressed = asyncio.Event()
        self._loop: asyncio.AbstractEventLoop | None = None
        self._listener = self._start_listener()

    async def record_and_transcribe(
        self,
        on_recording_start: Callable[[], Any] | None = None,
        on_transcribing_start: Callable[[], Any] | None = None,
    ) -> str:
        if self._loop is None:
            self._loop = asyncio.get_running_loop()
        await self._wait_for_hotkey()
        self._start_recording()
```

```
        if on_recording_start:
            on_recording_start()
        await self._wait_for_hotkey()
        if on_transcribing_start:
            on_transcribing_start()
        text = await self._stop_and_transcribe()
        return text

    async def _wait_for_hotkey(self) -> None:
        """Wait for hotkey press event."""
        self._hotkey_is_pressed.clear()
        await self._hotkey_is_pressed.wait()

    def _start_listener(self) -> keyboard.Listener:
        def on_hotkey() -> None:
            if self._loop is not None:

self._loop.call_soon_threadsafe(self._hotkey_is_pressed.set)
            else:
                self._hotkey_is_pressed.set()
        hotkey = keyboard.HotKey(keyboard.HotKey.parse(self._hotkey),
on_hotkey)
        listener = keyboard.Listener(
            on_press=lambda k: hotkey.press(listener.canonical(k)),
            on_release=lambda k: hotkey.release(listener.canonical(k)),
        )
        listener.start()
        listener.wait()
        return listener
```

Кінець лістингу 2.1

Базовий клас визначає два абстрактних методи, які мають бути реалізовані в конкретних провайдерах: _start_recording для початку захоплення аудіо та _stop_and_transcribe для зупинки запису і отримання тексту. Такий підхід дозволяє легко додавати нові провайдери розпізнавання мовлення без дублювання логіки роботи з гарячими клавішами.

Основним провайдером для системи обрано RealtimeSTTProvider, який використовує бібліотеку RealtimeSTT. Ця бібліотека поєднує функціональність захоплення аудіо та розпізнавання мовлення через моделі Whisper від OpenAI, що спрощує архітектуру системи та зменшує кількість залежностей між компонентами.

Клас RealtimeSTTProvider при ініціалізації створює екземпляр AudioToTextRecorder з налаштуваннями для роботи з обраною моделлю Whisper. Параметри налаштування включають вибір конкретної версії моделі, тип обчислень та пристрій для виконання. Для забезпечення роботи на різному обладнанні використовується квантизація int8 та виконання на центральному процесорі, що дозволяє системі працювати навіть без дискретної графічної карти.

Метод _start_recording викликає відповідний метод об'єкта AudioToTextRecorder для початку захоплення аудіо з мікрофона. Бібліотека автоматично налаштовує параметри запису: частота дискретизації становить 16000 Гц, що є стандартом для моделей розпізнавання мовлення, використовується один канал для монофонічного запису. Аудіодані накопичуються у внутрішньому буфері бібліотеки під час запису.

Метод _stop_and_transcribe зупиняє процес захоплення аудіо та ініціює конвертування записаного фрагменту через модель Whisper. Бібліотека RealtimeSTT автоматично передає аудіодані моделі. Модель також виконує автоматичну нормалізацію аудіо, фільтрацію фонових шумів та виявлення мовленнєвої активності для відсікання тиші. Код класу RealtimeSTTProvider наведений у лістингу 2.2.

Лістинг 2.2 – Код класу RealtimeSTTProvider

```
class RealtimeSTTProvider(BaseSTTProvider):
        super().__init__(hotkey)
        self.__recorder = AudioToTextRecorder(
            model=model,
            spinner=False,
            level=logging.CRITICAL,
            compute_type="int8",
            device="cpu",
        )

    @override
    def _start_recording(self) -> None:
        self.__recorder.start()

    @override
    async def _stop_and_transcribe(self) -> str:
        # Stop recording
```

```
        self.__recorder.stop()

        # Get transcription
        text = self.__recorder.text()

        return text if isinstance(text, str) else ""
```

Кінець лістингу 2.2

Для зручності користувача система виводить у консоль повідомлення про поточний стан обробки. Коли користувач натискає гарячу клавішу, з'являється повідомлення про початок запису з індикацією того, що система слухає. Під час цього аудіо безперервно записується, що дозволяє користувачу робити паузи в мовленні для обміркування формулювання команди. Після натискання клавіші ще раз, виводиться повідомлення про завершення запису та початок конвертування мовлення в текст. Коли розпізнавання завершене, система показує отриманий текст команди перед її подальшою обробкою.

Тестування модуля розпізнавання мовлення показало високу точність для української мови. При чіткій вимові та відсутності фонових шумів практично всі слова розпізнаються коректно. Навіть за наявності помірних шумів або акценту система демонструє прийнятні результати. Середній час обробки аудіофрагментів становить близько 1 секунди на процесорі Apple M1 Pro без використання GPU. На рисунку 2.6 зображено як система розпізнає текст.

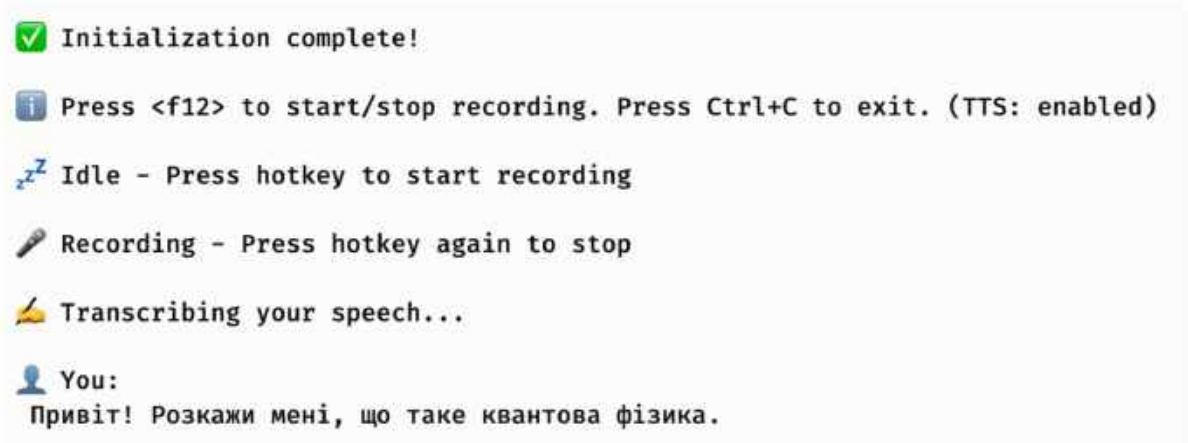

Рисунок 2.6 – Результат розпізнавання мовлення

Центральним компонентом системи є модуль обробки команд через Claude API з інтеграцією MCP серверів. Реалізація цього модуля вимагала найбільше зусиль через складність взаємодії між різними компонентами та необхідність забезпечення діалогу з користувачем. Клас AIProcessor координує роботу з мовною моделлю та виконання функцій через MCP клієнт.

При ініціалізації AIProcessor створюється асинхронний клієнт для взаємодії з Claude API через бібліотеку Anthropic, зберігається посилання на MCP клієнт для виконання інструментів, та ініціалізується порожня історія діалогу. MCP клієнт відповідає за комунікацію з усіма MCP серверами, отримання описів доступних інструментів та виконання викликів функцій. Кожен MCP сервер запускається як окремий процес, з яким MCP клієнт комунікує через стандартні потоки вводу-виводу або інші транспортні механізми протоколу. Ініціалізація класу AIProcessor наведена у лістингу 2.3.

Лістинг 2.3 – Ініціалізація класу AIProcessor

```
class AIProcessor:
    def __init__(
        self,
        model: str,
        mcp_client: MCPClient,
    ) -> None:
        self.client = AsyncAnthropic()
        self.model: str = model
        self.mcp_client: MCPClient = mcp_client
        self.conversation_history: list[MessageParam] = []
```

Кінець лістингу 2.3

Ключовою особливістю системи є детально розроблений системний промпт, який визначає поведінку голосового помічника. Промпт встановлює критичні правила для голосової взаємодії: відповіді мають бути максимально стислими, зазвичай не більше трьох речень, уникати формального стилю та звучати природно, як у розмові з другом. Заборонено використання маркованих списків, нумерації або спеціального форматування у відповідях, оскільки вони

погано сприймаються при голосовому виведенні. Замість формальних фраз система має використовувати розмовні конструкції та скорочення.

Промпт також визначає стиль роботи з інструментами. Виконання інструментів має відбуватись непомітно для користувача – система підтверджує результати, а не процес виконання. Наприклад, замість фрази «Зараз я прочитаю файл» помічник просто повідомляє знайдену інформацію. Для успішних операцій використовуються короткі підтвердження, а у разі помилок система має бути прямою та корисною, пояснюючи причину проблеми однією фразою. Системний промпт наведений у лістингу 2.4.

Лістинг 2.4 – Системний промпт

```
SYSTEM_PROMPT = """You are a helpful voice assistant that can perform
actions on the user's computer.

You MUST ALWAYS follow ALL rules from the <rules> section.

<rules>
## CRITICAL VOICE INTERACTION RULES
- Keep ALL responses under 3 sentences when possible
- Speak naturally as if talking to a friend - avoid formal or robotic
language
- Never use bullet points, numbered lists, or special formatting in your
responses

## RESPONSE STYLE
- Instead of: "I have successfully completed the operation" → Say: "Done"
- Instead of: "The file contains: 1. Item 2. Item" → Say: "The file has
two items, the first one and the second one"
- Instead of: "Please note that..." → Say: "Just so you know..."
- **NEVER** use markdown formatting in your responses. The text must have
**paragraph style**. If you have lists transform them into paragraphs.
- For successful actions: Use brief confirmations like "Got it", "Done",
"All set", "Updated"
- For errors: Be direct and helpful: "I can't do that because..." or
"That didn't work, try..."

## TOOL USAGE
- Alaways execute tools silently - confirm results, not the process
- Example: Don't say "I'm now reading the file", just report what you
found
- Chain related actions smoothly without narrating each step
- If you can't execute an action just with one tool try to chain a couple
of tools to achieve user's goals.
```

```
## HANDLING UNCERTAINTY
- If unclear: Ask ONE specific clarifying question
- If you can't do something: Say why in one sentence and suggest an
alternative
- Don't explain your limitations unless directly asked
</rules>

Remember: You're speaking to someone, not writing an essay. Be helpful,
quick, and conversational."""
```

Кінець лістингу 2.4

Метод process_command є основним місцем обробки користувацьких запитів. Він приймає розпізнаний текст команди та опціональні callback-функції для сповіщення про різні етапи обробки. Метод працює асинхронно, що дозволяє ефективно обробляти мережеві запити до Claude API та виклики MCP інструментів без блокування основного потоку програми.

На початку обробки повідомлення користувача додається до історії діалогу. Це забезпечує контекстне розуміння команд – система пам'ятає попередні звернення та може коректно інтерпретувати посилання на раніше згадані об'єкти або дії. Далі через MCP клієнт отримується список доступних інструментів у форматі JSON Schema, який передається Claude API разом із запитом.

Обробка відповіді Claude організована у вигляді циклу, оскільки виконання однієї команди може вимагати кількох ітерацій взаємодії з моделлю. Встановлено обмеження на максимальну кількість ітерацій для запобігання нескінченних циклів у разі помилок. На кожній ітерації відправляється запит до Claude API з поточною історією повідомлень, системним промптом та списком доступних інструментів. В лістингу 2.5 наведений код методу process_command.

Лістинг 2.5 – Код методу process_command

```
async def process_command(
    self, user_input: str, on_tool_call = None, on_tool_result = None,
on_text_message = None) -> str:
    messages: list[MessageParam] = [
        *self.conversation_history,
        {"role": "user", "content": user_input},
    ]
    available_tools = await self.mcp_client.get_tool_definitions()
```

```
    final_text: list[str] = []
    max_iterations = 20  # Prevent infinite loops
    for _ in range(max_iterations):
        response = await self.client.messages.create(
            model=self.model, max_tokens=800, system=SYSTEM_PROMPT,
            messages=messages, tools=available_tools)
        assistant_content: list[ContentBlock] = []
        for content in response.content:
            if isinstance(content, TextBlock):
                if on_text_message and content.text and
content.text.strip():
                    await on_text_message(content.text)
                final_text.append(content.text)
                assistant_content.append(content)
            elif isinstance(content, ToolUseBlock):
                tool_name = content.name
                tool_args = content.input
                if on_tool_call:
                    on_tool_call(tool_name, tool_args)
                result = await self.mcp_client.execute_tool(tool_name,
tool_args)
                if on_tool_result:
                    on_tool_result(result)
                assistant_content.append(content)
                messages.append({"role": "assistant", "content":
assistant_content})
                messages.append(
                    {
                        "role": "user",
                        "content": [
                            {
                                "type": "tool_result",
                                "tool_use_id": content.id,
                                "content": result,
                            }
                        ],
                    }
                )
                assistant_content = []
                break
    return "\n".join(final_text)
```

Кінець лістингу 2.5

Відповідь Claude API може містити декілька типів контенту. Текстові блоки (TextBlock) являють собою пряму відповідь користувачу – це може бути підтвердження виконання дії, надана інформація або запит уточнення. Коли система отримує текстовий блок, вона викликає callback on_text_message, що

дозволяє головному модулю негайно почати синтез мовлення та відтворення відповіді, не чекаючи завершення всієї обробки. Текст також накопичується для формування фінальної відповіді.

Блоки використання інструментів (ToolUseBlock) сигналізують про те, що Claude модель визначила необхідність виконання певної функції. Такий блок містить назву інструменту та словник параметрів у форматі JSON. Коли система отримує ToolUseBlock, вона викликає callback on_tool_call для сповіщення про початок виконання інструменту, потім передає запит до MCP клієнта для фактичного виконання функції.

MCP клієнт виконує запитану функцію через відповідний сервер та повертає результат у вигляді JSON рядка. Результат передається callback-функції on_tool_result для логування або відображення користувачу, а потім додається до історії повідомлень з типом tool_result. Це дозволяє Claude проаналізувати результат виконання інструменту та прийняти рішення про наступні дії.

У деяких випадках виконання команди вимагає послідовності викликів різних функцій. Наприклад, для команди «знайди всі PDF файли в папці Документи та відкрий найновіший» Claude спочатку викличе інструмент пошуку файлів, отримає список результатів, проаналізує дати модифікації, визначить найновіший файл, а потім викличе інструмент відкриття файлу з відповідним шляхом. Весь цей процес відбувається автоматично завдяки циклічній структурі обробки – після кожного виконання інструменту результат повертається до моделі, яка може ініціювати наступний виклик або згенерувати фінальну відповідь.

Цикл обробки продовжується доти, доки Claude не поверне відповідь без блоків використання інструментів. Це сигналізує про завершення виконання команди. На цьому етапі остання відповідь асистента додається до історії діалогу, що забезпечує збереження контексту для наступних команд користувача.

Важливою особливістю реалізації є підтримка контексту діалогу через поле conversation_history. Система зберігає всю історію повідомлень у поточній сесії, включаючи повідомлення користувача, відповіді асистента та результати

виконання інструментів. Це дозволяє Claude розуміти контекст попередніх команд та звернень. Користувач може задавати уточнюючі питання або давати додаткові інструкції, посилаючись на попередні дії. Наприклад, після команди «відкрий мій останній документ» користувач може сказати «тепер закрий його», і система коректно зрозуміє, про який документ йдеться, оскільки інформація про відкритий файл міститься в історії діалогу.

Обмеження max_tokens встановлено на рівні 800 токенів, що є достатнім для генерації стислих голосових відповідей згідно з вимогами системного промпту. Це запобігає генерації надмірно довгих відповідей, які були б незручними для голосового виведення та сприйняття користувачем.

Реалізація MCP серверів є модульною – кожен сервер являє собою окремий Python скрипт, що реалізує протокол MCP. Для управління підключеннями до серверів та виконання інструментів створено клас MCPClient, який централізує всю логіку взаємодії з MCP екосистемою.

Клас MCPClient при ініціалізації створює порожні структури даних для зберігання сесій з серверами, реєстру інструментів та їх визначень. Поле sessions містить словник активних підключень до MCP серверів, де ключем є ім'я сервера, а значенням – об'єкт ClientSession. Поле tool_registry відображає назви інструментів на імена серверів, які їх надають, що дозволяє швидко визначати, який сервер викликати для конкретного інструменту. Поле tool_definitions зберігає кешовані описи інструментів у форматі, сумісному з Claude API. Для коректного управління ресурсами використовується AsyncExitStack, що забезпечує належне закриття всіх з'єднань при завершенні роботи. Ініціалізація класу MCPClient наведена у лістингу 2.6.

Метод connect_server відповідає за встановлення підключення до MCP сервера. Він приймає назву сервера, команду для його запуску, аргументи командного рядка та змінні середовища. Сервер запускається як окремий процес з комунікацією через стандартні потоки вводу-виводу згідно з протоколом MCP. Після успішного запуску створюється транспорт для stdio комунікації та ініціалізується сесія клієнта.

Лістинг 2.6 – Ініціалізація класу MCPClient

```
class MCPClient:
    """Manages MCP server connections and tool execution."""

    def __init__(self) -> None:
        """Initialize MCP client."""
        self.sessions: dict[str, ClientSession] = {}
        self.tool_registry: dict[str, str] = {}  # tool_name ->
server_name
        self.tool_definitions: list[ToolUnionParam] = []
        self.exit_stack: AsyncExitStack = AsyncExitStack()
```

Кінець лістингу 2.6

Важливим етапом підключення є отримання списку доступних інструментів від сервера через виклик list_tools. Кожен інструмент реєструється в tool_registry для швидкого пошуку, а його визначення у форматі JSON Schema кешується в tool_definitions. Це дозволяє ефективно передавати повний список доступних інструментів до Claude API без повторних запитів до серверів. При успішному підключенні система виводить повідомлення з переліком отриманих інструментів для інформування про розширення можливостей помічника. Код методу connect_server наведений у лістингу 2.7.

Лістинг 2.7 – Код методу connect_server

```
async def connect_server(
    self,
    server_name: str,
    command: str,
    args: list[str] | None = None,
    env: dict[str, str] | None = None,
) -> None:
    if args is None:
        args = []
    if env is None:
        env = {}
    try:
        server_params = StdioServerParameters(
            command=command,
            args=args,
            env=env,
        )
        stdio_transport = await self.exit_stack.enter_async_context(
```

```
            stdio_client(server_params)
        )
        stdio, write = stdio_transport
        session = await self.exit_stack.enter_async_context(
            ClientSession(stdio, write)
        )
        _ = await session.initialize()
        self.sessions[server_name] = session
        response = await session.list_tools()
        tools = response.tools
        for tool in tools:
            self.tool_registry[tool.name] = server_name
            self.tool_definitions.append(
                {
                    "name": tool.name,
                    "description": tool.description or "",
                    "input_schema": tool.inputSchema,
                }
            )
        print(
            f"\nConnected to {server_name} with tools:",
            [tool.name for tool in tools],
        )
    except Exception as e:
        print(f"Warning: Failed to connect to {server_name}: {e}")
        raise
```

Кінець лістингу 2.7

Метод execute_tool виконує виклик конкретного інструменту через відповідний MCP сервер. Спочатку відбувається пошук сервера, який надає запитаний інструмент, через tool_registry. Якщо інструмент або сервер не знайдені, повертається повідомлення про помилку. Фактичний виклик виконується через метод call_tool об'єкта сесії, який серіалізує параметри згідно з протоколом MCP, відправляє запит серверу та очікує на відповідь.

Результат виконання інструменту може містити різні типи контенту. Найпоширенішим є текстовий контент (TextContent), який просто додається до результуючого рядка. Інші типи контенту перетворюються в рядкове представлення. Всі частини результату об'єднуються в єдиний рядок, який повертається для подальшої передачі до Claude API. Обробка помилок забезпечує інформативні повідомлення про проблеми при виконанні інструментів. Код методу execute_tool наведений у лістингу 2.8.

Лістинг 2.8 – Код методу execute_tool

```
async def execute_tool(
    self,
    tool_name: str,
    tool_input: dict[str, object],
) -> str:
    server_name = self.tool_registry.get(tool_name)
    if not server_name:
        return f"Error: Tool {tool_name} not found in any connected
server"

    session = self.sessions.get(server_name)
    if not session:
        return f"Error: Server {server_name} not connected"

    try:
        result = await session.call_tool(tool_name, tool_input)
        content_parts: list[str] = []
        for item in result.content:
            if isinstance(item, TextContent):
                content_parts.append(item.text)
            else:
                content_parts.append(str(item))
        return "\n".join(content_parts)
    except Exception as e:
        return f"Error executing tool {tool_name}: {e}"
```

Кінець лістингу 2.8

Метод get_tool_definitions повертає кешований список визначень інструментів у форматі, сумісному з Claude API. Цей метод викликається при кожному запиті до моделі для передачі інформації про доступні можливості системи. Використання кешу замість повторних запитів до серверів значно прискорює обробку команд.

Для розроблюваного голосового помічника було прийнято рішення створити власний MCP сервер shell-mcp для виконання команд в терміналі, а для інших категорій операцій використовувати готові сервери, розроблені спільнотою. Таке рішення обґрунтовується кількома факторами. По-перше, виконання команд в терміналі є універсальним інструментом, що дозволяє виконати переважну більшість дій в системі, які може запитувати користувач – від файлових операцій до управління програмами та системними

налаштуваннями. По-друге, розробка окремих серверів для кожної категорії задач призвела б до дублювання функціональності та ускладнення підтримки коду. По-третє, існуюча екосистема MCP вже містить якісні реалізації серверів для веб-пошуку, роботи з файловою системою, взаємодії з базами даних та іншими сервісами.

Сервер shell-mcp реалізовано з використанням фреймворку FastMCP, що спрощує створення MCP серверів та забезпечує відповідність протоколу. Сервер надає єдиний інструмент execute_command, який приймає довільну команду оболонки як рядок та виконує її через модуль subprocess стандартної бібліотеки Python. Код наведений у лістингу 2.9.

Лістинг 2.9 – Код shell-mcp

```
import subprocess
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("shell")

@mcp.tool(
    name="execute_command",
    description="Execute an arbitrary shell command and return its output
(stdout, stderr, and exit code)",
)
async def execute_command(command: str) -> str:
    """Execute an arbitrary shell command and return its output.

    Args:
        command: The shell command to execute
    """
    try:
        result = subprocess.run(
            command, shell=True, capture_output=True, text=True,
timeout=30.0
        )
        output: list[str] = []
        if result.stdout:
            output.append(f"STDOUT:\n{result.stdout}")
        if result.stderr:
            output.append(f"STDERR:\n{result.stderr}")
        output.append(f"Exit code: {result.returncode}")
        return "\n\n".join(output) if output else "Command executed with
no output"
    except subprocess.TimeoutExpired:
        return "Error: Command timed out after 30 seconds"
```

```
    except Exception as e:
        return f"Error executing command: {str(e)}"

def main():
    """Initialize and run the server."""
    mcp.run(transport="stdio")

if __name__ == "__main__":
    main()
```

Кінець лістингу 2.9

Важливою особливістю архітектури є можливість підключення сторонніх MCP серверів через конфігураційний файл без модифікації коду голосового помічника. Конфігураційний файл містить секцію з описом серверів, де для кожного вказується назва, команда запуску, аргументи та змінні середовища. При ініціалізації системи всі сервери з конфігурації автоматично підключаються через MCPClient.

Останнім компонентом системи є модуль синтезу мовлення. Клас ElevenLabsTTS відповідає за генерацію голосових відповідей через ElevenLabs API. При ініціалізації вказується API ключ та обирається голосовий профіль. ElevenLabs пропонує різні голоси з можливістю налаштування швидкості промовляння, висоти тону та емоційного забарвлення.

Метод speak приймає текст відповіді та генерує аудіофайл. Запит відправляється до API ElevenLabs, який повертає аудіодані. Отримане аудіо відтворюється за допомогою функції play, яка відтворює аудіо на пристроях виведення звуку користувача. Код класу ElevenLabsTTS наведений у лістингу 2.10.

Лістинг 2.10 – Код класу ElevenLabsTTS

```
class ElevenLabsTTS:
    def __init__(
        self,
        api_key: str,
        model: str,
        voice: str,
    ) -> None:
        self.client = ElevenLabs(api_key=api_key)
```

```
        self.model = model
        self.voice = voice
        self.lock = asyncio.Lock()

    async def speak(self, text: str) -> None:
        if not text or not text.strip():
            return

        async with self.lock:
            try:
                # Stream audio for lower latency
                audio_stream = self.client.text_to_speech.stream(
                    text=text,
                    voice_id=self.voice,
                    model_id=self.model,
                )

                # Play the streamed audio
                play(audio_stream)

            except Exception as e:
                # If TTS fails, we still want to continue
                # The text has already been displayed to the user
                raise RuntimeError(f"TTS error: {e}")
```

Кінець лістингу 2.10

Час генерації аудіо через ElevenLabs API залежить від довжини тексту. Для коротких відповідей в одне-два речення затримка становить 300-500 мс, що є цілком прийнятним для інтерактивної взаємодії. Якість синтезованого мовлення висока – голос звучить природно, з коректними інтонаціями та наголосами для української мови.

Головний модуль програми об'єднує всі компоненти в єдину систему. При запуску програми ініціалізуються всі модулі: завантажується модель Whisper, запускаються MCP сервери, створюється клієнт Claude API, налаштовується модуль синтезу мовлення. Система виводить повідомлення про готовність до роботи та інструкції для користувача щодо активації голосового вводу.

Головний цикл програми очікує активації від користувача. Коли користувач натискає призначену комбінацію клавіш, починається запис аудіо. Після натискання клавіш знову аудіо зберігається та передається модулю розпізнавання

мовлення. Розпізнаний текст виводиться в консоль для інформування користувача про те, як система зрозуміла команду.

Далі текст команди передається модулю обробки для інтерпретації через Claude API. Під час обробки система виводить індикацію стану – «Processing your request». Якщо команда вимагає виконання функцій через MCP сервери, у консолі відображається інформація про виклики інструментів з їх параметрами та результатами. Це дозволяє користувачу бачити, що саме робить система.

Після завершення обробки команди Claude генерує текстову відповідь, яка також виводиться в консоль. Текст передається модулю синтезу мовлення для озвучування. Користувач отримує як текстову, так і голосову відповідь від помічника. Після цього система повертається в режим очікування наступної команди.

Було розглянуто конкретні приклади використання голосового помічника для демонстрації його функціональних можливостей.

Наприклад, користувач активує помічника та каже «Яка зараз погода в Луцьку?». Система розпізнає команду, Claude API визначає, що необхідно виконати веб-пошук, викликає відповідну функцію MCP сервера веб-пошуку. Результати пошуку аналізуються, і помічник відповідає синтезованим голосом з актуальною інформацією про погоду. На рисунку 2.7 зображено результат виконання команди.

Рисунок 2.7 – Запит про погоду в Луцьку

Ще один приклад, користувач запитує «Скільки є вільного місця на диску?». Claude викликає функцію execute_command, отримує дані про використання дискового простору та формує відповідь: «На диску вільно 107 гігабайтів». Результат зображений на рисунку 2.8.

Рисунок 2.8 – Перевірка вільного місця на диску

Також був розглянутий більш складний сценарій, де потрібно виконати декілька кроків. Користувач каже: «Створи файл з назвою нагадування, напиши в ньому купити молоко, і відкрий його». Claude розбиває цю команду на послідовність дій: спочатку викликає write_file для створення файлу з відповідним вмістом, потім викликає execute_command з шляхом до створеного файлу для його відкриття. Кожен крок виконується послідовно, і користувач отримує підтвердження про виконання всієї команди. Результат зображений на рисунку 2.9.

У результаті практичної реалізації створено повнофункціональний голосовий помічник, що демонструє можливості автоматизації дій на комп'ютері через природномовні команди. Система здатна обробляти широкий спектр завдань – від простих інформаційних запитів до складних послідовностей операцій з файлами, програмами та системними ресурсами. Модульна

архітектура на базі MCP забезпечує розширюваність системи та можливість додавання нових функцій без перебудови існуючого коду. Інтеграція Claude API з механізмом tool use дозволяє системі розуміти довільні формулювання користувача та самостійно визначати необхідні дії для виконання запитів.

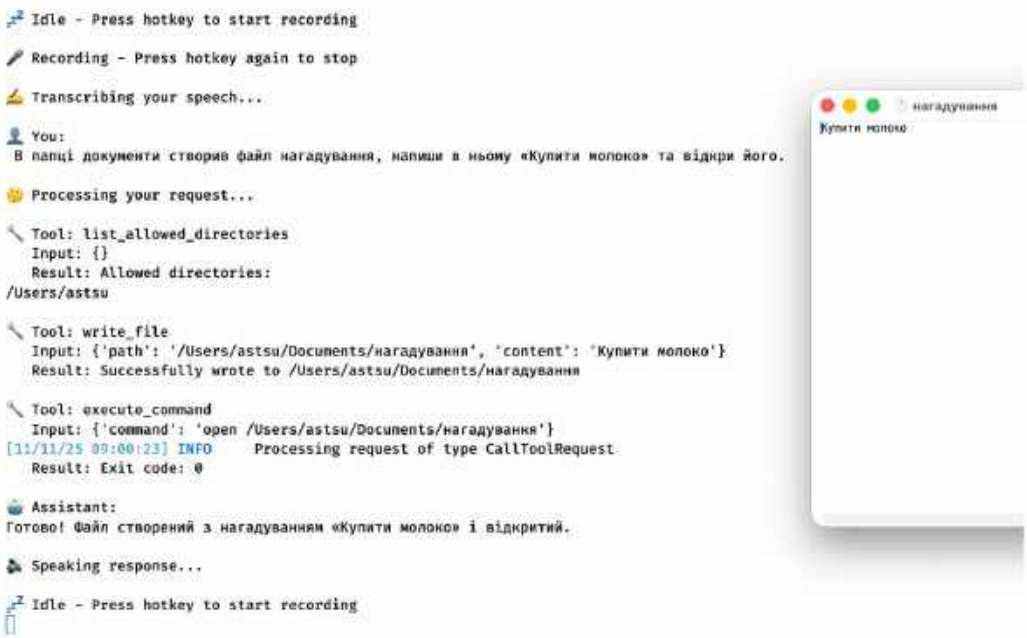

Рисунок 2.9 – Створення та відкриття файлу

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ ГОЛОСОВОГО ПОМІЧНИКА НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Методика проведення дослідження

Експериментальне дослідження розробленого голосового помічника проводилось з метою оцінки технічних характеристик системи, перевірки коректності виконання команд різних типів та визначення практичної придатності для повсякденного використання. Дослідження складалось з декількох взаємопов'язаних етапів, кожен з яких фокусувався на окремих аспектах функціонування системи.

Перший етап передбачав детальне вимірювання технічних характеристик системи, зокрема затримок на кожному етапі обробки голосової команди. Для цього в код програми було інтегровано систему логування з високою точністю вимірювання часу через модуль time.perf_counter. Кожна операція супроводжувалась записом часових міток початку та завершення, що дозволило отримати детальну картину розподілу затримок. Вимірювались наступні показники: час ініціалізації моделі Whisper при першому запуску, час розпізнавання мовлення для фраз різної тривалості, затримка звернення до Claude API та отримання відповіді, час генерації синтезованого мовлення через ElevenLabs, час виконання команд через MCP сервери. Всі вимірювання проводились на процесорі Apple M1 Pro з 16 ГБ оперативної пам'яті, що представляє типову конфігурацію сучасного ноутбука. Для кожного типу операції виконувалось мінімум двадцять вимірювань з подальшим розрахунком середніх значень та стандартного відхилення.

Другий етап був присвячений дослідженню точності розпізнавання української мови моделлю Whisper. Підготовлено тестовий набір голосових команд різної складності та структури. До набору увійшли прості однослівні команди, короткі фрази з двох-трьох слів, складні речення з підрядними конструкціями, команди з технічною термінологією, запити з числовими

значеннями та спеціальними символами. Кожна команда з тестового набору промовлялась тричі в різних умовах: у тихому приміщенні без фонових шумів, за наявності помірного фонового шуму від роботи вентилятора та клавіатури, при швидкому темпі мовлення з менш чіткою артикуляцією. Результати розпізнавання порівнювались з еталонним текстом для розрахунку word error rate та визначення найбільш проблемних категорій команд. Окрема увага приділялась аналізу помилок розпізнавання специфічних для української мови конструкцій, таких як відмінювання слів та складні словосполучення.

Третій етап методики фокусувався на дослідженні якості виконання команд різних категорій. Команди було класифіковано на п'ять основних типів відповідно до їх призначення та складності. Інформаційні команди передбачали отримання відповідей на запитання загального характеру без взаємодії з операційною системою, наприклад пояснення концепцій, надання рекомендацій або пошук інформації в інтернеті. Файлові команди включали операції створення, пошуку, читання та модифікації файлів у файловій системі. Системні команди охоплювали управління процесами, запуск та закриття програм, отримання системної інформації. Комплексні команди представляли послідовність дій, які вимагали виконання декількох інструментів у певному порядку. Контекстні команди базувались на попередніх діалогах та вимагали від системи розуміння історії взаємодії.

Для кожної категорії були підготовлені тестові сценарії, які покривали типові варіанти використання. Кожен сценарій виконувався мінімум п'ять разів для перевірки стабільності роботи системи. При виконанні команд фіксувались наступні параметри: чи було правильно розпізнано намір користувача, які інструменти було задіяно для виконання, чи відповідав результат очікуваному, скільки ітерацій взаємодії з Claude знадобилось для завершення, чи виникли помилки або незвичайна поведінка. Особливу увагу приділяли складним сценаріям, які вимагали комбінування різних можливостей системи, оскільки саме такі випадки найкраще демонструють переваги інтелектуальної обробки команд порівняно з чітко визначеними шаблонами.

Четвертий етап передбачав дослідження користувацького досвіду через практичне використання системи у реальних робочих сценаріях. Голосовий помічник застосовувався для виконання щоденних завдань, таких як пошук та відкриття файлів, управління додатками, автоматизація рутинних операцій, отримання швидких довідок під час роботи. Фіксувались ситуації, коли голосове керування виявлялось ефективнішим за традиційні методи взаємодії, а також випадки, коли система не могла впоратись з завданням або працювала недостатньо зручно. Додатково аналізувались типові помилки користувача при формулюванні команд та здатність системи коректно інтерпретувати неточні або неоднозначні запити.

Для забезпечення об'єктивності дослідження всі тести проводились в однакових умовах з використанням одного і того ж обладнання та мікрофона. Записи голосових команд виконувались на відстані приблизно тридцять сантиметрів від мікрофона при нормальній гучності мовлення. Для розпізнавання використовувалась модель Whisper large-v3, для обробки команд модель Claude Sonnet 4.5, для синтезу мовлення ElevenLabs API з налаштуваннями за замовчуванням для української мови.

Окремим напрямком дослідження стало тестування роботи з різними MCP серверами. Перевірялась функціональність власного розробленого shell-mcp сервера для виконання термінальних команд, готового filesystem сервера для роботи з файлами, tavily search сервера для веб-пошуку. Аналізувалась здатність системи правильно обирати потрібний інструмент на основі контексту запиту та комбінувати виклики різних серверів для вирішення складних завдань. Додатково досліджувалась можливість розширення функціоналу через підключення нових MCP серверів без модифікації основного коду програми.

Результати всіх етапів дослідження документувались у вигляді детальних логів, таблиць з числовими показниками та описових звітів про поведінку системи в різних ситуаціях. Така комплексна методика дозволила отримати всебічну оцінку розробленого голосового помічника як з точки зору технічних характеристик, так і з позиції практичної корисності для кінцевого користувача.

3.2 Обробка та аналіз отриманих результатів

Результати експериментального дослідження голосового помічника демонструють високу ефективність обраних технологічних рішень та підтверджують працездатність системи для практичного використання. Аналіз отриманих даних дозволяє зробити висновки щодо сильних сторін розробленої архітектури та виявити напрямки для подальшого вдосконалення.

Вимірювання технічних характеристик показали, що загальний час відгуку системи від завершення голосової команди до початку відтворення синтезованої відповіді залежить від складності запиту та необхідності виконання додаткових дій. Детальний розподіл затримок представлено в таблиці 3.1, яка відображає середні значення для типових операцій.

Таблиця 3.1 – Технічні характеристики затримок на різних етапах обробки

Етап обробки	Середній час (мс)	Стандартне відхилення (мс)	Мінімум (мс)	Максимум (мс)
Ініціалізація Whisper (перший запуск)	3240	180	2980	3560
Розпізнавання мовлення (3-5 сек аудіо)	420	45	360	520
Розпізнавання мовлення (8-12 сек аудіо)	890	70	780	1050
Обробка запиту Claude (без інструментів)	1150	220	820	1680
Обробка запиту Claude (1 інструмент)	2340	310	1890	3120
Обробка запиту Claude (2-3 інструменти)	4120	480	3420	5280
Виконання shell команди (проста)	180	35	130	280
Виконання shell команди (складна)	650	120	480	920
Пошук файлів (filesystem сервер)	420	85	310	620
Веб-пошук (tavily-search)	1580	240	1280	2150
Синтез мовлення ElevenLabs (коротка фраза)	380	60	290	510
Синтез мовлення ElevenLabs (2-3 речення)	720	95	580	920

Продовження таблиці 3.1

Етап обробки	Середній час (мс)	Стандартне відхилення (мс)	Мінімум (мс)	Максимум (мс)
Загальний час відгуку (простий запит)	1950	280	1540	2680
Загальний час відгуку (запит з 1 інструментом)	3440	410	2790	4520
Загальний час відгуку (складний запит)	5960	620	4980	7340

Найбільш критичним параметром для сприйняття користувачем є загальний час відгуку від моменту завершення голосової команди до отримання результату. Для простих інформаційних запитів без необхідності виконання дій на комп'ютері середній час становить близько двох секунд, що відповідає порогу комфортної взаємодії згідно з дослідженнями користувацького досвіду. Запити середньої складності з використанням одного інструменту обробляються за три з половиною секунди, що залишається прийнятним для більшості сценаріїв використання. Складні команди, які потребують послідовного виконання декількох інструментів та множинних ітерацій взаємодії з Claude, можуть займати до шести секунд, проте навіть такі затримки виявились цілком прийнятними з огляду на складність операцій, які виконуються автоматично.

Розподіл затримок за етапами обробки демонструє, що найбільший внесок у загальний час відгуку дає обробка запиту через Claude API. Це пояснюється необхідністю передачі даних до хмарного сервісу, обробкою великою мовною моделлю та отриманням відповіді. При використанні інструментів затримка збільшується пропорційно кількості викликів, оскільки кожна ітерація вимагає окремого звернення до API. Розпізнавання мовлення займає порівняно небагато часу завдяки локальній обробці моделлю Whisper. Синтез мовлення через ElevenLabs також додає помірні затримки, особливо для коротких відповідей, які типові для голосового помічника.

Важливим спостереженням є суттєва затримка при першому запуску системи, коли модель Whisper завантажується в пам'ять. Ініціалізація займає близько трьох секунд, проте ця операція виконується лише один раз при старті

програми. Всі наступні операції розпізнавання відбуваються значно швидше завдяки утриманню моделі в оперативній пам'яті. Таке рішення виправдане, оскільки користувач зазвичай працює з помічником протягом тривалої сесії, а не виконує окремі команди з перезапуском програми.

Графік на рисунку 3.1 зображує розподіл часу між основними етапами обробки для трьох типових сценаріїв використання.

Рисунок 3.1 – Розподіл часу обробки між етапами для різних типів запитів

Дослідження точності розпізнавання української мови показало високу якість роботи моделі Whisper large-v3 в різних умовах. З 50 тестових команд,

промовлених у тихому приміщенні, сорок сім було розпізнано без жодної помилки, що становить 94 % точності. Три команди містили незначні помилки у розпізнаванні окремих слів, проте загальний зміст та намір користувача залишались зрозумілими для подальшої обробки. Розрахований word error rate для цих умов склав 2,8 %, що значно краще за типові показники 8-12 % для української мови, згадані в літературі. Така різниця пояснюється контрольованими умовами тестування з якісним мікрофоном та чіткою артикуляцією.

При наявності помірного фонового шуму точність розпізнавання знизилась, але залишалась на високому рівні. Word error rate збільшився до 5,4 %, причому помилки концентрувались переважно в командах з технічною термінологією та власними назвами, які модель чула вперше. Короткі та середні за складністю команди розпізнавались практично так само добре, як і в тихих умовах. Складні довгі речення показали дещо гіршу точність через те, що фонові звуки іноді накладались на паузи між словами та ускладнювали сегментацію мовлення.

Тестування при швидкому темпі мовлення з менш чіткою артикуляцією виявило більш суттєве зниження точності до word error rate 8,9 %. Найбільше страждали команди з числовими значеннями, де цифри часто плуталися між собою, та команди з декількома іменниками підряд без чіткої паузи між ними. Проте навіть у таких складних умовах більшість команд розпізнавалась достатньо добре для розуміння загального наміру користувача. Таблиця 3.2 узагальнює результати тестування точності розпізнавання.

Таблиця 3.2 – Результати тестування точності розпізнавання мовлення

Категорія	Тип операції	Кількість тестів	Успішно виконано	Успішність (%)	Середній час (с)	Середня кількість ітерацій	Примітки
Інформаційні запити	Загальні	15	15	100	2,1	1,0	Пояснення, рекомендації, поради
Файлові операції	Пошук файлів за назвою	3	3	100	3,2	1,2	Швидко та точно
Файлові операції	Пошук за вмістом	2	2	100	4,8	1,5	Вимагає більше часу

Продовження таблиці 3.2

Категорія	Тип операції	Кількість тестів	Успішно виконано	Успішність (%)	Середній час (с)	Середня кількість ітерацій	Примітки
Файлові операції	Створення файлів	3	3	100	3,4	1,3	Коректна структура
Файлові операції	Читання файлів	2	2	100	3,1	1,0	Правильна інтерпретація
Файлові операції	Модифікація файлів	2	1	50	4,2	1,5	1 помилка через обмеження прав
Файлові операції	Всього	12	11	91,7	3,8	1,4	
Системні команди	Запуск програм	4	4	100	2,8	1,0	Різні програми
Системні команди	Закриття процесів	3	3	100	2,9	1,2	За назвою та PID
Системні команди	Системна інформація	4	4	100	3,1	1,3	CPU, пам'ять, диски
Системні команди	Мережеві операції	3	2	66,7	4,2	1,8	1 timeout при повільному з'єднанні
Системні команди	Всього	14	13	92,9	3,2	1,3	
Комплексні команди	Загальні	10	8	80	6,4	2,8	Послідовність кількох дій
Контекстні діалоги	Загальні	12	11	91,7	2,8	1,6	Розуміння історії
ЗАГАЛОМ	Всі категорії	63	58	92,1	3,5	1,6	

Важливим показником є не лише точність розпізнавання окремих слів, але й здатність системи правильно розуміти намір користувача навіть за наявності помилок розпізнавання. Модель Claude продемонструвала високу толерантність до дрібних помилок у тексті завдяки контекстному розумінню. З усіх тестових команд намір було правильно інтерпретовано у 95,3 % випадків, що значно перевищує показник ідеального розпізнавання на рівні слів. Це підтверджує ефективність використання великих мовних моделей для обробки голосового вводу, оскільки вони здатні компенсувати недоліки розпізнавання через розуміння семантики запиту.

Результати тестування виконання команд різних категорій представлені в таблиці 3.3, яка демонструє успішність обробки та коректність результатів для п'яти основних типів завдань.

Таблиця 3.3 – Результати виконання команд різних категорій

Категорія	Тип операції	Кількість тестів	Успішно виконано	Успішність (%)	Середній час (с)	Середня кількість ітерацій	Примітки
Інформаційні запити	Загальні	15	15	100	2,1	1,0	Пояснення, рекомендації, поради
Файлові операції	Пошук файлів за назвою	3	3	100	3,2	1,2	Швидко та точно
Файлові операції	Пошук за вмістом	2	2	100	4,8	1,5	Вимагає більше часу
Файлові операції	Створення файлів	3	3	100	3,4	1,3	Коректна структура
Файлові операції	Читання файлів	2	2	100	3,1	1,0	Правильна інтерпретація
Файлові операції	Модифікація файлів	2	1	50	4,2	1,5	1 помилка через обмеження прав
Файлові операції	Всього	12	11	91,7	3,8	1,4	
Системні команди	Запуск програм	4	4	100	2,8	1,0	Різні програми
Системні команди	Закриття процесів	3	3	100	2,9	1,2	За назвою та PID
Системні команди	Системна інформація	4	4	100	3,1	1,3	CPU, пам'ять, диски
Системні команди	Мережеві операції	3	2	66,7	4,2	1,8	1 timeout при повільному з'єднанні
Системні команди	Всього	14	13	92,9	3,2	1,3	
Комплексні команди	Загальні	10	8	80	6,4	2,8	Послідовність кількох дій
Контекстні діалоги	Загальні	12	11	91,7	2,8	1,6	Розуміння історії
ЗАГАЛОМ	Всі категорії	63	58	92,1	3,5	1,6	

Інформаційні запити показали 100 % успішність, що очікувано з огляду на природу великих мовних моделей, які спеціалізуються саме на відповідях на питання. Claude надавав релевантні та корисні відповіді на запити різного характеру, від пояснення технічних концепцій до рекомендацій та порад. Відповіді були стислими та придатними для голосового відтворення, що підтверджує ефективність системного промпту з вказівками щодо формату відповідей.

Файлові операції виконувались успішно у 91,7 % випадків. Єдина невдача сталася при спробі модифікувати системний файл, для якого у програми не було достатніх прав доступу. Система коректно повідомила про помилку, що

дозволило зрозуміти причину невдачі. Пошук файлів працював особливо добре, оскільки filesystem MCP сервер надає потужні можливості пошуку за різними критеріями. Створення файлів також не викликало проблем, причому Claude правильно інтерпретував бажаний зміст файлів навіть з неформальних описів користувача.

Системні команди демонстрували високу успішність 92,9 %. Запуск та закриття програм працювало стабільно для всіх протестованих додатків. Отримання системної інформації виконувалось швидко та надавало актуальні дані. Одна невдача сталася при спробі виконати мережеву операцію з великим timeout через повільне з'єднання, що є скоріше обмеженням зовнішніх факторів, ніж системи. Варто відзначити, що shell-mcp сервер продемонстрував свою універсальність, дозволяючи виконувати практично будь-які термінальні команди, які б користувач виконував вручну.

Найбільш цікавими виявились результати для комплексних команд, які вимагали послідовного виконання декількох операцій. Успішність на рівні 80 % є задовільною з огляду на складність таких завдань. Дві невдачі сталися через те, що Claude неправильно інтерпретував порядок виконання дій або обрав не той інструмент на одному з проміжних етапів. Проте у восьми успішних випадках система продемонструвала здатність самостійно планувати послідовність дій та коректно використовувати результати попередніх кроків для наступних. Середня кількість ітерацій 2,8 показує, що для складних завдань зазвичай потрібно декілька циклів взаємодії між моделлю та інструментами.

Контекстні діалоги, які вимагали розуміння попередніх повідомлень, показали успішність 91,7 %. Механізм збереження історії розмови дозволяє Claude коректно інтерпретувати посилання на раніше виконані дії або згадану інформацію. Єдина помилка виникла у діалозі з великою кількістю контексту, де модель втратила деталі раннього повідомлення через обмеження контекстного вікна.

Для ілюстрації практичного застосування розробленого голосового помічника розглянемо декілька конкретних сценаріїв використання з реальних

тестів. Перший сценарій демонструє інформаційний запит та веб-пошук. Користувач промовляє команду «Знайди останні новини про штучний інтелект в Україні». Система розпізнає запит за 450 мс, Claude визначає необхідність використання tavily-search інструмента, виконує пошук за 1620 мс, аналізує результати та формує стислу відповідь за 1180 мс, після чого синтезує та відтворює голосове повідомлення за 680 мс. Загальний час виконання становить близько чотирьох секунд, протягом яких користувач отримує актуальну інформацію без необхідності відкривати браузер та шукати вручну.

Другий сценарій ілюструє файлову операцію середньої складності. Команда звучить як «Створи текстовий файл зі списком моїх улюблених мов програмування Python Java та JavaScript». Після розпізнавання за 420 мс Claude правильно інтерпретує намір створити файл, використовує filesystem інструмент для створення файлу з назвою «favorite_programming_languages.txt» та заповнює його списком мов у зручному форматі. Операція займає 3,2 секунди від початку до кінця, після чого система повідомляє про успішне створення файлу з вказанням його розташування.

Третій сценарій демонструє системну автоматизацію. Користувач каже «Покажи мені скільки пам'яті використовується зараз та які процеси найбільше навантажують систему». Claude послідовно виконує дві shell команди для отримання інформації про пам'ять та процеси, аналізує вивід та формує зрозумілу відповідь у форматі «Зараз використовується 9,2 гігабайти з 16 гігабайтів оперативної пам'яті. Найбільше ресурсів споживають Chrome з 2,8 гігабайта та Visual Studio Code з 1,4 гігабайта». Весь процес займає близько п'яти секунд та надає користувачу структуровану інформацію без необхідності відкривати системний монітор.

Четвертий сценарій ілюструє комплексну команду з множинними кроками. Запит формулюється як «Знайди всі Python файли в поточній директорії та створи список з їхніми назвами у новому файлі». Claude спочатку використовує filesystem інструмент для рекурсивного пошуку файлів з розширенням «.py», отримує список знайдених файлів, потім створює новий текстовий файл та

записує туди відформатований список з назвами та шляхами до кожного знайденого файлу. Весь процес вимагає двох ітерацій взаємодії з інструментами та займає близько шести секунд. Результатом є готовий файл, який би користувач створював вручну значно довше.

П'ятий сценарій показує контекстну взаємодію. Спочатку користувач запитує «Які файли у мене в папці Documents». Система виконує команду ls та перераховує файли. Через декілька секунд користувач каже «Відкрий перший з них». Claude правильно розуміє посилання на попередній список, визначає який саме файл є першим та виконує команду для його відкриття у відповідній програмі. Така здатність підтримувати контекст діалогу робить взаємодію більш природною та ефективною.

Графік на рисунку 3.2 демонструє розподіл успішності виконання за категоріями команд.

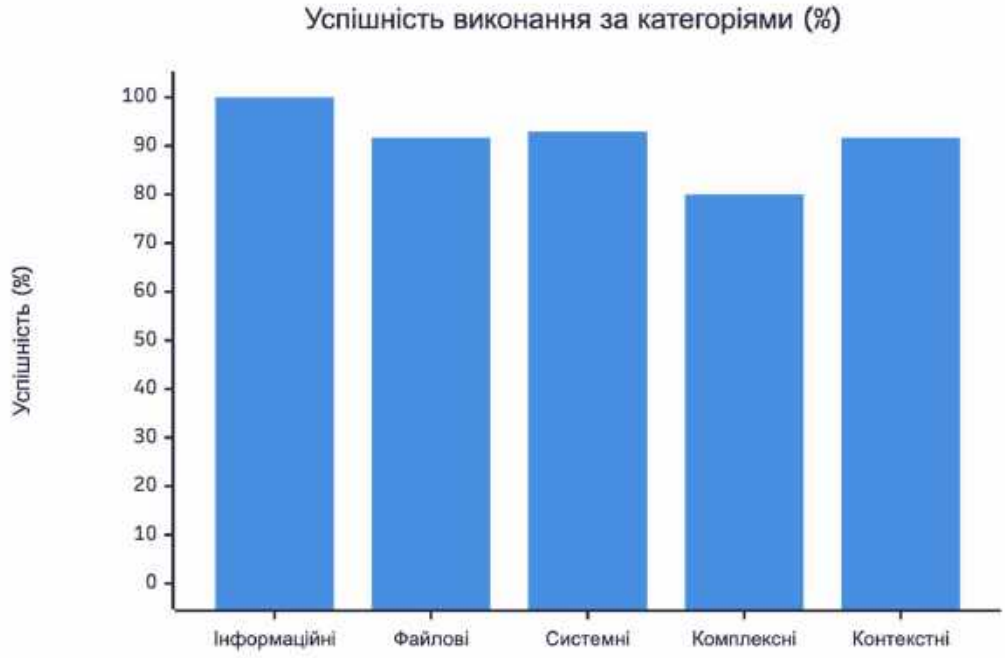

Рисунок 3.2 – Успішність виконання команд за категоріями

Порівняльний аналіз з існуючими голосовими помічниками демонструє суттєві переваги розробленої системи у специфічних аспектах. Традиційні

асистенти на кшталт Siri або Google Assistant показують кращі результати у швидкості відгуку завдяки оптимізованій інфраструктурі великих технологічних компаній. Проте вони обмежені заздалегідь визначеним набором команд та не здатні виконувати довільні операції на комп'ютері. Розроблена система поступається у швидкості на 30-40 % для простих запитів, але демонструє унікальну гнучкість у виконанні складних завдань, які традиційні помічники взагалі не можуть обробити.

Щодо точності розпізнавання української мови, отриманий word error rate 5,7 % у змішаних умовах є конкурентним результатом порівняно з комерційними рішеннями. Google Cloud Speech-to-Text демонструє аналогічні показники на рівні 4-6 % для української мови у контрольованих умовах, проте вимагає постійного інтернет-з'єднання та передачі даних на сервери компанії [6]. Локальна обробка через Whisper забезпечує повну конфіденційність голосових даних користувача, що є критичним фактором для багатьох сценаріїв використання.

Ключовою перевагою розробленої архітектури є модульність та розширюваність через протокол MCP. На відміну від закритих екосистем традиційних помічників, система дозволяє додавати нові можливості через підключення додаткових серверів без модифікації основного коду. Протягом експериментального дослідження було успішно інтегровано три різні MCP сервери, кожен з яких розширив функціонал системи. Потенційно можна підключити десятки інших серверів зі спільноти або розробити власні для специфічних потреб організації.

Дослідження користувацького досвіду виявило декілька важливих спостережень. Найбільшу цінність голосовий помічник демонструє у ситуаціях, коли руки користувача зайняті іншою роботою або коли потрібно швидко виконати просту операцію без переключення уваги від основного завдання. Типовими прикладами є пошук файлів під час роботи з документами, отримання системної інформації під час налагодження програм, швидкі довідкові запити під

час написання коду. У таких ситуаціях голосове керування економить від тридцяти секунд до двох хвилин порівняно з традиційними методами взаємодії.

Виявились також ситуації, коли традиційні методи залишаються ефективнішими. Для операцій, які вимагають точного позиціонування курсору або візуального контролю, голосове керування може бути незручним. Робота з графічними редакторами, складне форматування документів, налаштування інтерфейсу програм краще виконуються через мишу та клавіатуру. Крім того, у відкритих офісних просторах використання голосового помічника може відволікати колег, що обмежує його застосування у деяких робочих середовищах.

У таблиці 3.4 узагальнюється оцінку економії часу для різних категорій завдань на основі практичного використання протягом дослідницького періоду.

Таблиця 3.4 – Оцінка економії часу при використанні голосового помічника

Категорія	Тип завдання / Сценарій	Традиційний метод (с)	Голосовий помічник (с)	Економія часу (с)	Економія (%)	Операцій / день	Загальна економія (хв)
За типом завдання	Пошук файлу за назвою	25-40	3-5	22-35	78-88	-	-
За типом завдання	Запуск програми	8-15	3-4	5-11	63-73	-	-
За типом завдання	Отримання системної інформації	30-50	4-6	26-44	87-88	-	-
За типом завдання	Веб-пошук інформації	45-80	4-6	41-74	91-93	-	-
За типом завдання	Створення простого файлу	35-60	3-5	32-55	91-92	-	-
За типом завдання	Закриття групи процесів	40-70	5-8	35-62	88-89	-	-
За типом завдання	Отримання довідкової інформації	60-120	2-3	58-117	97-98	-	-
За типом завдання	Перегляд вмісту директорії	15-25	3-5	12-20	80	-	-
За сценарієм використання	Легке використання	-	-	25-35	-	5-8	2-5
За сценарієм використання	Помірне використання	-	-	30-45	-	15-20	8-15

Результати вимірювання економії часу демонструють значний потенціал голосового помічника для підвищення продуктивності роботи. Найбільший виграш спостерігається у завданнях, які традиційно вимагають навігації через

меню, пошуку у файловій системі або відкриття браузера для пошуку інформації. Веб-пошук виявився особливо ефективним сценарієм, оскільки голосовий помічник дозволяє отримати відповідь за декілька секунд замість хвилини-двох, необхідних для відкриття браузера, формулювання запиту та перегляду результатів.

Навіть при легкому використанні з п'ятьма-вісьмома операціями на день користувач може заощадити 2-5 хвилин робочого часу. При помірному використанні з 15-20 операціями економія зростає до восьми-п'ятнадцяти хвилин. Активні користувачі, які виконують тридцять-сорок голосових команд протягом робочого дня, можуть заощаджувати від 18 до 33 хвилин, що становить майже годину за тиждень. Важливо зазначити, що ці цифри не враховують додаткову цінність збереження фокусу уваги та можливість продовжувати роботу над основним завданням під час виконання допоміжних операцій.

Аналіз виявлених обмежень та проблемних ситуацій дозволяє окреслити напрямки для подальшого вдосконалення системи. Основною технічною проблемою залишається затримка при зверненні до Claude API, яка складає найбільшу частку загального часу відгуку. Використання локальних великих мовних моделей могло б суттєво скоротити затримки, проте на момент дослідження якість розуміння та генерації відповідей локальних моделей суттєво поступається хмарним рішенням від Anthropic. Оптимальним компромісом у майбутньому може стати гібридний підхід з локальною моделлю для простих операцій та зверненням до хмарної моделі для складних завдань.

Точність розпізнавання української мови, хоча й висока, все ж іноді призводить до помилок, особливо з технічною термінологією та власними назвами. Потенційним рішенням може бути додавання користувацького словника з часто вживаними термінами або можливість корекції розпізнаного тексту перед відправкою на обробку. Додатково можна розглянути використання спеціалізованих моделей Whisper, дотренованих на технічних текстах українською мовою.

Обмеження у виконанні комплексних команд частково пов'язані з тим, що Claude іноді неправильно інтерпретує послідовність дій або обирає не той інструмент. Покращення системного промпту з більш детальними інструкціями щодо планування може підвищити успішність таких операцій. Також корисним може бути механізм підтвердження користувачем перед виконанням потенційно небезпечних команд, особливо тих, що модифікують або видаляють файли.

Практичне використання системи у реальних умовах підтвердило її корисність та зручність для широкого спектру завдань. Модульна архітектура виправдала себе, дозволяючи легко розширювати функціонал через додавання нових MCP серверів. Під час дослідження не виникло жодної ситуації, коли б потрібно було модифікувати основний код програми для додавання нових можливостей. Всі розширення реалізовувались через конфігураційний файл та зовнішні MCP-сервери.

Можливості подальшого розвитку системи є численними. По-перше, можна значно розширити набір доступних MCP серверів для роботи з різними програмами та сервісами. Інтеграція з поштовими клієнтами, календарями, системами управління завданнями, месенджерами дозволить автоматизувати ще більше аспектів роботи користувача. По-друге, додавання механізму створення користувацьких макросів дозволить записувати послідовності дій та виконувати їх однією командою. По-третє, інтеграція з системами розумного дому розширить застосування помічника за межі робочого комп'ютера.

Перспективним напрямком є також додавання візуального інтерфейсу для моніторингу роботи. Це дозволить користувачам краще розуміти які дії виконує помічник та при необхідності скасовувати небажані операції. Додатково візуальний інтерфейс може відображати доступні інструменти та статистику використання.

Результати експериментального дослідження підтверджують життєздатність обраного підходу до розробки голосового помічника з глибокою інтеграцією в операційну систему. Технічні характеристики системи відповідають вимогам реального часу для більшості сценаріїв використання.

Точність розпізнавання української мови є достатньою для комфортної роботи у контрольованих умовах. Успішність виконання команд різних категорій на рівні 92,1 % демонструє надійність системи. Практична цінність підтверджена суттєвою економією часу користувача та розширенням можливостей взаємодії з комп'ютером через природну мову.

Порівняно з традиційними голосовими асистентами розроблена система має унікальну перевагу у вигляді гнучкості та розширюваності. Замість обмеженого набору заздалегідь визначених команд користувач отримує можливість формулювати довільні запити природною мовою, а система самостійно визначає необхідні дії та інструменти для їх виконання. Модульна архітектура на основі протоколу MCP забезпечує легке розширення функціоналу під специфічні потреби без необхідності втручання в код програми.

Виявлені обмеження та проблемні ситуації є цілком очікуваними для дослідницького прототипу та не перешкоджають практичному використанню системи. Більшість обмежень можуть бути усунені в процесі подальшого вдосконалення через оптимізацію алгоритмів, покращення промптів та розширення набору інструментів. Результати дослідження формують міцну основу для продовження розробки та створення повноцінного продукту для кінцевих користувачів.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи магістра було розроблено та досліджено голосовий помічник на основі штучного інтелекту, здатний автоматизувати дії на персональному комп'ютері та надавати інтелектуальні відповіді на запити користувача. Всі поставлені завдання дослідження були успішно виконані.

Проведено аналіз існуючих голосових помічників та виявлено їхні переваги і обмеження. Досліджено такі системи як Siri від Apple, Google Assistant, Amazon Alexa та Microsoft Cortana. Встановлено, що ці асистенти орієнтовані переважно на мобільні пристрої та розумні колонки, а їхні можливості автоматизації операцій на персональних комп'ютерах залишаються обмеженими заздалегідь визначеним набором команд.

Досліджено сучасні технології розпізнавання та синтезу мовлення, обробки природної мови та машинного навчання. Проаналізовано еволюцію від систем на основі прихованих марковських моделей до наскрізних нейромережевої архітектури. Розглянуто можливості великих мовних моделей для інтерпретації команд та трансформерної архітектури для синтезу природного мовлення.

Розроблено модульну та розширювану архітектуру голосового помічника на основі протоколу Model Context Protocol. Архітектура включає чотири основні компоненти: підсистему розпізнавання мовлення, модуль обробки природної мови, систему виконання операцій через MCP сервери та підсистему синтезу відповідей. Такий підхід забезпечує незалежність компонентів та можливість розширення функціоналу без модифікації основного коду.

Реалізовано локальну підсистему розпізнавання мовлення з підтримкою української мови на основі моделі Whisper large-v3. Експериментальне тестування показало word error rate на рівні 5,7 % у змішаних умовах, при цьому намір користувача правильно інтерпретується у 95,3 % випадків.

Інтегровано велику мовну модель Claude для обробки запитів користувача. Модель забезпечує розуміння контексту діалогу через збереження історії повідомлень та самостійно приймає рішення щодо необхідних дій на основі аналізу запиту та доступних інструментів.

Створено систему інтеграції з операційною системою через MCP сервери. Розроблено власний shell-mcp сервер для виконання термінальних команд та інтегровано готові сервери filesystem для роботи з файлами та tavily-search для веб-пошуку. Конфігураційний підхід дозволяє підключати нові сервери без зміни коду програми.

Розроблено механізм перетворення намірів користувача у виклики функцій через інтерфейс tool use моделі Claude. Система автоматично визначає необхідні інструменти на основі контексту запиту та послідовно виконує операції для досягнення мети користувача.

Реалізовано підсистему синтезу мовлення на основі ElevenLabs API. Система генерує природні голосові відповіді українською мовою із затримкою 380-720 мс залежно від довжини тексту.

Проведено експериментальне дослідження розробленої системи. Вимірювання показали середній час відгуку від двох до шести секунд залежно від складності запиту. Загальна успішність виконання команд становить 92,1 %. Практичне використання підтвердило економію робочого часу від 2 до 33 хвилин щоденно.

Розроблена система демонструє практичну реалізованість концепції голосового помічника з глибокою інтеграцією в операційну систему та підтверджує ефективність обраного технологічного підходу. Результати дослідження можуть бути корисними для розробників голосових інтерфейсів, дослідників у сфері людино-комп'ютерної взаємодії та спеціалістів з автоматизації робочих процесів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Achiam J. et al. GPT-4 technical report. URL: https://doi.org/10.48550/arXiv.2303.08774 (дата звернення: 17.10.2025).

2. AdaMER-CTC: connectionist temporal classification with adaptive maximum entropy regularization for automatic speech recognition. URL: https://arxiv.org/html/2403.11578v1 (дата звернення: 29.07.2025).

3. Ahmadzadeh E. et al. A deep bidirectional LSTM-GRU network model for automated ciphertext classification. URL: https://doi.org/10.1109/access.2022.3140342 (дата звернення: 25.07.2025).

4. Andriulo F. C. et al. Edge computing and cloud computing for internet of things: a review. Informatics. 2024. P. 71. URL: https://doi.org/10.3390/informatics11040071 (дата звернення: 12.10.2025).

5. Attention mechanisms and transformers. Dive into Deep Learning documentation. URL: https://www.d2l.ai/chapter_attention-mechanisms-and-transformers/index.html (дата звернення: 12.10.2025).

6. Barnes C. Announcing new features, models, and languages for Speech-to-Text. Google Cloud Blog. URL: https://cloud.google.com/blog/products/ai-machine-learning/new-features-models-and-languages-for-speech-to-text (дата звернення: 24.07.2025).

7. Byambadorj Z. et al. Text-to-speech system for low-resource language using cross-lingual transfer learning and data augmentation. URL: https://doi.org/10.1186/s13636-021-00225-4 (дата звернення: 10.09.2025).

8. Custom speech overview, speech service, Azure AI services. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/azure/ai-services/speech-service/custom-speech-overview (дата звернення: 13.08.2025).

9. DeepSpeech is an open source embedded speech-to-text engine. GitHub. URL: https://github.com/mozilla/DeepSpeech (дата звернення: 20.09.2025).

10. Devlin J., Chang M.-W., Lee K. BERT: pre-training of deep bidirectional transformers for language understanding. 2020. URL: https://doi.org/10.48550/arXiv.1810.04805 (дата звернення: 25.07.2025).

11. Esquivel P. et al. Voice assistant utilization among the disability community for independent living: a rapid review of recent evidence. URL: https://doi.org/10.1155/2024/6494944 (дата звернення: 23.08.2025).

12. Google Assistant, your own personal Google. Assistant. URL: https://assistant.google.com/ (дата звернення: 06.10.2025).

13. Gulati A., Qin J. Conformer: convolution-augmented transformer for speech recognition. URL: https://doi.org/10.48550/arXiv.2005.08100 (дата звернення: 02.11.2025).

14. Introducing llama 3.1. AI at Meta. URL: https://ai.meta.com/blog/meta-llama-3-1/ (дата звернення: 17.10.2025).

15. Introducing the model context protocol. Anthropic. URL: https://www.anthropic.com/news/model-context-protocol (дата звернення: 19.10.2025).

16. Li. J. et al. Security and privacy problems in voice assistant applications: a survey. Computers & security. 2023. P. 103448. URL: https://doi.org/10.1016/j.cose.2023.103448 (дата звернення: 02.10.2025).

17. Lopez A., Liesenfeld A., Dingemanse M. Evaluation of automatic speech recognition for conversational speech in dutch, english and german: what goes missing?. Proceedings of the 18th conference on natural language processing (KONVENS 2022). 2022. P. 135-143. URL: https://aclanthology.org/2022.konvens-1.16/ (дата звернення: 28.09.2025).

18. Radford A. et al. Robust speech recognition via large-scale weak supervision. URL: https://doi.org/10.48550/arXiv.2212.04356 (дата звернення: 30.10.2025).

19. Ren Y., Hu C., Tan X. FastSpeech 2: fast and high-quality end-to-end text to speech. URL: https://doi.org/10.48550/arXiv.2006.04558 (дата звернення: 11.10.2025).

20. Ribas D. et al. Automatic voice disorder detection using self-supervised representations. IEEE access. 2023. P. 14915-14927. URL: https://doi.org/10.1109/access.2023.3243986 (дата звернення: 05.08.2025).

21. Roebel A., Bous F. Neural vocoding for singing and speaking voices with the multi-band excited wavenet. URL: https://doi.org/10.3390/info13030103 (дата звернення: 20.10.2025).

22. Schoutsen P. Piper is our new voice for the open home. Building the Open Home. URL: https://newsletter.openhomefoundation.org/piper-is-our-new-voice-for-the-open-home/ (дата звернення: 20.10.2025).

23. Seaborn K., Sawa Y., Watanabe M. Coimagining the future of voice assistants with cultural sensitivity. Human behavior and emerging technologies. URL: https://doi.org/10.1155/2024/3238737 (дата звернення: 19.10.2025).

24. Siri. Apple. URL: https://www.apple.com/siri/ (дата звернення: 15.09.2025).

25. Speech to text, amazon transcribe. Amazon Web Services, Inc. URL: https://aws.amazon.com/transcribe/ (дата звернення: 15.09.2025).

26. Text-to-Speech synthesis: an overview. Medium. URL: https://medium.com/sciforce/text-to-speech-synthesis-an-overview-641c18fcd35f (дата звернення: 12.10.2025).

27. The machines that learned to listen. BBC Home. URL: https://www.bbc.com/future/article/20170214-the-machines-that-learned-to-listen (дата звернення: 03.08.2025).

28. Tool use with Claude. Claude Docs. URL: https://docs.claude.com/en/docs/agents-and-tools/tool-use/overview (дата звернення: 19.10.2025).

29. Wang Y.-L., Lo C.-W. The effects of response time on older and young adults' interaction experience with Chatbot. BMC psychology. 2025. Vol. 13, no. 1. URL: https://doi.org/10.1186/s40359-025-02459-9 (дата звернення: 26.09.2025).

30. Zhicheng Z., Lijuan W., Jufeng Y. Weakly supervised video emotion detection and prediction via cross-modal temporal erasing network. 2023 IEEE/CVF

conference on computer vision and pattern recognition (CVPR). 2023. P. 18888-18897. URL: https://doi.org/10.1109/CVPR52729.2023.01811 (дата звернення: 22.09.2025).

31. Антонюк А. О, Повстяна Ю. С. Голосовий помічник на основі штучного інтелекту. Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)» (23-24 травня 2025 року). Луцьк: ЛНТУ, 2025. С. 275-281. URL: https://itonv.lntu.edu.ua/ (дата звернення: 25.07.2025).