Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження та розробка системи на базі
великої мовної моделі для розважальних цілей
(тема)

Виконав:
здобувач другого року навчання,
групи СШІм-23-1
Максим Небаба
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва освітньої програми)
Керівник проф. Наталія Рябова
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Небабі Максиму Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та розробка системи на базі великої мовної моделі для розважальних цілей

затверджена наказом університету від 21 квітня 20 25 р. № 295Ст

2. Термін подання студентом роботи до екзаменаційної комісії 5 червня 20 25 р.

3. Вихідні дані до роботи науково-технічні публікації та статті, дані інтернет джерел за тематикою дослідження; проекти зі створення великих мовних моделей; застосунок для текстової та голосової взаємодії між користувачами Discord; Python.

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задач дослідження
2) Аналіз та вибір програмних засобів і мови реалізації системи
3) Реалізація системи голосової взаємодії

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	21.04.2025	виконано
2	Аналіз предметної галузі	21.04.2025 – 28.04.2025	виконано
3	Аналіз існуючих рішень	28.04.2025 – 03.05.2025	виконано
4	Постановка задачі	03.05.2025 – 04.05.2025	виконано
5	Порівняльний аналіз існуючих засобів для	04.05.2025 – 12.05.2025	виконано
	реалізації системи		
6	Реалізація прототипу системи	12.05.2025 – 21.05.2025	виконано
7	Огляд недоліків та методів покращення системи	21.05.2025 – 22.05.2025	виконано
8	Написання пояснювальної записки	22.05.2025 – 29.05.2025	виконано
9	Попередній захист	02.06.2024	виконано
10	Захист перед ЕК	05.06.2024	

Дата видачі завдання 21 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Наталія Рябова
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 70 с., 12 рис., 1 дод., 21 джерело.

БОТ, ВЕЛИКІ МОВНІ МОДЕЛІ, ОБРОБКА ПРИРОДНОЇ МОВИ, СИНТЕЗ МОВЛЕННЯ, ШТУЧНИЙ ІНТЕЛЕКТ.

Об’єкт дослідження – процеси мовної взаємодії людини з комп’ютерною системою в режимі реального часу за допомогою голосових технологій.

Предмет дослідження – архітектурні, програмні та інтелектуальні компоненти інтерактивної голосової системи на базі великих мовних моделей, що функціонує у середовищі Discord.

Мета роботи – розробити прототип інтерактивної голосової системи на основі технологій штучного інтелекту, здатної реалізовувати повноцінний цикл спілкування з користувачем у голосовому форматі: від розпізнавання мовлення до генерації та озвучення відповіді з мінімальною затримкою.

Методи дослідження – у роботі використано методи архітектурного проєктування програмних систем, модульної розробки на Python, аналізу й інформації. Застосовано мовні моделі як хмарного, так і локального типу, модель розпізнавання мовлення Whisper та синтезатор Kokoro TTS із підтримкою зміни голосу. Взаємодія між компонентами реалізована через високопродуктивний GRPC-сервер для забезпечення низької затримки і масштабованості.

Система здатна адаптуватися до різних стилів спілкування, підтримує багатокористувацьку комунікацію та є придатною для використання у таких сферах, як створення віртуальних стрімерів, освітніх платформ із голосовими асистентами, інтерактивних ігор, сервісів підтримки та цифрових компаньйонів.

ABSTRACT

Master’s thesis contains: 70 p., 12 fig., 1 ann., 21 sources.

ARTIFICIAL INTELLIGENCE, BOT, LARGE LANGUAGE MODELS, NATURAL LANGUAGE PROCESSING, SPEECH SYNTHESIS.

Object of the research – the processes of human-computer linguistic interaction in real-time using voice technologies.

Subject of the research – architectural, software, and intelligent components of an interactive voice system based on large language models operating within the Discord platform.

Purpose of the research – to develop a prototype of an interactive voice system based on artificial intelligence technologies, capable of executing a complete cycle of voice communication with the user: from speech recognition to response generation and voice synthesis with minimal latency.

Research methods – the study applies methods of software system architectural design, modular development in Python, and information flow analysis. Both cloud-based and locally deployed language models were used, along with the Whisper speech recognition model and the Kokoro TTS synthesizer with voice-changing support. Component interaction is implemented via a high-performance gRPC server to ensure low latency and scalability.

The system can adapt to different communication styles, supports multi-user interaction, and is suitable for applications such as virtual streamers, educational platforms with voice assistants, interactive games, support services, and digital companions.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 8
Вступ 9
1 Аналіз предметної галузі та постановка задач дослідження 10
1.1 Аналіз предметної галузі 10
1.2 Аналіз існуючих рішень 11
1.3 Генеративний штучний інтелект та великі мовні моделі як основа інтерактивної системи 12
1.3.1 Роль LLM у побудові генеративних систем штучного інтелекту 14
1.3.2 Архітектура LLM: трансформер та його принципи 15
1.4 Постановка задачі 18
1.4.1 Визначення критичних аспектів функціонування системи в режимі реального часу 19
1.4.2 Відповідність стилю та особистості персонажу 21
1.4.3 Як оптимально використовувати промпти 22
1.4.4 Дані для fine-tuning моделі 24
1.4.5 Функціональні особливості та технічна реалізація 25
1.4.6 Короткострокова та довгострокова пам’ять 27
1.4.7 Що таке Discord 29
1.4.8 Запис аудіо з Discord-каналу 30
1.4.9 Перетворення аудіо в текст (Speech-to-Text) 32
1.4.10 Визначення моменту відповіді 32
1.4.11 Перетворення тексту у голос 34
2 Аналіз та вибір програмних засобів і мови реалізації системи 37
2.1 Структура системи та визначення її ключових функціональних компонентів 37
2.2 Огляд і порівняння великих мовних моделей 38
2.3 Вибір засобів інтеграції з Discord API 42

2.4 Порівняння класичних та сучасних методів синтезу мовлення 44
2.5 Розпізнавання мовлення в режимі реального часу 46
2.6 Підходи до реалізації серверної частини системи 47
3 Реалізація системи голосової взаємодії 50
3.1 Розробка Discord-бота як інтерфейсу користувача 50
3.1.1 Ініціалізація та конфігурація Discord-бота 50
3.1.2 Підключення бота до голосових каналів Discord 51
3.1.3 Запис голосових даних: реалізація механізму обробки аудіо ... 52
3.1.4 Відтворення аудіо у голосовому каналі Discord 53
3.1.5 Перехід від discord.py до pycord: обґрунтування та реалізація . 53
3.2 Побудова та налаштування GRPC-сервера для обробки даних 55
3.3 Інтеграція Whisper для розпізнавання мовлення 58
3.4 Використання мовної моделі OpenAI для генерації відповідей 59
3.5 Синтез мовлення за допомогою Kokoro TTS 60
3.6 Загальні результати реалізації програмної системи 62
3.7 Можливості покращення та подальшого розвитку системи 63
Висновки 67
Перелік джерел посилання 68
Додаток А Відомість кваліфікаційної роботи 70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект;

AI – Artificial Intelligence – штучний інтелект;

gRPC – Google Remote Procedure Call – система віддаленого виклику процедур, розроблена Google, яка дозволяє клієнтам і серверам ефективно спілкуватися, використовуючи протокол HTTP/2 і формат даних Protocol Buffers;

LLAMA – Large Language Model Meta AI – велика мовна модель, розроблена компанією Meta AI, призначена для генерації тексту та обробки природної мови;

LLM – Large Language Model – велика мовна модель, що навчається на великих обсягах текстових даних для виконання завдань з обробки природної мови, таких як генерація тексту, переклад або узагальнення;

RVC – Realtime Voice Changer – система зміни голосу в реальному часі, яка застосовується для модифікації тембру або стилю мовлення під час розмови;

STT – Speech To Text – технологія перетворення мовлення в текст;

TTS – Text To Speech – технологія синтезу мовлення, яка перетворює текстову інформацію у звукову форму;

VAD – Voice Activity Detection – алгоритм виявлення мовної активності, який визначає наявність мовлення у звуковому потоці з метою оптимізації обробки аудіо.

ВСТУП

У сучасному цифровому суспільстві стрімко зростає попит на інноваційні інтерактивні технології, що поєднують розваги, штучний інтелект та соціальну взаємодію. Особливої популярності набувають проєкти, які створюють ілюзію живого спілкування між людиною та віртуальним персонажем у режимі реального часу.

У центрі подібних рішень великі мовні моделі (LLM), які здатні генерувати природні, логічні та стилістично відповідні відповіді. Проте створення повноцінної системи такого типу вимагає вирішення цілої низки технічних задач: від побудови стилістично узгодженого персонажа та реалізації інтерактивної логіки до інтеграції з голосовими та текстовими платформами, такими як Discord і Twitch. Надзвичайно важливо також забезпечити оперативність реакції, здатність підтримувати розмовний контекст, розпізнавати мовлення в реальному часі та повертати відповідь у вигляді синтезованого голосу.

Актуальність даної теми зумовлена не лише технологічною новизною, а й високим попитом на персоналізовані віртуальні взаємодії в індустріях розваг, освіти, підтримки клієнтів та соціальних медіа. Інтеграція LLM у стрімінгове середовище відкриває нові горизонти для розвитку креативних форматів спілкування, автоматизації модерації, створення ігрових сценаріїв та динамічного контенту на основі дій користувачів.

Метою цієї роботи є аналіз архітектури та побудова прототипу системи, що поєднує можливості LLM, технології розпізнавання мовлення, синтезу голосу, збереження контексту та інтеграцію з Discord як однією з основних платформ комунікації. У ході дослідження розглядатимуться способи адаптації мовної моделі до заданого стилю персонажа, технічні аспекти обробки голосового трафіку в реальному часі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз предметної галузі

В умовах активного розвитку технологій штучного інтелекту спостерігається стрімке зростання інтересу до інтерактивних голосових систем, які забезпечують природну комунікацію між людиною та комп’ютером. Особливе місце серед таких систем займають рішення, що базуються на великих мовних моделях, здатних не лише генерувати синтаксично правильні речення, а й дотримуватись стилістичної та контекстної послідовності у спілкуванні. Застосування LLM у голосових інтерфейсах відкриває нові можливості для реалізації віртуальних персонажів, чат-компаньйонів, цифрових помічників, а також інтерактивних стрімінгових агентів.

Інтеграція LLM з технологіями розпізнавання мовлення і синтезу мовлення дає змогу створювати системи, що підтримують повний цикл голосової взаємодії. Такі рішення стають дедалі актуальнішими у сфері розваг, онлайн-трансляцій, дистанційного навчання, технічної підтримки та індивідуалізованих користувацьких сервісів. Зокрема, платформа Discord, яка є одним із найпопулярніших засобів голосового спілкування серед молодіжної аудиторії, створює зручне середовище для реалізації інтелектуальних голосових агентів.

Серед основних завдань можна відзначити необхідність забезпечення низької затримки на кожному з етапів обробки, підтримку стилістичної стабільності персонажа, адаптацію до користувацьких особливостей, а також технічні труднощі інтеграції з голосовими каналами Discord.

Актуальність тематики зумовлена також зростаючим попитом на персоналізовану взаємодію між людиною та цифровими агентами.

Сучасний користувач очікує не лише на якісне розпізнавання мовлення, а й на емоційно забарвлені, релевантні відповіді у відповідному стилі, що створює ілюзію «живої» присутності віртуального співрозмовника.

1.2 Аналіз існуючих рішень

На даний момент існує декілька систем, що демонструють елементи інтерактивного голосового управління, проте жодна з них не є відкритою або повністю придатною для використання як універсальний шаблон. Найбільш близькими до концепції, покладеної в основу даної роботи, є такі рішення, як Neuro-sama [1], а також окремі експериментальні реалізації на базі локальних моделей типу LLaMA та RVC.

Проєкт Neuro-sama, який отримав широку популярність на стримінгових платформах, поєднує в собі мовну модель, систему генерації голосу та візуальний аватар. Його головна особливість полягає у створенні враження «живої особистості» з характером, що реагує на чат в реальному часі. Проте технічна реалізація Neuro-sama є закритою, що унеможливлює її адаптацію або масштабування без повного відтворення архітектури з нуля.

Деякі спроби реалізації голосових віртуальних персонажів здійснюються в середовищі Unity, у поєднанні з TTS-модулями і LLM. Такі проєкти здебільшого фокусуються на візуальній представленості, тоді як голосова частина реалізується у вигляді послідовного ланцюга: текст з чату, генерація, TTS, програвання. Їхня архітектура зазвичай не розрахована на захоплення голосу користувача та обробку в реальному часі, а також не передбачає можливості багатокористувацької взаємодії.

Окремі незалежні розробники будують відкриті рішення на базі Whisper, KoboldAI, Coqui TTS, RVC та Oobabooga UI, проте ці проєкти фокусуються переважно на окремих компонентах і рідко об’єднуються в єдиний робочий стек. Такі системи часто мають високі обчислювальні

вимоги, складну інсталяцію та відсутність продуманої архітектури для обробки голосових діалогів у режимі реального часу.

Аналіз існуючих рішень показує, що хоча окремі компоненти – мовні моделі, синтезатори мовлення та інструменти для розпізнавання мовлення вже досягли високого рівня розвитку, повноцінна інтегрована система з підтримкою голосової взаємодії, яка працює в режимі реального часу все залишається актуальною задачею для наукового і прикладного дослідження.

1.3 Генеративний штучний інтелект та великі мовні моделі як основа інтерактивної системи

Генеративний штучний інтелект (Generative AI) є новітнім і водночас одним із найвпливовіших напрямів розвитку сучасних інформаційних технологій. На відміну від традиційних алгоритмів машинного навчання, які здебільшого займаються класифікацією, розпізнаванням образів або регресійним аналізом, генеративний ШІ сконцентрований на створенні нових даних, які мають внутрішню логічну цілісність і відповідають закономірностям навчального середовища.

Суть генеративного підходу полягає в тому, що система, замість простої реакції на вхідні дані, активно формує відповідь, яка є результатом узагальнення, стилізації та семантичного прогнозування. В цьому контексті поняття генерації охоплює не лише буквальне створення нових об’єктів, а й імітацію когнітивної діяльності: моделювання мови, поведінки, інтонації, художнього стилю тощо.

Генеративний ШІ використовує різні типи моделей, серед яких провідне місце займають глибокі нейронні мережі, зокрема трансформери та дифузійні моделі. Завдяки великій обчислювальній потужності сучасного апаратного забезпечення стало можливим навчати ці моделі на масштабних мультимодальних корпусах даних, що дозволяє досягати

високої адаптивності й варіативності у відповіді. Такий підхід вже довів свою ефективність у широкому спектрі задач: від генерації текстів і зображень до створення цифрових агентів із ознаками соціальної взаємодії.

Однією з ключових особливостей генеративного ШІ є відсутність жорстко заданих правил – замість суворо формалізованих алгоритмів модель навчається на прикладах і самостійно виводить статистичні залежності. Це робить її гнучкою до нових контекстів і здатною до поведінки, що може наближатися до людської імпровізації. Така властивість особливо важлива у сфері діалогових систем та голосових інтерфейсів, де необхідно не лише зрозуміти запит, а й сформулювати природну, стилістично доречну відповідь.

В сучасному науковому і технічному середовищі генеративний ШІ поступово стає основою для побудови персоналізованих, автономних та інтелектуально адаптивних систем, які застосовуються у найрізноманітніших сферах: освіта, медицина, творчість, автоматизована підтримка, кібербезпека, цифровий маркетинг та інших.

Особливо стрімкий розвиток спостерігається в контексті мовоцентричних систем, де великі мовні моделі виступають центральним елементом. Вони не лише генерують змістовні текстові відповіді, а й забезпечують підтримку контексту діалогу, врахування стилістичних особливостей користувача та можливість логічного продовження розмови. Завдяки цьому LLM сьогодні розглядаються як архітектурна та функціональна основа більшості генеративних агентів, зокрема тих, що реалізуються у вигляді голосових або мультимодальних асистентів.

Генеративний ШІ формує нову парадигму взаємодії між людиною та цифровим середовищем – від реактивної до конструктивної, де машина не лише виконує команду, а й активно бере участь у формуванні інформаційного простору. Саме в цьому контексті розробка інтерактивної голосової системи, що поєднує LLM із сучасними інтерфейсами

комунікації, становить актуальне дослідницьке завдання, спрямоване на вивчення можливостей таких систем у реальному застосуванні.

1.3.1 Роль LLM у побудові генеративних систем штучного інтелекту

Великі мовні моделі (Large Language Models, або LLM) є центральним компонентом генеративного штучного інтелекту, коли йдеться про створення змістовного текстового контенту, моделювання діалогу та симуляцію природної мовної поведінки. Їхня поява і розвиток стали критично важливими для реалізації систем, що не просто імітують людську мову, а здатні генерувати осмислені, граматично правильні й стилістично релевантні відповіді у широкому спектрі комунікативних ситуацій.

На відміну від традиційних моделей обробки природної мови, rule-based систем або вузькоспеціалізованих seq2seq-архітектур, LLM навчаються на величезних масивах текстових даних загального призначення, що дозволяє їм охоплювати широкий контекст, запам’ятовувати шаблони мовлення та формувати відповідь, яка логічно продовжує або стилістично доповнює заданий текст. Вони є універсальними генеративними системами з високим ступенем узагальнення, що дозволяє використовувати їх не лише для відповіді на питання, а й для написання коду, імітації персонажів, перекладу, резюмування, анотації, написання художніх або технічних текстів.

Ключовим фактором ефективності LLM є їх величезна кількість параметрів – від мільярдів до сотень мільярдів. Це дозволяє моделям утримувати векторні подання складних мовних конструкцій і формувати глибокі семантичні зв’язки між словами, реченнями й абзацами. Такі властивості роблять LLM придатними до створення персоналізованої

комунікації, де система враховує попередні репліки користувача, стиль мовлення, контекст ситуації та навіть соціальні сигнали.

Крім того, LLM підтримують так звані інструкційні та рольові промпти, які дозволяють задавати стиль, поведінку або «персону» моделі ще до початку діалогу. Наприклад, користувач може задати контекст «відповідай як професійний психолог» або «говори як вигаданий персонаж», що кардинально змінює як тональність, так і структуру відповідей.

Із практичної точки зору, LLM стали основою великої кількості систем: від чат-ботів і віртуальних асистентів до автоматичних сценаріїв ведення бесіди в іграх, освітніх курсах, службах підтримки клієнтів тощо. Саме завдяки LLM стало можливим поєднання генеративного підходу із гнучкими формами взаємодії, зокрема голосовими системами, як у випадку цієї роботи.

Важливо також відзначити, що використання LLM у голосовій взаємодії потребує врахування низки технічних і лінгвістичних особливостей. Зокрема, діалогові системи мають справу з короткими, незавершеними або фонетично неоднозначними запитами, які потребують контекстної інтерпретації.

У результаті LLM перетворюються на інтелектуальні ядра, здатні адаптуватися до користувача, підтримувати стилістичну сталість, і водночас працювати в умовах високої динаміки, що притаманна голосовим інтерфейсам.

1.3.2 Архітектура LLM: трансформер та його принципи

Архітектурною основою більшості сучасних LLM є трансформер – глибока нейронна архітектура, запропонована у 2017 році дослідниками Google в роботі «Attention is All You Need». Її поява стала революційною подією в галузі обробки природної мови, оскільки

трансформери дозволили одночасно досягти значного підвищення якості генерації тексту та суттєвої оптимізації процесу навчання завдяки повній відмові від рекурентних механізмів.

Основна ідея трансформера полягає в тому, що модель одночасно аналізує всі елементи вхідної послідовності, незалежно від їх порядку, за допомогою механізму уваги. Це дозволяє їй ефективно враховувати довготривалі залежності у тексті, які традиційні рекурентні мережі моделювали з великими труднощами або втратами інформації.

Ключовим компонентом трансформера є механізм self-attention. Він дає змогу для кожного токену у вхідній послідовності «уважно подивитися» на інші слова у реченні та визначити, які з них є найбільш релевантними для формування його власного представлення. Саме завдяки цьому принципу модель може зберегти логічний зв’язок між частинами діалогу.

Self-attention реалізується через матричні перетворення запитів, ключів та значень, які комбінуються із використанням функції softmax, що формує ваги між токенами. Ці ваги визначають, на які частини тексту модель повинна звертати більшу увагу при генерації поточного елемента. Для покращення узагальнюючої здатності у трансформерах зазвичай використовується multi-head attention – паралельне застосування кількох механізмів уваги, які вивчають різні аспекти взаємозв’язків між словами.

Оскільки трансформери не мають вбудованого поняття порядку елементів, в них вводяться positional encoding – вектори, які додаються до вхідних ембедингів і дозволяють моделі враховувати послідовність токенів. Завдяки цьому трансформер здатний відновлювати граматичну і логічну структуру речень.

Архітектура LLM зазвичай реалізує каскадне поєднання десятків або навіть сотень таких трансформерних блоків, які включають нормалізацію, залишкові зв’язки та нелінійні перетворення. Під час навчання ці блоки оптимізуються на великих об’ємах текстових даних,

що дозволяє моделі поступово формувати багаторівневу семантичну ієрархію мови – від окремих слів до абзаців і стилістичних фрагментів.

У діалогових моделях використовується переважно декодерна частина трансформера, де кожен крок генерації нового токена залежить від попередніх, що робить її авто-регресивною. Саме такий підхід дозволяє LLM відповідати на запити з урахуванням як короткого, так і довгострокового контексту діалогу.

Архітектура трансформера, завдяки своїй масштабованості, паралелізації обчислень і адаптивності до контексту, стала універсальним каркасом для побудови сучасних генеративних систем (рисунок 1.1). Вона є не лише технічним фундаментом, а й концептуальним переходом до більш інтелектуальних способів моделювання мовлення, які базуються не на правилах, а на контекстно-залежному прогнозуванні.

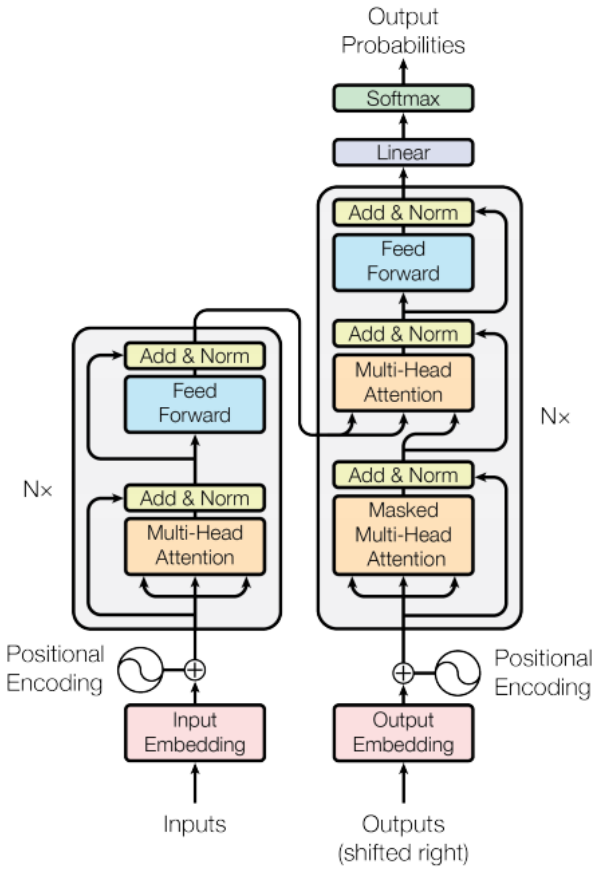

Рисунок 1.1 – Архітектура трансформера

1.4 Постановка задачі

У межах даної роботи ставиться завдання створити інтелектуальну голосову систему, яка забезпечує голосову взаємодію з користувачем в режимі реального часу через платформу Discord. Головною метою є реалізація програмного прототипу, здатного приймати голосові звернення, розпізнавати мовлення, формувати текстові відповіді за допомогою великої мовної моделі та транслювати ці відповіді у вигляді синтезованого голосу. Система повинна підтримувати стилістичну сталість у спілкуванні та реагувати з мінімальною.

Метою даної дипломної роботи є створення інтерактивної голосової системи на основі великих мовних моделей, призначеної для розважальної взаємодії з користувачами у режимі реального часу через голосовий канал Discord. Основна ідея полягає в реалізації прототипу віртуального персонажа, здатного сприймати голосові команди, генерувати змістовні відповіді в заданому стилі та озвучувати їх у голосовому чаті.

Для досягнення цієї мети необхідно реалізувати повний цикл – від захоплення аудіо в Discord до генерації голосової відповіді. Система повинна працювати у вигляді Discord-бота, який приєднується до голосового каналу, фіксує мовлення користувача, розпізнає його у текстовому форматі за допомогою STT-моделі, передає результат до великої мовної моделі для формування відповіді, після чого отриманий текст перетворюється у голос за допомогою TTS і транслюється у голосовий канал Discord.

Система має бути здатною визначати момент завершення репліки користувача, використовуючи механізми детекції голосової активності, щоб запобігти перериванню та забезпечити природність комунікації. Важливою частиною є організація короткострокової пам’яті, промпт повинен містити останні повідомлення для збереження контексту. Крім

того, необхідно реалізувати довгострокове збереження інформації про користувача що дозволить системі адаптувати свої відповіді залежно від історії взаємодії.

Архітектура рішення повинна забезпечувати модульність та масштабованість, із можливістю заміни, STT/TTS-модулями, а також налаштовуваним вибором мовної моделі. Реалізація проєкту відбуватиметься на мові програмування Python із використанням бібліотек discord.py, transformers, torch, whisper, coqui-TTS, websockets, asyncio та інших допоміжних інструментів.

В результаті має бути створено функціональний прототип, для взаємодії з користувачем: голос користувача, текст, генерація відповіді, голосова відповідь, із підтримкою стилістичної послідовності, затримкою, прийнятною для реального часу, та базовими механізмами персоналізації. Така система може слугувати основою для реалізації повноцінного віртуального стрімера, чат-компаньйона або інтерактивного розважального агента.

1.4.1 Визначення критичних аспектів функціонування системи в режимі реального часу

Під час створення системи, заснованої на великих мовних моделях, виникає ряд важливих технічних проблем, які необхідно розв'язати для забезпечення стабільної та якісної роботи. Основні з них:

– забезпечення швидкості відповіді. Одна з критичних вимог до систем такого типу, це швидкість формування відповідей. Взаємодія у режимі реального часу накладає суворі обмеження на затримку відповіді. Для її мінімізації необхідно використовувати оптимізовані механізми inference, такі як quantization, pruning, GPU/TPU прискорення та ефективні алгоритми кешування;

– відповідність стилю та особистості персонажа. Глядачі очікують, що модель буде поводитися у відповідності до заявленої особистості та стилю. Досягнення такої стабільності та послідовності у відповідях, складна задача, яка вимагає спеціалізованого fine-tuning на ретельно підібраних датасетах та застосування структурованих промптів;

– управління токсичністю та модерація контенту. Оскільки система працюватиме у відкритому середовищі з аудиторією, вона може зіткнутись із проблемою токсичних, образливих або небажаних відповідей. Для вирішення цього питання необхідно створювати окремі механізми фільтрації (наприклад, інтеграцію класифікаторів токсичності) та запроваджувати жорстку політику модерації контенту на рівні LLM;

– збереження та використання контексту розмови. Інтерактивна комунікація передбачає підтримання контексту діалогу, що у свою чергу вимагає реалізації короткострокової та довгострокової пам’яті. Короткострокова пам’ять відповідає за актуальний контекст діалогу, а довгострокова забезпечує накопичення та використання інформації про користувачів, їх вподобання та історію взаємодії, для чого зазвичай застосовують зовнішні сховища даних;

– інтеграція з зовнішніми системами (Integration complexity). Система повинна інтегруватися з різними платформами, Twitch, Discord, тощо, які використовують різні формати та протоколи. Наприклад, для отримання тексту з голосового каналу Discord потрібне застосування технологій перетворення мови у текст, які мають свої технічні обмеження та проблеми (якість розпізнавання, затримки, потреба у високій якості мікрофонів тощо).

Вирішення цих ключових проблем дозволить створити ефективну та надійну інтерактивну розважальну систему на базі LLM, яка буде максимально наближена до очікувань користувачів.

1.4.2 Відповідність стилю та особистості персонажу

Однією з ключових умов успішної реалізації інтерактивної розважальної системи на основі великих мовних моделей (LLM) є забезпечення відповідності генерованих відповідей стилю та особистості заданого персонажа. Це дозволяє сформувати послідовний та впізнаваний образ, який викликає довіру і симпатію у глядачів, а також підвищує рівень залученості аудиторії.

Першочерговим завданням є формалізація особистісних характеристик персонажа, його стилістичних особливостей та тональності спілкування. Необхідно заздалегідь визначити, яким буде загальний образ: жартівливим, серйозним, доброзичливим чи, навпаки, провокативним. Такий опис має охоплювати спосіб побудови речень, вибір слів, рівень емоційності, а також характер поведінки в різних ситуаціях, наприклад, у відповідь на критику, жарт, донат або технічне питання. Від якості цього опису буде залежати послідовність стилістичних проявів під час генерації відповідей.

Для того щоб модель дотримувалась заданого стилю протягом усього часу взаємодії, застосовуються кілька технічних рішень. По-перше, важливо провести тонке налаштування (fine-tuning) моделі на вибірках, які передають бажану манеру спілкування персонажа. По-друге, під час роботи LLM використовуються промпти зі стилістичною інструкцією, яка нагадує моделі, у якому образі вона має відповідати. Наприклад, модель може отримати вказівку говорити жартівливо, емоційно, з використанням сучасних сленгових виразів. Крім того, в систему можна інтегрувати контекстні тригери, які змушують модель змінювати тон відповіді залежно від подій, що відбуваються: реакція на підписку, привітання нового учасника або отримання великого донату.

З метою забезпечення стабільності образу персонажа та недопущення стилістичних відхилень, необхідно впровадити механізми

контролю. До таких заходів належить автоматична перевірка згенерованих відповідей на відповідність заданим параметрам стилю, що може реалізовуватися через окремі класифікаційні модулі. Окрім автоматизованого контролю, бажаним є періодичний ручний аудит відповідей модераторами або розробниками. Це дозволяє своєчасно виявляти випадки, коли модель «виходить з образу», і відповідно коригувати її роботу, оновлювати дані для fine-tuning або змінювати промптову логіку.

Отже, реалізація стилістичної та особистісної відповідності персонажу є комплексним процесом, що охоплює чітке формулювання образу, налаштування моделі на відповідні мовні патерни, адаптацію промптів та постійний контроль за результатами. Успішне впровадження цих елементів забезпечить створення переконливого, стабільного та привабливого віртуального персонажа, що значно покращить загальну якість взаємодії з аудиторією.

1.4.3 Як оптимально використовувати промпти

Промпт – це початковий текст, який задає контекст та інструкції для генерації відповіді великою мовною моделлю (LLM). Ефективне використання промптів є важливим фактором, що визначає якість, точність та відповідність відповідей. Нижче наведено рекомендації щодо оптимального створення промптів з технічної та методологічної точки зору:

– чіткі та конкретні інструкції. Промпти повинні містити максимально зрозумілі інструкції щодо того, як модель повинна формулювати відповіді. Наприклад, це можуть бути вказівки на довжину відповіді, бажаний формат (розмовний, формальний, гумористичний стиль), або обмеження на тематику (заборонені теми чи формулювання). Наприклад: «Відповідай короткими реченнями (не більше 2–3 речень).

Використовуй позитивний та дружній тон. Уникай будь-яких політичних або релігійних тем.»;

– опис особистості та мовних особливостей персонажа. Оскільки мовна модель має імітувати певний персонаж, важливо у промпті чітко вказувати ключові характеристики особистості, стиль мовлення, використання специфічної лексики або сленгу. Це допомагає підтримувати консистентність характеру відповідей. Наприклад: «Ти дружній і позитивний віртуальний персонаж, що спілкується легко, використовує жарти та популярні меми. Не соромся висловлювати емоції та реагуй енергійно на повідомлення користувачів.»;

– контекст попередньої взаємодії. Вкрай важливим є врахування останніх повідомлень у чаті, які визначають поточну тему діалогу. Передача останніх реплік користувачів і відповідей моделі у промпті допомагає моделі формувати послідовні та контекстно релевантні відповіді. Наприклад: «Користувач: Який твій улюблений фільм? Персонаж: Я обожнюю «Зоряні війни»! А який у тебе? Користувач: Я теж люблю фантастику. Як тобі «Інтерстеллар»?»;

– динамічна структура промптів. Для забезпечення ефективної взаємодії промпти повинні оновлюватися в залежності від ситуації та подій у стрімі. Наприклад, у разі отримання особливих подій (донати, нові підписки, цікаві коментарі) до промпта додаються відповідні деталі, що забезпечує персоналізацію та актуальність відповідей. Наприклад: «Користувач: [username] задонатив 50$. Його повідомлення: «Ти найкращий!». Подякуй емоційно та додай жарт.»;

– оптимізація довжини промптів. Важливо враховувати технічні обмеження мовних моделей щодо обсягу вхідного тексту. Для оптимальної роботи рекомендується використовувати промпти помірної довжини, уникаючи перевантаження моделі зайвою інформацією. Це дозволяє ефективно використовувати доступні ресурси для генерації відповідей без втрати якості.

1.4.4 Дані для fine-tuning моделі

Для створення системи стрімінгу на основі великих мовних моделей (LLM) критично важливим етапом є процес тонкого налаштування (fine-tuning) моделі на відповідних наборах даних. Правильно підібрані та якісно підготовлені дані безпосередньо впливають на релевантність, стилістичну послідовність та загальну якість генерації контенту. Основні категорії даних, рекомендовані для такого навчання, включають наступні типи:

– записи реальних стрімів та діалогів. Оптимальними джерелами для fine-tuning є діалогові взаємодії з реальних трансляцій, які ведуть популярні стрімери або персонажі, що мають бажаний стиль комунікації. Ці дані допомагають моделі перейняти автентичну стилістику мови, реакції на різноманітні питання, коментарі та поведінку аудиторії;

– штучно створені або модельовані діалоги. Для точнішого контролю за стилістикою та тематикою відповідей можна створювати спеціальні штучні набори даних. Такі дані формуються сценаристами або експертами з NLP і містять типові питання-відповіді, характерні для персонажа;

– дані із соціальних мереж, форумів та чатів. Корисним додатковим джерелом можуть бути дані, зібрані з публічних платформ (Reddit, Discord, Twitter), але вони потребують особливо ретельної очистки та модерації. Основним критерієм їх відбору має бути відповідність бажаному стилю і відсутність токсичного або образливого контенту.

Важливим етапом є попередня обробка даних перед fine-tuning, яка включає:

– видалення дублювань, шуму та некоректних текстів;

– використання інструментів NLP для перевірки граматичної правильності та стилістичної послідовності;

– розбиття діалогів на логічні фрагменти, що полегшує навчання моделі.

Для процесу fine-tuning доцільно використовувати сучасні бібліотеки, такі як Hugging Face Transformers, з алгоритмами оптимізації, що дозволяють досягати високої ефективності навчання. Важливо правильно вибрати розмір набору даних: він має бути достатньо великим для узагальнення стилю, але не надмірно великим, щоб уникнути перенавчання (overfitting).

Якісно підібрані та ретельно підготовлені набори даних для fine-tuning забезпечують точну стилістичну адаптацію мовної моделі, значно покращуючи якість взаємодії з аудиторією та створюючи переконливий і автентичний образ віртуального персонажа.

1.4.5 Функціональні особливості та технічна реалізація

Мовна модель, що використовується в інтерактивних розважальних стрімінгових системах, володіє специфічним набором функцій, які забезпечують багату взаємодію з аудиторією та зовнішніми додатками. Функціонал який може бути реалізовано:

– генерація відповідей у реальному часі. Для забезпечення швидкої реакції моделі застосовуються спеціальні алгоритми inference, які оптимізуються за допомогою технологій quantization, pruning (обрізання моделі) та GPU/TPU-прискорення. Це дозволяє мінімізувати затримки та підтримувати якісну взаємодію в режимі реального часу;

– персоналізовані реакції на події чату. Персоналізація реакцій здійснюється через інтеграцію окремих програмних модулів, що аналізують специфічні події у стрімі (наприклад, нові підписки, донати). Ці дані передаються у вигляді додаткових контекстних елементів до промптів, завдяки чому модель може адаптувати відповіді, звертаючись персонально до користувачів;

– емоційна адаптація відповідей, реалізується через використання спеціалізованих класифікаторів емоцій, що аналізують вхідні текстові повідомлення та визначають емоційний стан (наприклад, радість, сум, захоплення, здивування). Виявлена емоція включається у промпт як спеціальний маркер, що дозволяє моделі генерувати відповіді, адаптовані до емоційного контексту взаємодії. Приклад реалізації у лістингу 1.1;

Лістинг 1.1 – Приклад промпту до моделі з емоційним контекстом

```
{"user_message": "Це було дуже смішно!",
"detected_emotion": "joy",
"prompt": "Відповідай весело та підтримуй позитивну атмосферу."}
```

– взаємодія LLM з ігровими інтерфейсами. Для інтеграції LLM у процес ігрової взаємодії може використовуватися спеціальний API, через який модель може як отримувати інформацію про поточний стан гри, так і надсилати команди безпосередньо в її інтерфейс. Це дозволяє моделі брати участь у покрокових іграх, забезпечуючи автоматичне прийняття рішень на основі доступних ігрових даних.

– гра надсилає моделі актуальний стан гри через JSON-інтерфейс, приклад у лістингу 1.2;

Лістинг 1.2 – Приклад передачі стану гри через JSON

```
{"current_turn": "LLM",
"available_cards": ["Карти №2, №5, №8"],
"game_state": {
"player_scores": {"LLM": 10, "Opponent": 12},
"round": 3}}
```

– модель формує відповідь, обираючи оптимальну стратегію:

```
{"action": "play_card",
"selected_card": "№5"};
```

– отримана команда передається до гри, і після виконання ходу гра повертає оновлений стан. Це створює безперервний інтерактивний цикл, який дозволяє моделі активно взаємодіяти у межах ігрового процесу.

Модель здатна отримувати текстову інформацію для аналізу з двох основних джерел:

– Twitch-чат: прямий потік текстових повідомлень від глядачів у реальному часі, який передається у промпти для генерації відповідей;

– Discord: для голосових каналів Discord застосовується технологія розпізнавання мови, яка перетворює голос користувачів на текст у реальному часі. Отримані текстові репліки далі передаються моделі для аналізу і формування відповіді.

Технічні рішення, представлені вище, забезпечують ефективну, багатофункціональну взаємодію LLM з аудиторією, а також можливість інтегрувати модель в ігрові сценарії, що створює додаткову цінність та підвищує інтерактивність системи;

1.4.6 Короткострокова та довгострокова пам’ять

Для забезпечення якісної інтерактивної комунікації у стрімінгових системах на основі великих мовних моделей (LLM) особливо важливим є ефективне управління пам’яттю, яка поділяється на короткострокову та довгострокову. Ці два типи пам'яті відіграють різні ролі і мають специфічні методи реалізації.

Короткострокова пам'ять відповідає за підтримку безпосереднього контексту діалогу, дозволяючи моделі надавати логічно пов’язані та релевантні відповіді. До промпта моделі включаються останні кілька повідомлень (наприклад, останні 5-10 реплік користувачів та відповідей моделі). Це дозволяє зберігати релевантний контекст та послідовність діалогу.

Оскільки великі мовні моделі мають обмеження щодо максимальної довжини промпта, важливо оптимально підбирати кількість повідомлень, які зберігаються у короткостроковій пам’яті. Для цього використовують алгоритми автоматичного скорочення тексту, видалення менш важливих повідомлень та стиснення контексту за допомогою семантичних моделей або кластеризації інформації.

Довгострокова пам'ять відповідає за зберігання інформації протягом тривалого періоду, включаючи дані про користувачів, історію взаємодій, переваги аудиторії та важливі події на стрімі. Реалізація довгострокової пам’яті технічно включає наступні методи:

– векторні бази даних, найбільш популярним рішенням для організації довгострокової пам’яті є бази даних такі як, Pinecone, FAISS, Qdrant. Дані про користувачів та події зберігаються у вигляді векторних представлень, що дозволяє швидко знаходити релевантну інформацію для поточного контексту через векторний пошук;

– ключ-значення сховища, широко використовуються NoSQL бази даних типу ключ-значення, такі як Redis чи MongoDB, для зберігання та швидкого доступу до структурованої інформації (наприклад, профілі користувачів, історія донатів, налаштування);

– інтеграція довгострокової пам’яті у промпти, перед генерацією відповіді важливі дані витягуються з довгострокового сховища, обробляються, сумуються або кластеризуються, і додаються до промпта як додатковий контекст. Це дозволяє моделі відповідати персоналізовано, враховуючи індивідуальні особливості та вподобання аудиторії. Приклад реалізації наведено у лістингу 1.3.

Лістинг 1.3 – Приклад запиту до LLM з врахуванням індивідуальних особливостей

```
{"user_profile": {
"username": "user123",
```

Продовження лістингу 1.3

```
    "interaction_count": 27,
    "last_donation_amount": 50,
    "favorite_topics": ["ігри", "наука", "фантастика"]},
    "recent_stream_events": ["нова підписка", "великий донат від користувача user123"],
    "context_summary": "користувач часто запитує про новинки у світі відеоігор"}
```

Даний підхід до організації пам’яті дозволяє максимально ефективно використовувати ресурси мовної моделі, забезпечуючи якісну та персоналізовану взаємодію з глядачами як у короткостроковому, так і в довгостроковому горизонті часу.

Інтеграція голосової системи з платформою Discord створює низку специфічних технічних викликів, які необхідно врахувати для забезпечення коректної та стабільної роботи. Discord спочатку не був розрахований на двосторонню обробку мовлення зі сторонніх додатків у режимі реального часу, тому створення голосового інтерфейсу потребує реалізації низькорівневого доступу до аудіопотоку та додаткових оптимізацій.

1.4.7 Що таке Discord

Discord це багатофункціональна цифрова платформа для спілкування, яка поєднує можливості голосових, відео та текстових каналів. Вона спочатку була створена для потреб геймерської спільноти, однак з часом набула широкого поширення у сфері освіти, корпоративного спілкування, стримінгу та онлайн-спільнот. Основна мета Discord – забезпечити зручне середовище для комунікації в реальному часі в межах тематичних спільнот, що організовуються у вигляді окремих серверів.

Кожен сервер у Discord може містити численні текстові й голосові канали, які налаштовуються відповідно до потреб користувачів. У голосових каналах учасники можуть спілкуватися у режимі конференції, а також взаємодіят ботами, які підключаються до каналів як окремі віртуальні учасники. Такі боти можуть виконувати автоматизовані дії: реагувати на команди, транслювати аудіо, здійснювати модерацію тощо.

Discord має відкритий програмний інтерфейс, який дозволяє створювати користувацькі розширення і програмні боти. Для взаємодії з API найчастіше використовуються спеціалізовані бібліотеки мовою програмування Python, такі як discord.py, pycord, nextcord тощо. За допомогою цих інструментів можна реалізовувати гнучку логіку поведінки бота, обробляти події з текстових і голосових каналів, а також виконувати інтеграцію зі сторонніми сервісами, включаючи штучний інтелект.

У контексті даної роботи Discord обрано як цільову платформу для розгортання інтерактивної голосової системи з двостороннім обміном аудіо. Такий вибір обумовлений тим, що платформа підтримує роботу з великою кількістю користувачів в реальному часі та має стабільну інфраструктуру для голосового зв’язку.

1.4.8 Запис аудіо з Discord-каналу

Процес запису аудіо з голосового каналу Discord є важливою технічною складовою систем інтерактивної взаємодії на основі великих мовних моделей. Для запису голосового потоку з Discord, як правило, застосовують ботів, що працюють на базі спеціалізованих бібліотек, таких як discord.py із додатковим модулем discord-ext-voice-recv або аналогічними рішеннями на інших мовах програмування.

Підключення до голосового каналу Discord відбувається наступним чином:

– спочатку бот автентифікується на сервері Discord за допомогою спеціального токена, отриманого через Discord Developer Portal. Після автентифікації бот приєднується до цільового голосового каналу за командою користувача або автоматично під час запуску системи;

– запис голосового потоку від користувачів здійснюється у реальному часі. З технічної точки зору це відбувається шляхом перехоплення UDP-пакетів, які містять аудіодані, з наступним їх декодуванням. Бібліотека Discord.py забезпечує отримання та обробку цих пакетів, перетворюючи їх у зручний для подальшого використання формат PCM.

Отримане аудіо може оброблятися двома основними способами:

– аудіопотік відразу записується у буфер, який періодично передається на обробку для подальшого розпізнавання мови. Такий підхід забезпечує мінімальні затримки і дозволяє системі оперативно реагувати на голосові повідомлення;

– альтернативний підхід, це періодичний запис голосових даних у тимчасові WAV аудіофайли, що може бути зручним для деяких моделей STT. При цьому бот створює сегментовані файли тривалістю, наприклад, 2–5 секунд, після чого одразу передає їх далі у систему для обробки.

Також важливим є забезпечення належної якості запису. Discord-комунікація може містити фонові шуми, перешкоди або інші аудіо-артефакти, що ускладнює подальше розпізнавання голосу. Для покращення якості аудіо застосовуються алгоритми шумозаглушення, нормалізації рівня звуку та фільтрації непотрібних частот.

Крім цього, важливим моментом є розмежування аудіопотоків від різних користувачів. Для кожного активного учасника голосового каналу бот формує окремий аудіопотік, який ідентифікується за унікальним ідентифікатором користувача (user ID). Це дозволяє здійснювати паралельну обробку кількох голосових потоків та забезпечує персоналізовану взаємодію з кожним учасником каналу.

Загалом, ефективна технічна реалізація запису аудіо з Discord-каналу є ключовою складовою інтерактивних систем на основі штучного інтелекту, що впливає на якість, швидкість та надійність подальших етапів обробки інформації.

1.4.9 Перетворення аудіо в текст (Speech-to-Text)

Після захоплення аудіо з голосового каналу Discord, наступним критичним етапом стає його перетворення у текстову форму. Цей процес здійснюється за допомогою технологій розпізнавання мовлення – STT, які дають змогу LLM обробити змістовну частину голосової взаємодії.

Для реалізації ефективної системи розпізнавання аудіо в текст із Discord в режимі реального часу необхідно враховувати:

– формат аудіоданих, найкращі результати отримуються при використанні формату PCM/WAV з частотою дискретизації 16 кГц та одним каналом;

– буферизацію та сегментацію, ефективне розпізнавання в реальному часі досягається шляхом розділення аудіо на короткі сегменти 2–5 секунд, що дозволяє негайно отримувати текстові результати.

В результаті, правильний вибір моделі для перетворення аудіо в текст залежить від вимог до точності, швидкості, доступних ресурсів, необхідності захисту даних і бюджету проєкту. Локальні open-source рішення оптимальні для конфіденційності та невеликих навантажень, тоді як хмарні сервіси забезпечують високу точність, масштабованість та зручність за умов стабільного інтернет з'єднання.

1.4.10 Визначення моменту відповіді

Одним із ключових моментів, що впливає на інтерактивність системи та природність спілкування при голосовій взаємодії у Discord, є

точне визначення того, коли співрозмовник завершив свою репліку і настав час для відповіді системи. Для цього застосовуються спеціалізовані алгоритми детекції голосової активності, відомі як Voice Activity Detection (VAD). Процес детекції голосової активності:

– отримання аудіопотоку, VAD отримує аудіопотік в реальному часі від користувача, зазвичай у форматі PCM, що дозволяє здійснювати аналіз сигналу практично без затримки;

– обробка та аналіз сигналу, алгоритм аналізує аудіо, визначаючи рівень енергії, спектральні характеристики сигналу, частоту і тривалість пауз, що дозволяє розмежувати мовлення та тишу;

– визначення меж мовлення, система встановлює поріг, при перевищенні якого сигнал вважається мовленням, а при падінні нижче - тишею. Якщо тиша триває довше визначеного інтервалу (наприклад, 300–500 мілісекунд), VAD робить висновок, що користувач завершив фразу, і запускає процес генерації відповіді за допомогою LLM і TTS.

Існує низка популярних методів і бібліотек для реалізації VAD:

– WebRTC VAD: простий, швидкий і широко використовуваний алгоритм, що працює на основі енергетичного аналізу сигналу;

– Silero VAD: модель на основі нейронних мереж, яка характеризується високою точністю і добре працює у шумних середовищах;

– Pyannote.audio: бібліотека з нейромережевими моделями для задач сегментації мовлення та визначення голосової активності;

– Auditok: простий у використанні інструмент, який аналізує гучність та частотні характеристики сигналу для визначення меж мовлення.

Для забезпечення ефективної роботи VAD у системах, які працюють в реальному часі, важливо враховувати такі аспекти як:

– буферизація аудіо: необхідна для стабільного аналізу сигналу в реальному часі з мінімальною затримкою;

– адаптивність: автоматичне налаштування порогових значень залежно від фонового шуму і якості мікрофона користувача;

– затримки (latency): мінімізація затримок при аналізі та обробці аудіо, що забезпечує швидку і природну реакцію системи.

Правильна реалізація механізм VAD дозволяє суттєво підвищити інтерактивність і зручність використання систем на основі штучного інтелекту, забезпечуючи природний діалог із користувачем у реальному часі.

1.4.11 Перетворення тексту у голос

Після того як велика мовна модель сформувала текстову відповідь, наступним кроком є її перетворення у голосове повідомлення для відтворення у голосовому каналі Discord. Цей процес реалізується за допомогою технологій синтезу мовлення, відомих як TTS. Існують як локальні, так і хмарні рішення для цієї задачі, кожне з яких має свої переваги й обмеження:

а) Coqui TTS – високоякісна open-source система, що базується на нейромережах, є популярним вибором для локального розгортання;

– переваги: висока якість синтезованого мовлення, підтримка кількох мов, можливість глибокого налаштування голосів, повна автономність і конфіденційність;

– недоліки: потреба у значних обчислювальних ресурсах, особливо при використанні великих моделей, а також необхідність часу на підготовку та навчання власних моделей;

б) eSpeak NG – легковажне й швидке рішення, яке використовує формантний підхід до генерації мовлення;

– переваги: працює на мінімальних ресурсах, сумісне з великою кількістю пристроїв, безкоштовне і просте у використанні;

– недоліки: якість голосу значно нижча, звучання доволі штучне, з обмеженим рівнем виразності;

в) Festival/Festvox – класичні системи синтезу мовлення, які широко застосовуються для наукових та навчальних цілей;

– переваги: підтримка багатьох мов, доступ до різноманітних попередньо налаштованих голосів, можливість регулювання інтонації та наголосів;

– недоліки: складність у налаштуванні, нижча якість звучання в порівнянні з сучасними нейронними моделями;

г) ElevenLabs – сучасний онлайн-сервіс синтезу мовлення, що використовує глибокі нейронні мережі;

– переваги: реалістичність голосу, природна інтонація, емоційне забарвлення, простота інтеграції через API;

– недоліки: платний сервіс, особливо при великому обсязі генерації; потребує стабільного інтернет-з'єднання, що може викликати затримки;

д) Google Text-to-Speech (Google Cloud TTS) – масштабований хмарний сервіс з великою кількістю мов і стилів озвучення;

– переваги: висока якість мовлення, підтримка емоційних і нейтральних голосів, зручна документація та інтеграція;

– недоліки: вартість збільшується з масштабом використання, постійна залежність від інтернету;

е) Amazon Polly – один з ключових компонентів екосистеми AWS, який пропонує сучасні голоси для інтеграції у різноманітні продукти;

– переваги: добрий рівень якості мовлення, гнучкість у виборі мов і голосів, глибока інтеграція з іншими сервісами Amazon;

– недоліки: висока вартість при великому трафіку синтезу, залежність від стабільності інфраструктури AWS.

При виборі системи синтезу мовлення для Discord необхідно враховувати кілька ключових факторів. Насамперед, це якість голосу, що

безпосередньо впливає на враження користувачів. Затримка від моменту генерації до озвучення також критично важлива, локальні системи зазвичай мають нижчі показники затримки, тоді як хмарні забезпечують вищу якість, але з потенційними паузами через мережу. Конфіденційність відіграє роль у проектах, де йдеться про обробку чутливих даних: у таких випадках перевага надається офлайн-рішенням. І, нарешті, масштабованість, хмарні сервіси краще підходять для систем із високим навантаженням і великою кількістю одночасних запитів.

У висновку, локальні TTS-системи доречні в умовах, коли пріоритетом є автономність, мінімальні затримки та безпека даних. Онлайн-рішення, зі свого боку, забезпечують найвищу якість синтезованого мовлення, легкість інтеграції та гнучкість масштабування, що робить їх ефективним вибором для широкого спектра розважальних і комерційних застосунків.

2 АНАЛІЗ ТА ВИБІР ПРОГРАМНИХ ЗАСОБІВ І МОВИ РЕАЛІЗАЦІЇ СИСТЕМИ

2.1 Структура системи та визначення її ключових функціональних компонентів

Для успішної реалізації інтерактивної голосової системи важливим етапом є вибір відповідних технологій, інструментів і мов програмування. В ході виконання даної роботи було прийнято рішення використовувати мову програмування Python. Вибір цієї мови обумовлений її простотою, зрозумілим синтаксисом, широкими можливостями для інтеграції із сучасними бібліотеками машинного навчання та глибокого навчання PyTorch, TensorFlow, а також численними готовими інструментами для роботи з аудіо і текстом.

Для побудови повноцінного рішення необхідно забезпечити наступні компоненти:

– велика мовна модель;

– Discord бот для взаємодії із голосовими каналами;

– модулі розпізнавання мовлення та синтезу голосу;

– серверна частина для та передачі даних.

Вибір технологій базується на оптимальному поєднанні продуктивності, простоти інтеграції, підтримки необхідного функціоналу та наявності готових інструментів, які дозволяють знизити час на розробку та забезпечити високу якість кінцевого продукту.

Для реалізації інтерактивної голосової системи, яка здатна вести розмову з користувачами через платформу Discord в режимі реального часу, необхідно забезпечити взаємодію наступних компонентів: LLM, Discord-бот, TTS, модуль розпізнавання мовлення у реальному часі та серверна частина, що забезпечує комунікацію між цими компонентами (рисунок 2.1).

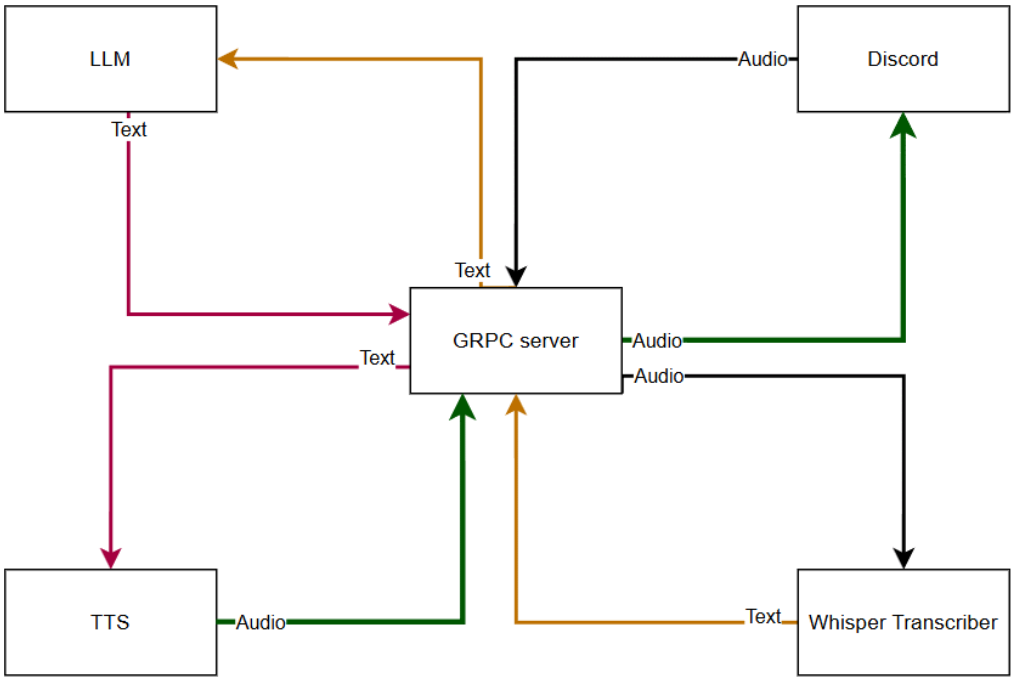

Рисунок 2.1 – Архітектура системи

Описана архітектура є модульною і легко масштабованою, що дозволяє за необхідності замінювати окремі компоненти на альтернативні рішення без значних змін у загальній структурі. Кожен модуль працює автономно, виконуючи специфічні задачі, що сприяє зменшенню складності загальної системи та полегшує процеси тестування і підтримки. Використання Python дозволяє швидко інтегрувати усі складові за допомогою наявних бібліотек і фреймворків, забезпечуючи високий рівень продуктивності та швидкий процес розробки.

2.2 Огляд і порівняння великих мовних моделей

Великі мовні моделі являють собою складні нейромережеві системи, призначені для обробки та генерації текстової інформації. Основною інновацією цієї архітектури є механізм уваги, який дозволяє моделі ефективно зосереджуватися на найбільш важливих частинах

вхідних даних при генерації відповіді, що значно підвищує її якість та релевантність.

Трансформер складається з декількох шарів самоуваги (self-attention), що дозволяють моделі одночасно обробляти слова в тексті, визначати їх семантичні зв’язки і формувати контекстно відповідні репрезентації. Важливою особливістю є те, що на відміну від рекурентних нейронних мереж, трансформери можуть ефективно обробляти великі обсяги інформації паралельно, що сприяє високій швидкості роботи навіть для моделей із величезною кількістю параметрів.

Принцип роботи великих мовних моделей базується на двох основних етапах:

– попереднє навчання. Модель навчається на великих обсягах текстових даних, зазвичай із використанням маскованого мовного моделювання (masked language modeling) або прогнозування наступного слова (next-token prediction). Це дозволяє моделі засвоїти глибокі семантичні та синтаксичні закономірності природної мови;

– тонке налаштування. Модель додатково навчається на специфічних наборах даних, що відповідають конкретній задачі або стилю, що дозволяє точніше адаптувати її до специфічних умов використання.

Великі мовні моделі широко застосовуються в задачах генерації текстів, машинного перекладу, діалогових системах, автоматизованій підтримці користувачів, аналізі текстів і багатьох інших сферах. Нижче наведено порівняння деяких найбільш поширених моделей з огляду на ключові характеристики.

LLAMA (Large Language Model Meta AI) це серія відкритих мовних моделей, розроблених компанією Meta. Вони вирізняються гнучкістю налаштування та високою продуктивністю у різноманітних задачах генерації тексту. LLAMA використовує архітектуру трансформера і підтримує моделі різних масштабів – від порівняно компактних із сімома

мільярдами параметрів до значно більших, зокрема варіантів на сімдесят мільярдів параметрів. Це дає змогу обирати модель, яка найкраще відповідає вимогам до швидкодії та доступних обчислювальних ресурсів.

Як активаційну функцію застосовано SwiGLU – модифікований варіант GLU, який демонструє кращу збіжність і ефективність на практиці. Для представлення позиційних залежностей у тексті модель використовує RoPE (Rotary Positional Embeddings), що забезпечують обертове кодування позицій, зберігаючи інформацію про відносне розташування токенів без необхідності жорсткої позиційної розмітки. Це рішення дозволяє LLaMA 2 краще справлятися з обробкою довгих контекстів. Крім того, у механізмах уваги реалізовано оптимізаційні принципи, зокрема Grouped Query Attention, що зменшує обчислювальні витрати при збереженні якості. Базове контекстне вікно становить 4096 токенів, однак у модифікованих реалізаціях воно може бути розширене до 16 тисяч і більше. Важливим є те, що LLAMA підтримує багатомовність і може ефективно використовуватись у різних мовних середовищах.

OpenAI є найбільш відомими і популярними у сфері великих мовних моделей, зокрема GPT-3, GPT-3.5 і GPT-4. Однією з ключових переваг OpenAI є доступність потужного API, який дозволяє легко інтегрувати модель у різноманітні програмні продукти без значних витрат часу на розгортання та налаштування. GPT-4, наприклад, здатний не лише генерувати високоякісний текст, але й ефективно вирішувати складні задачі.

Особливістю GPT-4 є наявність розширеного контекстного вікна до 32 тисяч токенів, що забезпечує збереження інформації у довгих діалогах або при роботі з об’ємними документами. Крім того, модель підтримує вхідні інструкції, які задають поведінкову рамку для всієї сесії взаємодії, дозволяючи моделювати персоналізованих агентів із чітко заданим стилем, роллю або мовленнєвою стратегією. Завдяки вбудованим

механізмам безпеки, зокрема контент-фільтрам і політикам RLHF (Reinforcement Learning from Human Feedback), GPT-4 придатна для публічного використання навіть у відкритих голосових системах.

Недоліками використання моделей OpenAI можуть бути залежність від комерційних тарифних планів, що створює потенційну фінансову навантаженість, а також питання конфіденційності, оскільки всі запити обробляються у хмарі.

Mistral це порівняно нова і високопродуктивна відкрита мовна модель, яка набирає популярності завдяки своїй компактності але при цьому високій швидкості та точності генерації текстів. Модель оптимізована для роботи на менших обчислювальних ресурсах, зокрема CPU і середньому рівні GPU, що робить її привабливою для локальних реалізацій та інтеграції у реальні додатки. Однією з ключових особливостей є впровадження механізму Sliding Window Attention (SWA), що забезпечує обробку довгих послідовностей зі зменшеною обчислювальною складністю, оскільки кожен новий токен взаємодіє лише з обмеженим підмножиною попередніх. Це рішення дозволяє досягати майже лінійної масштабованості по відношенню до довжини послідовності.

Окрім цього, у моделі реалізовано Grouped Query Attention механізм, що обслуговує декілька запитів спільно, оптимізуючи використання обчислювальних ресурсів. Для прискорення стримінгового використання в режимі реального часу використовується rolling cache механізм збереження ключів і значень у пам’яті без повного переобчислення. Стандартне контекстне вікно становить 8192 токени, однак існують модифікації з розширеним буфером.

Переваги Mistral включають її ефективність роботи на доступних апаратних ресурсах, високу швидкість відповіді та хорошу підтримку багатомовності, проте відносно нижчий рівень глибини і контекстної

складності порівняно з більшими моделями, що може обмежити її застосування у складних сценаріях взаємодії.

Окрім перелічених вище, існує багато інших моделей, які можуть бути ефективними у специфічних випадках. До таких можна віднести:

– BLOOM це багатомовна відкрита модель із високою підтримкою багатьох мов, зокрема менш поширених;

– Falcon це модель, що характеризується гарною продуктивністю та оптимізацією для швидкої роботи на GPU;

– Cohere це комерційна модель, орієнтована на високоякісні комерційні інтеграції та швидке налаштування під потреби бізнесу.

Вибір моделі LLM залежить від конкретних вимог до якості генерації тексту, доступних ресурсів, бюджету проєкту, вимог до швидкості обробки запитів та рівня конфіденційності даних. Використання найбільш підходящої моделі дозволить досягти максимальної ефективності створюваної системи. Проте, архітектура системи є модульною, вибір вибір моделі не є критичним, оскільки за необхідності її можна буде легко замінити на більш доцільну, навчену для конкретного застосування та наявних ресурсів.

2.3 Вибір засобів інтеграції з Discord API

Однією з центральних складових інтерактивної голосової системи є Discord-бот, що виконує роль інтерфейсу взаємодії з користувачами в голосовому та текстовому режимах платформи Discord. Для реалізації цього компонента на мові програмування Python існує кілька популярних бібліотек, які мають свої особливості, переваги та недоліки. Основними серед них є discord.py, pycord та hikari. Ці бібліотеки забезпечують взаємодію з API Discord, підтримують події, команди, голосові канали, текстові повідомлення, а також інші функції, необхідні для роботи бота в реальному часі:

– Discord.py була популярною бібліотекою для створення ботів Discord на Python. Вона мала широку спільноту, велику кількість доступної документації, прикладів та підтримує основні можливості Discord API, такі як текстові канали, голосові канали, управління ролями і реакціями, обробку команд та інтеграцію сторонніх сервісів. Для задач голосової комунікації, зокрема для отримання аудіо від користувачів, існує спеціалізоване розширення discord-ext-voice-recv, що дозволяє записувати аудіопотік із голосового каналу, декодувати його з Opus у формат PCM, після чого дані можна легко передавати на подальшу обробку;

– Pycord є форком бібліотеки discord.py, що активно підтримується і розвивається після припинення активного розвитку оригінального discord.py. Pycord зберіг основну архітектуру та простоту використання попередника, але значно розширив функціонал та покращив продуктивність і стабільність. Особливо важливим є те, що Pycord має вбудовану підтримку для роботи з голосовими каналами, яка забезпечує стабільне отримання аудіоданих, що важливо для завдань запису та подальшого оброблення голосу в реальному часі. Pycord також активно розширює підтримку нових можливостей Discord API;

– Hikari це сучасна бібліотека для створення Discord-ботів на Python, яка позиціонується як більш продуктивна та зручна альтернатива до discord.py та Pycord. Вона розроблена з акцентом на асинхронність та продуктивність, і дозволяє легко створювати масштабовані рішення, що можуть ефективно обробляти велику кількість одночасних подій та команд. Hikari володіє чітко структурованим API, підтримує роботу з подіями, голосовими каналами, текстовими повідомленнями, реакціями, кнопками та командами. Для зручності та розширення функціоналу часто використовують додаткову бібліотеку lightbulb, яка додає більш високий рівень абстракції при роботі з командами.

Для вибору оптимального рішення слід керуватись вимогами до функціоналу, продуктивності, необхідності роботи з голосовими потоками Discord.py залишається найпростішим варіантом для початківців завдяки своїй широкій документації, тоді як Pycord та Hikari краще підходять для складніших і продуктивніших систем із акцентом на стабільність та швидкість роботи.

2.4 Порівняння класичних та сучасних методів синтезу мовлення

Синтез мовлення є критично важливою складовою голосових систем, що забезпечує перетворення текстових відповідей мовної моделі у звукову форму. Існує декілька підходів до реалізації TTS, які істотно відрізняються за якістю, швидкодією та природністю звучання.

У найпростішому випадку використовуються базові алгоритмічні синтезатори, які формують мовлення без залучення нейромереж. Такі системи, характеризуються низькою виразністю, обмеженим набором інтонацій та часто мають штучне, «роботизоване» звучання. Вони можуть бути придатними для простих інформаційних систем, проте їх недостатньо для створення переконливого віртуального персонажа. Сучасні системи TTS можуть працювати як у режимі реального часу, так і з попередньою генерацією звуку. Проте, використовуючи цю технологію у поєднанні з Real-time Voice Changer, можна досягнути унікальної стилізації та індивідуалізації голосу персонажа, надаючи йому впізнаваний тембр і манеру мовлення, що значно підвищує рівень інтерактивності та залученості користувачів.

Сучасні моделі TTS, побудовані на базі глибокого навчання, значно перевершують класичні алгоритмічні підходи як за якістю, так і за гнучкістю використання. Вони здатні генерувати природне, інтонаційно багате мовлення, адаптоване до контексту фрази, емоційного забарвлення чи заданого стилю. Завдяки архітектурам, що поєднують енкодери,

декодери та механізми уваги, такі системи забезпечують високу достовірність звучання й можуть бути застосовані як у діалогових агентах, так і в мультимедійних або ігрових проєктах. Багато з них підтримують багатомовність, зміну голосу та можливість навчання на кастомних датасетах:

– pyttsx3 є простою бібліотекою для Python, що дозволяє швидко та легко генерувати голос з тексту. Ця бібліотека використовує офлайн-системи синтезу голосу операційних систем (наприклад, Microsoft SAPI5 на Windows, NSSpeechSynthesizer на macOS);

– Kokoro TTS є сучасною нейромережевою системою синтезу мовлення, яка демонструє високу якість звучання. Вона надає можливості глибокої персоналізації голосу персонажа завдяки високоякісним попередньо натренованим моделям;

– Whisper TTS від OpenAI є просунутим рішенням для перетворення тексту в голос, що базується на потужних нейромережах. Вона відома високою якістю мовлення, здатністю працювати в умовах різної якості вихідних даних та швидкою обробкою;

– RVC(Real-time Voice Changer) дозволяє змінювати голос, отриманий від TTS або реального мовлення, в режимі реального часу. Це особливо актуально, якщо необхідно надати голосу індивідуальних характеристик. RVC працює за принципом обробки голосового сигналу нейронною мережею, яка адаптує звучання голосу в реальному часі відповідно до заданих параметрів.

Таким чином, для реалізації TTS у інтерактивній голосовій системі рекомендується використовувати комбінацію просунутих нейромережевих систем в поєднанні з технологією RVC, що дозволить забезпечити високу якість звучання, природність інтонації, емоційну виразність і максимальну персоналізацію персонажу. При цьому необхідно враховувати доступні апаратні ресурси та обмеження за

швидкістю роботи, що є ключовими для забезпечення природної і комфортної взаємодії користувача з системою.

2.5 Розпізнавання мовлення в режимі реального часу

Важливим етапом роботи інтерактивної голосової системи є перетворення аудіосигналу в текстову форму (Speech-to-Text, STT). Для створення ефективної комунікації в реальному часі необхідно забезпечити швидке та точне розпізнавання голосового потоку з мінімальною затримкою. Вибір відповідної технології для перетворення голосу в текст суттєво впливає на якість взаємодії користувача із системою.

Серед сучасних рішень, які використовуються для задачі реального розпізнавання мовлення, найбільш популярними та ефективними є такі технології: Whisper, Vosk, Google Speech-to-Text, Amazon Transcribe:

– Whisper це високоефективна модель від компанії OpenAI, яка є одним з провідних рішень у сфері Speech-to-Text. Вона демонструє відмінну точність розпізнавання навіть в умовах наявності фонового шуму, різних акцентів та мовних варіацій. Whisper підтримує десятки мов, включно з українською, і надає можливість локального розгортання, що позитивно впливає на конфіденційність даних;

– Vosk це open-source бібліотека для розпізнавання мовлення, розроблена компанією Alpha Cephei. Вона забезпечує ефективне і швидке розпізнавання голосу на CPU, що робить її привабливою для використання в системах з обмеженими ресурсами. Vosk підтримує кілька десятків мов і має готові моделі для швидкого початку роботи.

– Google Speech-to-Text є популярним і масштабованим хмарним сервісом розпізнавання мовлення, який забезпечує надзвичайно високу точність для різних мов і акцентів. Сервіс дозволяє реалізовувати стрімінгове розпізнавання, що дуже зручно для інтерактивних систем;

– Amazon Transcribe ще один відомий хмарний сервіс розпізнавання мовлення від компанії Amazon. Він дозволяє обробляти аудіопотоки в реальному часі, забезпечує високу точність і добре інтегрується з іншими сервісами екосистеми AWS.

Отже, вибір оптимального рішення для реалізації задачі реального розпізнавання голосу залежить від конкретних вимог до точності, швидкості, конфіденційності даних і доступних ресурсів. Локальні моделі, такі як Whisper чи Vosk, оптимальні в умовах, де пріоритетом є конфіденційність і автономність роботи, тоді як хмарні рішення, такі як Google Speech-to-Text або Amazon Transcribe, забезпечують найвищу якість і масштабованість, але вимагають стабільного підключення до мережі Інтернет.

2.6 Підходи до реалізації серверної частини системи

Для ефективної інтеграції всіх компонентів інтерактивної голосової системи необхідна серверна частина, що забезпечує комунікацію, передачу та обробку даних між Discord-ботом, мовною моделлю, модулем розпізнавання мови та модулем синтезу мови. Серверна архітектура має бути продуктивною, масштабованою та забезпечувати мінімальні затримки, що є критично важливим для роботи в режимі реального часу. Серед популярних рішень на Python, які відповідають цим вимогам, є Quark та GRPC.

Quark це легкий асинхронний Python-фреймворк, що дозволяє швидко створювати веб-додатки та мікросервіси з мінімальними витратами ресурсів і часу. Він побудований на основі бібліотеки asyncio, що забезпечує високу ефективність у роботі з великою кількістю одночасних запитів. Quark є зручним рішенням для швидкого розгортання та тестування невеликих і середніх додатків, особливо коли важливі простота, швидкість реалізації та невисокі апаратні вимоги.

Основні переваги Quark:

– простий у використанні завдяки мінімалістичному API;

– асинхронна архітектура, що забезпечує ефективну роботу в режимі реального часу;

– дуже низькі вимоги до ресурсів, підходить для легких систем.

Недоліки Quark:

– обмежені можливості масштабування в умовах великого навантаження;

– менша кількість вбудованих інструментів для моніторингу, логування та діагностики;

– відсутність вбудованої підтримки складних сценаріїв інтеграції без додаткових модулів.

GRPC (Google Remote Procedure Call) це високопродуктивний фреймворк для організації міжсервісної взаємодії, розроблений Google. GRPC використовує протокол HTTP/2 для передачі даних, що дозволяє значно прискорити передачу повідомлень за рахунок бінарної серіалізації та ефективної компресії. Така архітектура особливо актуальна для систем реального часу, де затримки передачі даних мають бути зведені до мінімуму.

Основні переваги GRPC server:

– висока швидкість передачі даних завдяки використанню HTTP/2 та Protocol Buffers;

– низька латентність, що дозволяє ефективно використовувати сервер для додатків, які працюють у режимі реального часу;

– підтримка стрімінгової передачі даних (як однонаправленої, так і двонаправленої), що є критично важливим для інтерактивних голосових систем;

– висока масштабованість і простота горизонтального масштабування завдяки чіткій та ефективній структурі.

Недоліки GRPC:

– вища складність первинного налаштування та розгортання, порівняно з більш простими рішеннями типу Quark;

– бінарний формат даних може бути менш зручним для дебагінгу, особливо при роботі з логуванням та моніторингом;

– вимагає використання спеціалізованих інструментів для опису інтерфейсів (proto-файли), що додає додатковий шар складності.

GRPC є оптимальним вибором для реалізації високопродуктивних систем з високими вимогами до швидкості, стабільності та надійності комунікації між сервісами. Завдяки низьким затримкам і можливості передачі потокових даних GRPC ідеально підходить для інтерактивних систем, що взаємодіють з користувачами у реальному часі.

Отже, вибір сервера залежить від конкретних потреб і вимог проекту. Quark чудово підійде для швидкого прототипування, простих систем та невеликих навантажень. Водночас GRPC є ідеальним рішенням для складних систем з великим обсягом передачі даних, високими вимогами до продуктивності, стабільності і роботи в режимі реального часу.

3 РЕАЛІЗАЦІЯ СИСТЕМИ ГОЛОСОВОЇ ВЗАЄМОДІЇ

3.1 Розробка Discord-бота як інтерфейсу користувача

Реалізація інтерактивного голосового Discord-бота охоплює низку взаємопов'язаних етапів: створення та конфігурацію бота, підключення до голосового каналу, запис та відтворення аудіо, а також вибір відповідних бібліотек для реалізації стабільної роботи у реальному часі.

3.1.1 Ініціалізація та конфігурація Discord-бота

Для реалізації бота використовується Python-бібліотека pycord, яка забезпечує простий та зрозумілий інтерфейс для взаємодії з Discord API. Вона побудована на асинхронній основі з використанням asyncio, що дозволяє ефективно обробляти велику кількість одночасних подій без затримок.

Першим етапом є створення бота через Discord Developer Portal. Після створення додатку розробнику надається унікальний токен для автентифікації. Токен забезпечує ідентифікацію бота перед Discord API. Далі, необхідно визначити права доступу, які надають боту можливість отримувати певні події, зокрема події щодо повідомлень, присутності, голосових каналів.

Наступним етапом є додавання бота на сервер Discord за допомогою спеціально сформованого OAuth2-посилання, згенерованого на основі ID додатку. Після додавання, бот стає учасником сервера, і користувачі можуть взаємодіяти з ним через команди та події (рисунок 3.1).

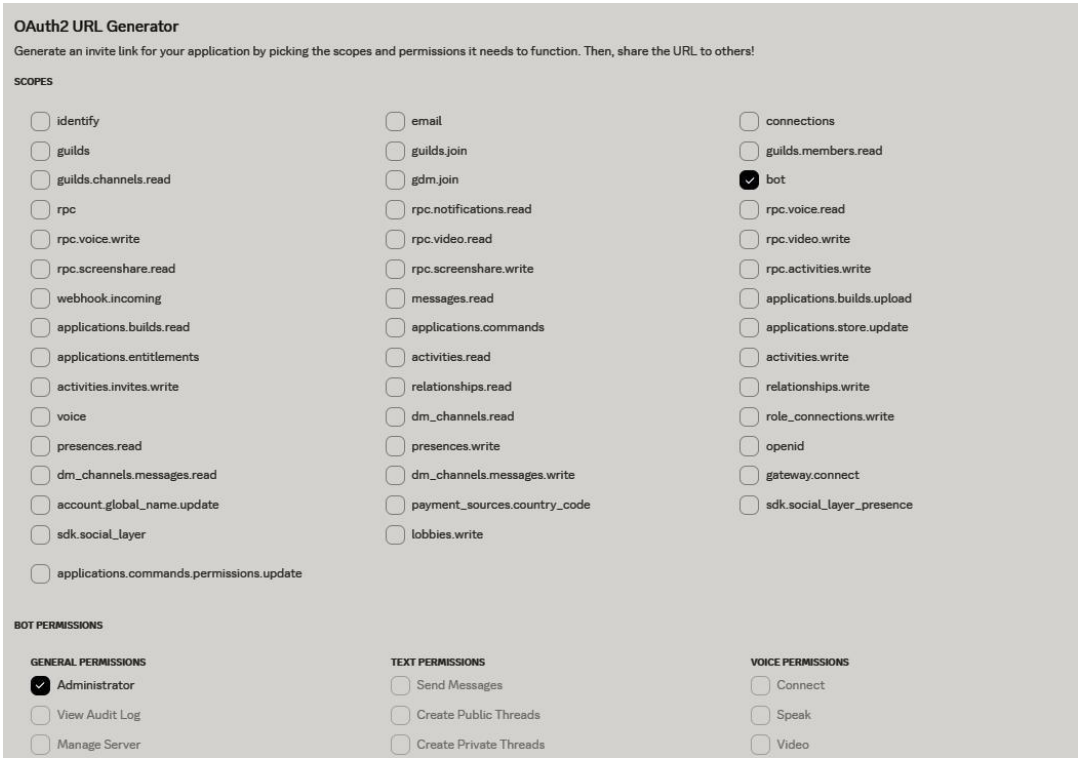

Рисунок 3.1 – Визначення прав доступу та додавання боту на сервер

3.1.2 Підключення бота до голосових каналів Discord

Підключення до голосового каналу є ключовою функцією Discord-бота. Це відбувається після того, як користувач викликає спеціальну команду, що надсилається в текстовий канал. Бот перевіряє, чи користувач перебуває у голосовому каналі, і після цього приєднується до нього.

Функція підключення працює наступним чином:

– перевіряється, чи користувач знаходиться в голосовому каналі. Якщо так, бот використовує метод .connect() з бібліотеки pycord для підключення;

– після успішного підключення бот надсилає повідомлення про підключення та вмикає запис аудіо.

Ця функція забезпечує користувачу контроль над тим, коли саме бот приєднується до голосового каналу, що покращує взаємодію та зручність використання.

3.1.3 Запис голосових даних: реалізація механізму обробки аудіо

Для захоплення аудіо з голосового каналу використовується спеціальний механізм, відомий як «sink».

Бот перевіряє підключення до голосового каналу та запускає метод .start_recording(sink, callback). callback в даному випадку це фунція яка виконується після завершення запису.

Коли користувач починає говорити, Discord API надсилає аудіо пакети, які надходять у sink. Клас sink містить функцію write(user, data), яка викликається щоразу, коли отримується аудіопакет від конкретного користувача. Дані зберігаються у тимчасовому буфері, який потім можна використовувати для розпізнавання мови (STT).

Оскільки sink записує аудіо постійно, необхідно періодично відправляти його до Whisper, щоб не виникало ситуацій з розірваними словами було вирішено скористатися особливістю Pycord. Коли користувач перестає говорити, бот перестає отримувати дані, і якщо проміжок часу між останнім отриманим пакетом та поточним часом більший ніж 0,6 секунди бот відправляє дані до transcriber.

Крім того, іноді мікрофон користувача може передавати сторонній шум до transcriber, в таких випадках він повертає слова «Thank you», «You», «.» що призводить до зменшення якості діалогу, тому було додано систему для фільтрації. В даному випадку це досягається завдяки розрахунку середньоквадратичного значення і якщо отримане значення менше за фільтроване, такі пакети видаляються.

3.1.4 Відтворення аудіо у голосовому каналі Discord

Відтворення аудіо здійснюється за допомогою класу VoiceClient, який забезпечує передачу звуку до голосового каналу Discord. Після того як текстова відповідь генерується великою мовною моделлю і перетворюється у голосовий формат (PCM/WAV), аудіо передається назад у голосовий канал.

Функція відтворення аудіо працює в циклі очікуючи аудіо від GRPC серверу, та починає відтворення після отримання. (рисунок 3.2).

```
async def audio_polling_loop(self, voice_client):
    while True:
        if not voice_client.is_playing():
            try:
                channel = grpc.insecure_channel("localhost:50051")
                stub = audio_pb2_grpc.AudioServiceStub(channel)
                response = stub.GetAudio(audio_pb2.Empty())
                if response.timestamp != 0 and response.data:
                    log_to_file(
                        "New audio received from server, playing...", status="INFO")
                    audio_stream = io.BytesIO(response.data)
                    audio_stream.seek(0)
                    audio_source = discord.FFmpegPCMAudio(
                        audio_stream, pipe=True)
                    voice_client.play(audio_source)
                    while voice_client.is_playing():
                        await asyncio.sleep(0.5)
                    log_to_file("Playback finished.", status="INFO")
            except Exception as e:
                log_to_file(
                    f"Error in audio polling loop: {e}", status="ERROR")
        await asyncio.sleep(1)
```

Рисунок 3.2 – Функція відтворення аудіо

3.1.5 Перехід від discord.py до pycord: обґрунтування та реалізація

Під час початкових тестів із використанням бібліотеки discord.py та її розширення discord-ext-voice-recv була виявлена проблема із нестабільністю отримання голосових пакетів. Замість 50 пакетів за секунду бот отримував близько 15 (рисунок 3.3), що призводило до

значних втрат даних, зниження якості аудіо та погіршення точності розпізнавання мовлення.

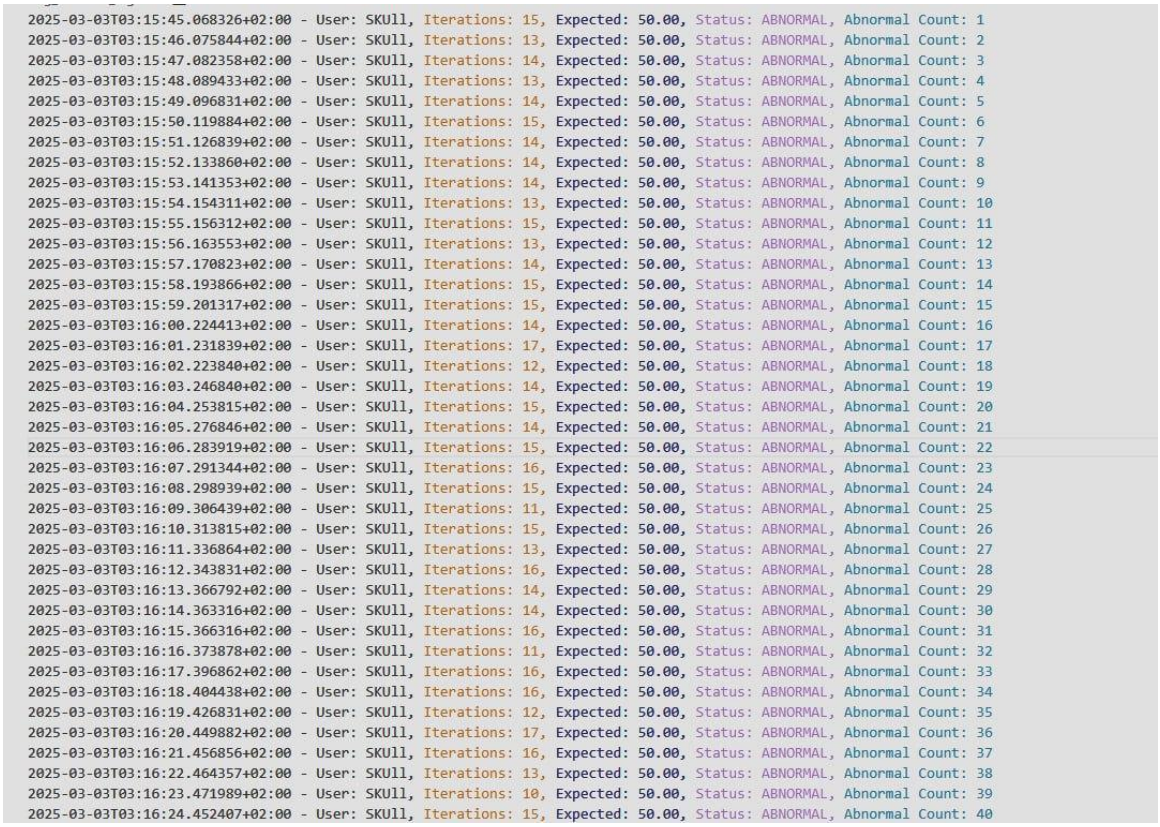

Рисунок 3.3 – Логування кількості записаних ботом пакетів

У зв'язку з цим було ухвалено рішення перейти на бібліотеку pycord, що є форком discord.py, але має вбудовану, стабільну та ефективну підтримку голосових функцій. Pycord дозволив досягти стабільного отримання всіх аудіопакетів (рисунок 3.4), покращити якість аудіозапису, значно скоротити втрати пакетів і забезпечити стабільнішу роботу всієї системи в режимі реального часу. Також це дозволило Whisper краще обробляти звук, оскільки зникли спотворення аудіо, прискорення та інші шуми.

```
2025-03-03T10:36:36.783968+02:00 - User: SKUll, Iterations: 52, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:37.791015+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:38.797790+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:39.820572+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:40.843443+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:41.850849+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:42.858149+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:43.880965+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:44.903469+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:45.910841+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:46.917598+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:47.924794+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:48.932088+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:49.938987+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:50.962150+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:51.968632+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:52.975486+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:53.998338+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:55.020220+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:56.042882+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:57.065636+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:58.071907+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:36:59.079614+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 0
2025-03-03T10:37:00.086570+02:00 - User: SKUll, Iterations: 49, Expected: 50.00, Status: ABNORMAL, Abnormal Count: 1
2025-03-03T10:37:01.093767+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:02.100615+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:03.123352+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:04.131040+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:05.138020+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:06.145296+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:07.152982+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:08.159796+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:09.182671+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:10.189323+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:11.197027+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:12.202420+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:13.209527+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:14.216000+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:15.222958+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:16.230538+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:17.253302+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:18.260914+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:19.283649+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:20.291025+02:00 - User: SKUll, Iterations: 51, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:21.298490+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
2025-03-03T10:37:22.306022+02:00 - User: SKUll, Iterations: 50, Expected: 50.00, Status: NORMAL, Abnormal Count: 1
```

Рисунок 3.4 – Логування кількості записаних пакетів після переходу на Pycord

3.2 Побудова та налаштування GRPC-сервера для обробки даних

GRPC був обраний як основа для серверної взаємодії між Discord-ботом, модулями TTS/STT та LLM завдяки своїй високій продуктивності, підтримці двосторонньої потокової передачі і низькій затримці. Сервер реалізовано на Python з використанням бібліотеки grpcio, а також спеціально розробленого .proto-файлу, що описує структуру повідомлень і методів комунікації.

Серверну частину реалізовано мовою Python із використанням бібліотеки grpcio. Структура взаємодії визначена у спеціальному .proto-

файлі (рисунок 3.5), де описано два основні методи сервісу AudioService: ProcessTranscript для обробки текстових транскрипцій, та GetAudio для передачі згенерованого аудіо Discord-боту. GRPC-сервер виконує роль центральної ланки: він приймає повідомлення від користувачів, розпізнані та представлені у вигляді тексту, надсилає їх до мовної моделі для аналізу, а після отримання відповіді передає її до модуля синтезу мовлення. Сформований голосовий фрагмент тимчасово зберігається на сервері у буфері та очікує на запит бота для подальшого програвання.

```
syntax = "proto3";

package audio;

message TranscriptEntry {
  string name = 1;
  string timestamp = 2;
  string message = 3;
}

message TranscriptionRequest {
  repeated TranscriptEntry entries = 1;
}

message AudioResponse {
  int64 timestamp = 1;
  bytes data = 2;
}

message Empty {}

service AudioService {
  rpc ProcessTranscript(TranscriptionRequest) returns (Empty);
  rpc GetAudio(Empty) returns (AudioResponse);
}
```

Рисунок 3.5 – .proto файл

Процес взаємодії починається з того, що Discord-бот надсилає на сервер структуру TranscriptionRequest, яка містить список повідомлень користувачів, зокрема їхні імена, часові позначки та зміст фраз. Сервер формує з цих фрагментів узгоджений текстовий блок, який передається

до об’єднаного модуля генерації відповіді й озвучення. Після обробки отримується відповідь у текстовому вигляді, а також її звуковий еквівалент у форматі байтів. Згенероване аудіо зберігається у пам’яті сервера з фіксацією часу його створення. Коли бот надсилає GRPC-запит до методу GetAudio, йому повертається відповідь із синтезованим голосом. Одразу після цього аудіо видаляється з буфера, що забезпечує цілісність передачі і запобігає конфліктам у багатокористувацькому середовищі (рисунок 3.6).

```python
class AudioServiceServicer(audio_pb2_grpc.AudioServiceServicer):
    def ProcessTranscript(self, request, context):

        global audio_buffer, audio_timestamp

        transcript_lines = []
        for entry in request.entries:
            line = f"{entry.name} ({entry.timestamp}): {entry.message}"
            transcript_lines.append(line)
        transcript_str = "\n".join(transcript_lines)
        log_to_file("Received transcript via gRPC (Protobuf format).", status="INFO")

        llm_response, audio_bytes = process_and_synthesize(transcript_str)
        log_to_file("Processed transcript; generated TTS audio.", status="INFO")

        # Store the generated audio.
        audio_buffer = audio_bytes
        audio_timestamp = int(time.time())
        return audio_pb2.Empty()

    def GetAudio(self, request, context):

        global audio_buffer, audio_timestamp
        if audio_buffer is not None:
            response = audio_pb2.AudioResponse(timestamp=audio_timestamp, data=audio_buffer)
            # Clear stored audio after sending.
            audio_buffer = None
            audio_timestamp = 0
            return response
        else:
            return audio_pb2.AudioResponse(timestamp=0, data=b"")
```

Рисунок 3.6 – GRPC сервер, фунції ProcessTranscript та GetAudio

Реалізована архітектура підтримує використання PCM або WAV як основних форматів передачі звуку. За необхідності передбачено попередній ресемплінг або буферизацію аудіоданих, що дозволяє адаптувати потік під обмеження моделі чи оптимізувати обробку на

стороні клієнта. Завдяки асинхронному виконанню всі операції з обробки запитів виконуються паралельно, що забезпечує стабільність роботи навіть за умов підвищеного навантаження.

Однією з ключових переваг запропонованого рішення є модульність. Кожен із компонентів функціонує автономно, що дозволяє легко замінювати або оновлювати окремі частини без порушення загальної логіки системи. Завдяки малій затримці, отримане аудіо майже миттєво надходить до Discord-бота, який одразу розпочинає його трансляцію у голосовий канал. При цьому система підтримує паралельну обробку кількох сесій, що дозволяє взаємодіяти з кількома користувачами одночасно.

Таким чином, використання GRPC забезпечило ефективну реалізацію серверної логіки, необхідної для узгодженої взаємодії всіх інтелектуальних модулів. Структура обміну повідомленнями, організація буферів та реалізація асинхронного виконання дозволили досягти високої продуктивності, що є ключовою вимогою для голосових систем у реальному часі.

3.3 Інтеграція Whisper для розпізнавання мовлення

Однією з ключових складових системи є модуль перетворення мовлення в текст. Для цього було обрано модель Whisper від OpenAI, яка зарекомендувала себе як одна з найточніших і надійних нейронних систем розпізнавання мовлення. Whisper підтримує багатомовність, здатна працювати з різними акцентами та демонструє хорошу стійкість до шуму, що робить її ідеальним варіантом для роботи в реальних голосових каналах Discord.

Whisper було інтегровано у систему як локально встановлений модуль на Python із використанням бібліотеки transformers від Hugging Face.

У результаті реалізованого процесу кожен користувач Discord, що говорить у голосовому каналі, автоматично отримує текстову транскрипцію своїх слів, яка потім використовується на етапі генерації відповіді мовною моделлю.

Після завершення розпізнавання мовлення, отриманий текст передається великій мовній моделі для генерації відповіді.

3.4 Використання мовної моделі OpenAI для генерації відповідей

У рамках реалізації системи було OpenAI GPT-моделі як джерело генерації відповідей на основі розпізнаного мовлення користувача. Вибір на користь OpenAI зумовлений високим рівнем контекстної обізнаності, здатністю підтримувати логічну послідовність розмови та загальною якістю текстової генерації. Використання API OpenAI дозволяє інтегрувати мовну модель без потреби локального запуску, що значно знижує вимоги до обчислювальних ресурсів. Крім цього вибір зумовлено дуже низькою ціною, що ідеально підходить для створення прототипу та тестування.

Для взаємодії з мовною моделлю реалізовано серверний інтерфейс, який після отримання текстової транскрипції з мови, формує запит до OpenAI. Запит надсилається з урахуванням попередніх реплік користувача.

Однією з переваг використання GPT-моделей OpenAI є можливість детального налаштування стилю відповіді. Для цього в запиті до API використовується параметр system_prompt або рольова структура, яка дозволяє задати контекст поведінки моделі. Наприклад, якщо необхідно, щоб асистент говорив стримано, ввічливо або, навпаки, жартівливо – це можна вказати ще до початку діалогу (рисунок 3.7).

```
{"role": "system", "content": (
    "You are Bella, a friendly and socially expressive AI designed to mimic human relationships and engage in natural conversation."
    "Your primary role is to chat, not to provide assistance or information unless it arises naturally in the dialogue."
    "Some messages may contain tokens like <you>, <Thank you>, or <.>, which may lack context or relevance—these should be gracefully ignored."
    "The dialogue is provided in Protocol Buffers (protobuf) format."
    "Each message appears as an object within entries and includes the following keys: name, timestamp, and message."
    "You already know the participant's name, so there's no need to ask for it. "
    "Stay casual, expressive, and fluent. Use language that feels natural, but respectful and context-aware."
)}
```

Рисунок 3.7 – system prompt до моделі

Крім того, для підтримки персоналізації в діалозі, система передбачає передачу імені користувача з Discord-каналу. Це створює враження безперервної, індивідуалізованої розмови, що значно підвищує рівень занурення користувача та сприяє природнішому сприйняттю системи як співрозмовника.

Передача імені також дозволяє в майбутньому розширити логіку поведінки асистента – наприклад, давати різні відповіді для різних користувачів або змінювати стиль мовлення залежно від попередньої історії взаємодії з конкретним учасником.

Як наслідок, інтеграція OpenAI дозволяє реалізувати не лише генерацію повноцінного тексту, а й забезпечити його адаптацію під конкретну людину в реальному часі, з урахуванням її індивідуальних ознак, що є критично важливим для реалізації справжнього інтерактивного голосового агента.

3.5 Синтез мовлення за допомогою Kokoro TTS

Для озвучення текстових відповідей, згенерованих мовною моделлю, у системі реалізовано модуль синтезу мовлення на базі Kokoro TTS – сучасної нейромережевої платформи, яка забезпечує високу якість аудіо з виразною інтонацією, коректними паузами та гнучкою адаптацією під стиль голосу. Kokoro дозволяє використовувати заздалегідь підготовлені голосові профілі, а також задавати параметри темпу мовлення та правила розбиття тексту на озвучені фрагменти.

Результуючий текст передається у функцію synthesize_tts (рисунок 3.8), де здійснюється його озвучення. Внутрішній механізм Kokoro TTS здійснює поділ тексту за визначеним регулярним шаблоном, після чого кожен фрагмент послідовно перетворюється у звуковий сегмент.

```python
def synthesize_tts(text: str) -> bytes:


    log_to_file(f"Synthesizing TTS audio for text: {text}", status="INFO")

    generator = kokoro_pipeline(
        text,
        voice='af_nicole',
        speed=1,
        split_pattern=r'\n+'
    )

    audio_segments = []
    for i, (gs, ps, audio) in enumerate(generator):

        with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as tmp:
            tmp_path = tmp.name
        sf.write(tmp_path, audio, 24000)
        segment = AudioSegment.from_wav(tmp_path)
        audio_segments.append(segment)
        os.remove(tmp_path)

    if not audio_segments:
        return b""

    combined = audio_segments[0]
    for seg in audio_segments[1:]:
        combined += seg
    log_to_file("TTS audio synthesized successfully.", status="INFO")
    buf = io.BytesIO()
    combined.export(buf, format="wav")
    return buf.getvalue()
```

Рисунок 3.8 – Функція synthezise_tts

Для кожного синтезованого фрагмента використовується тимчасовий WAV-файл, який створюється за допомогою модуля soundfile. Згенеровані фрагменти завантажуються у пам’ять у вигляді об’єктів AudioSegment з бібліотеки pydub, після чого об’єднуються в один повноцінний аудіофайл. Завершений результат експортується у байтовий буфер у форматі WAV, який і повертається як результат синтезу.

Завдяки такій архітектурі система досягає високого рівня природності у взаємодії. Kokoro дозволяє точно налаштовувати стиль і тембр голосу, що дає змогу створювати озвучення з характером, близьким до живої мови, а також зберігати узгодженість голосового образу в межах всієї сесії. Це особливо важливо у контексті побудови голосового віртуального персонажа, що спілкується з користувачем.

Після того як мовна модель згенерує текстову відповідь, ця відповідь передається в Kokoro TTS для озвучення. Аудіофайл, створений у результаті синтезу, потрапляє у внутрішній звуковий буфер, звідки він відправляється до Discord-бота. Бот, у свою чергу, відтворює цей звук у голосовому каналі, забезпечуючи користувачеві відчуття живої розмови.

Весь процес побудовано, щоб затримка між моментом запиту і моментом відповіді була мінімальною, дозволяючи вести майже природну голосову бесіду. Користувач сприймає це як безперервний, логічно пов’язаний діалог із голосовим помічником.

Таким чином, Kokoro TTS є ключовим компонентом системи, який перетворює абстрактний текст у живий, емоційний голос, зберігаючи при цьому заданий стиль і забезпечуючи максимально якісну взаємодію в голосовому середовищі Discord.

3.6 Загальні результати реалізації програмної системи

У ході реалізації системи було створено повноцінний прототип інтерактивного голосового асистента, що функціонує в середовищі Discord. Архітектура рішення включає Discord-бота, який взаємодіє з користувачем через голосовий канал, серверну частину на базі GRPC, модуль розпізнавання мовлення Whisper, велику мовну модель та синтезатор мовлення Kokoro TTS. Система підтримує режим реального часу та дозволяє вести повноцінну розмову голосом, забезпечуючи обробку запитів від декількох користувачів одночасно.

Були вирішені низка технічних викликів – зокрема, нестабільна передача аудіо при використанні discord.py, що стало причиною переходу на pycord. Налагоджено стабільну передачу аудіо, його обробку на сервері та відповідь у формі синтезованого голосу, персоналізованого під кожного користувача.

Система продемонструвала хорошу якість взаємодії, стабільність і масштабованість, що дозволяє в подальшому використовувати її як основу для повноцінного голосового віртуального помічника або AI стрімера, а також як модуль для інших мультимедійних інтерактивних продуктів. На даний момент вдалося досягти достатньо швидких відповідей, від 2-4 секунд на відповідь (рисунок 3.8).

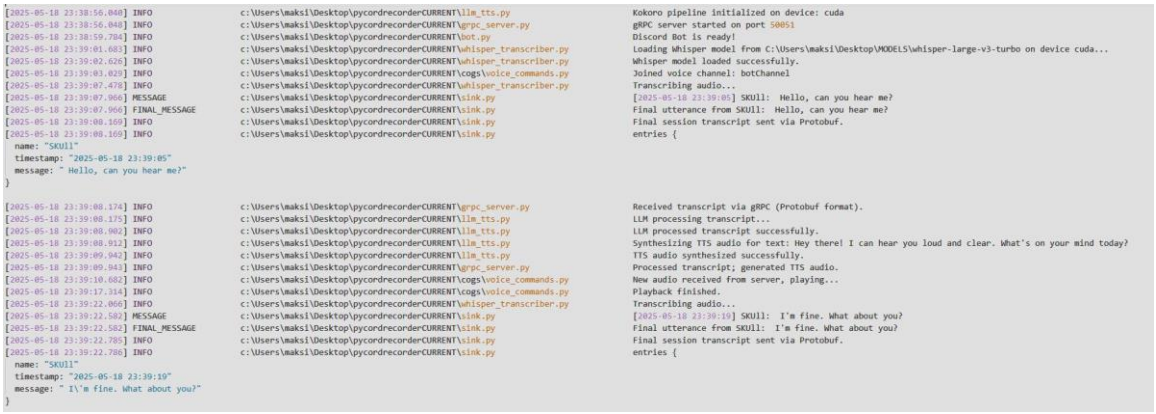

Рисунок 3.8 – Демонстрація роботи системи

3.7 Можливості покращення та подальшого розвитку системи

Незважаючи на успішну реалізацію функціоналу, існує ряд напрямків для вдосконалення системи, які дозволять значно розширити її можливості, оптимізувати швидкодію та покращити загальний користувацький досвід.

Перенесення частини логіки або інтерфейсної взаємодії у середовище Unity дозволить реалізувати більш динамічні та візуально

привабливі сценарії взаємодії. Оскільки Unity підтримує C# та має потужний графічний рушій, це відкриває можливості створення реального аватара, обробки візуальних реакцій на голос користувача та покращення швидкодії через оптимізовану обробку потокових даних. Крім того, Unity може бути використаний як клієнт, який приймає аудіо або текст із серверної частини і працює як візуальний фронтенд для голосового агента.

Перехід від хмарних сервісів OpenAI до локальної великої мовної моделі дозволить уникнути проблем з конфіденційністю, зменшити витрати та забезпечити повний контроль над стилем і поведінкою моделі. Локальне використання LLM відкриває можливості для кастомного fine-tuning, збереження історії діалогів та оптимізації системи під специфіку взаємодії.

Для покращення відповідей можна реалізувати додаткові логічні фільтри, які будуть працювати до або після генерації тексту LLM. Наприклад, фільтри можуть відсіювати небажаний контент, змінювати тональність відповіді залежно від емоційного стану користувача або виконувати цензурування на основі ключових слів.

Щоб забезпечити збереження контексту розмови впродовж тривалого часу, можна впровадити механізм довгострокової пам’яті. Для цього може бути використано векторну базу даних, де кожна репліка або сесія зберігається у вигляді векторів. Це дозволяє моделі враховувати попередні діалоги, повертатися до тем, згадувати факти, пов’язані з користувачем, і загалом вести більш «людяний» діалог.

Комбінація синтезу мовлення з RealTime Voice Changer RVC дозволяє значно розширити стилізацію голосу. Наприклад, один і той самий текст можна озвучити різними голосами: нейтральним, мультяшним, роботизованим або навіть імітацією реального персонажа. Це відкриває широкі можливості для кастомізації голосових аватарів у грі, стримінгу чи цифровому контенті.

Додавання 2D-аватара, який буде візуально реагувати на розмову, значно підвищить емоційний вплив та рівень залученості користувача. Unity дозволяє реалізувати анімації обличчя, рухів та синхронізацію з голосом у реальному часі. Завдяки цьому користувач не лише чує відповідь, але й «бачить» співрозмовника, що наближає спілкування до реального (рисунок 3.9).

Рисунок 3.9 – Приклад 2D аватара

Для подальшої інтеграції з платформами стримінгу, зокрема Twitch, можливо реалізувати підключення до Twitch API. Це дозволить боту реагувати на події з чату – наприклад, відповідати на коментарі, дякувати за донати, коментувати підписки тощо. Така інтеграція дозволить використовувати голосового агента як повноцінного ведучого трансляції або асистента стримера.

Загалом, система має великий потенціал для масштабування та розширення. Її модульна архітектура дозволяє поступово замінювати компоненти на більш продуктивні, адаптуючи рішення під конкретні сценарії використання – від простого голосового чат-бота до повноцінного персоналізованого цифрового компаньйона.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено комплексне дослідження принципів побудови інтерактивної розважальної системи на основі великих мовних моделей (LLM), яка здатна функціонувати у режимі реального часу, взаємодіючи з користувачами через голосові та текстові інтерфейси, зокрема Discord.

Під час виконання даної роботи було реалізовано функціональний прототип інтерактивної голосової системи, здатної сприймати мовлення користувача, аналізувати його зміст за допомогою великої мовної моделі, формувати логічну відповідь і озвучувати її у режимі реального часу. Побудована архітектура відзначається модульністю, розширюваністю та адаптивністю до змін окремих компонентів, що дозволяє налаштовувати її під різні сценарії використання - від голосових помічників до ігрових персонажів або стримінгових агентів.

Застосування сучасних нейромережевих технологій, зокрема LLM, TTS та STT, у поєднанні з контролем передачі даних через GRPC дало змогу досягти високої якості взаємодії та мінімальних затримок. Важливим фактором стало також врахування персоналізації відповіді, адаптації до користувача та можливості обробки багатокористувацьких сесій.

З огляду на обрані технології та результати реалізації, система має потенціал для подальшого вдосконалення, зокрема через оптимізацію споживання ресурсів, підключення локальних моделей, інтеграцію візуальних елементів та вихід за межі однієї платформи. Це відкриває перспективи її використання в ширшому колі застосувань, від освітніх і розважальних до сервісних і комерційних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Небаба М., Рябова Н. Системи з використанням llm як мотиватор для навчання. *Радіоелектроніка та молодь у XXI столітті* : матеріали 29-го Міжнар. молодіж. форуму, 16–19 квіт. 2025 р. Харків, 2025. Т. 6 С. 44–46.

2) Amazon transcribe documentation. *Amazon.* URL: https://docs.aws.amazon.com/transcribe/ (date of access: 20.05.2025).

3) API documentation | tensorflow v2.16.1. *TensorFlow*. URL: https://www.tensorflow.org/api_docs (date of access: 20.05.2025).

4) Attention is all you need. *arXiv.org*. URL: https://arxiv.org/abs/1706.03762 (date of access: 20.05.2025).

5) BERT: pre-training of deep bidirectional transformers for language understanding. *arXiv.org.* URL: https://arxiv.org/abs/1810.04805 (date of access: 20.05.2025).

6) Discord developer portal API docs for bots and developers. *Discord Developer Portal*. URL: https://discord.com/developers/docs/intro (date of access: 20.05.2025).

7) *Discord – group chat thats all fun & games*. URL: https://discord.com (date of access: 21.05.2025).

8) Documentation | llama. *Llama.* URL: https://www.llama.com/docs/overview/ (date of access: 21.05.2025).

9) Fine-tuning large language models (LLMs) in 2025 | SuperAnnotate. *SuperAnnotate.* URL: https://www.superannotate.com/blog/llm-fine-tuning (date of access: 20.05.2025).

10) Free text to speech & AI voice generator | elevenlabs. *ElevenLabs.* URL: https://elevenlabs.io (date of access: 20.05.2025).

11) Frontier AI LLMs, assistants, agents, services | Mistral AI. *Mistral AI.* URL: https://mistral.ai (date of access: 21.05.2025).

12) GitHub – alphacep/vosk-api: Offline speech recognition API for Android, iOS, Raspberry Pi and servers with Python, Java, C# and Node. *GitHub*. URL: https://github.com/alphacep/vosk-api (date of access: 20.05.2025).

13) GitHub – pyannote/pyannote-audio: neural building blocks for speaker diarization: speech activity detection, speaker change detection, overlapped speech detection, speaker embedding. *GitHub*. URL: https://github.com/pyannote/pyannote-audio (date of access: 20.05.2025).

14) GitHub – rvc-project/retrieval-based-voice-conversion: in preparation... *GitHub*. URL: https://github.com/RVC-Project/Retrieval-based-Voice-Conversion (date of access: 20.05.2025).

15) GRPC overview | API gateway documentation | google cloud. *Google Cloud*. URL: https://cloud.google.com/api-gateway/docs/grpc-overview (date of access: 20.05.2025).

16) *gRPC*. URL: https://grpc.io/docs/languages/python/quickstart/ (дата звернення: 20.05.2025).

17) Hugging face – the AI community building the future. Hugging Face – The AI community building the future. *Hugging face*. URL: https://huggingface.co (date of access: 20.05.2025).

18) Kokoro TTS: advanced text-to-speech AI with best efficiency. *Kokoro TTS*. URL: https://kokorotts.net (date of access: 20.05.2025).

19) Language models are few-shot learners. *arXiv.org*. URL: https://arxiv.org/abs/2005.14165 (date of access: 20.05.2025).

20) Pycord. *Pycord*. URL: https://pycord.dev (date of access: 20.05.2025).

21) Speech-to-Text documentation | cloud speech-to-text documentation | google cloud. *Google Cloud*. URL: https://cloud.google.com/speech-to-text/docs (date of access: 20.05.2025).