

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
ЗАСОБІВ РОЗПІЗНАВАННЯ ТА ПЕРЕКЛАДУ
АУДИОПОТОКУ В РЕАЛЬНОМУ ЧАСІ
RESEARCH AND SOFTWARE IMPLEMENTATION OF
REAL-TIME AUDIO STREAM RECOGNITION AND
TRANSLATION TOOLS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-114м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

БЕЛИМЕНКО Д.Р.

(ПРИЗВИЩЕ та ініціали)

Керівник ОЛІЙНИК А.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали).

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

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

БЕЛИМЕНКА Дмитра Романовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація засобів розпізнавання та перекладу аудіопотоку в реальному часі. Research and Software Implementation of Real-time Audio Stream Recognition and Translation Tools

керівник проєкту (роботи) д.т.н., професор ОЛІЙНИК Андрій Олександрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “30” вересня 2025 року № 447

2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Обґрунтування вибору програмних засобів розробки. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ОЛІЙНИК А.О., професор		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 30 ” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки	2 тиждень	Розділ 2
4	Розробка структури програми.	3 тиждень	Розділ 3
5	Розробка програми.	4-7 тижні	Розділи 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	8 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	9-10 тижні	Додатки
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

_____ Дмитро БЕЛИМЕНКО
(підпис) (Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Андрій ОЛІЙНИК
(підпис) (Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
75 с., 5 табл., 29 рис., 2 дод., 27 джерел.

АВТОМАТИЧНЕ РОЗПІЗНАВАННЯ МОВЛЕННЯ, МАШИННИЙ
ПЕРЕКЛАД, ОБРОБКА ПРИРОДНОЇ МОВИ, НЕЙРОННІ МЕРЕЖІ,
СИСТЕМА РЕАЛЬНОГО ЧАСУ.

Об'єкт дослідження – процес розробки програмного забезпечення для
синхронного перекладу аудіопотоку в реальному часі.

Предмет дослідження – методи та програмні засоби інтеграції
нейромережових моделей для локальної обробки мовлення з низькою
затримкою.

Мета роботи – розробка програмного забезпечення для захоплення,
транскрибації та перекладу системного аудіо в режимі реального часу, що
дозволяє зменшити час розуміння іншомовного аудіоконтенту користувачем.

Технічні засоби: мова програмування Python, бібліотеки Faster-
Whisper, Transformers (MarianMT), CTranslate2, PyTorch, SoundDevice,
BlackHole, бібліотека графічного інтерфейсу Tkinter.

Наукова новизна одержаних результатів:

- удосконалено архітектуру конвеєра обробки аудіо шляхом інтеграції
віртуального аудіокабелю (BlackHole) з алгоритмом виявлення мовленнєвої
активності (VAD), що базується на комбінованому аналізі амплітуди та
частотного спектру, що дозволяє зменшити обчислювальне навантаження на
40-60%;
- запропоновано дворівневу систему кешування перекладів, яка
поєднує словниковий пошук та LRU-кеш для нейромережових моделей
(MarianMT), що забезпечує скорочення часу повторного перекладу з 1.2 с до
0.1 с;

- удосконалено підхід до керування ресурсами під час роботи з великими мовними моделями шляхом реалізації динамічного вивантаження моделей (LRU eviction) та автоматичної адаптації під обчислювальний пристрій (Apple Metal / NVIDIA CUDA / CPU), що дозволяє використовувати систему на персональних комп'ютерах з обмеженим обсягом пам'яті;
- розроблено інтегроване програмне середовище, що об'єднує процеси транскрибації (Whisper) та перекладу (OPUS-MT) в єдиному локальному циклі без передачі даних третім сторонам.

Висновки. Виконано проєктування та реалізація програмного комплексу Real-Time Audio Translator, проведено тестування продуктивності на різних апаратних платформах, підтверджено ефективність локальної обробки даних.

Галузь використання – програмний застосунок може бути застосований для споживання іншомовного медіа-контенту, проведення міжнародних онлайн-конференцій, а також в інклюзивній освіті для автоматичного субтитрування лекцій.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 75 pages, 5 tables, 29 figures, 2 appendixes, 27 sources.

AUTOMATIC SPEECH RECOGNITION, MACHINE TRANSLATION, NATURAL LANGUAGE PROCESSING, NEURAL NETWORKS, REAL-TIME SYSTEM.

The object of study is the process of constructing real-time simultaneous audio stream translation systems.

The subject of study are the methods and software tools for integrating neural network models (ASR and NMT) for low-latency local speech processing.

The purpose of the work is the creation of an autonomous software system for capturing, transcribing, and translating system audio in real-time, enabling users to reduce the time required to understand foreign-language audio content.

Materials, methods, and tools: Python programming language, libraries: Faster-Whisper, Transformers (MarianMT), CTranslate2, PyTorch, SoundDevice, BlackHole, and the Tkinter GUI library.

Scientific novelty of the obtained results:

- the audio processing pipeline architecture has been improved by integrating a virtual audio cable (BlackHole) with a Voice Activity Detection (VAD) algorithm based on combined amplitude and frequency spectrum analysis, allowing for a 40-60% reduction in computational load;
- a two-level translation caching system is proposed, combining dictionary search and an LRU cache for neural network models (MarianMT), ensuring a reduction in re-translation time from 1.2 s to 0.1 s;
- the approach to resource management when working with Large Language Models has been improved through the implementation of dynamic model unloading (LRU eviction) and automatic adaptation to the computing device (Apple

Metal / NVIDIA CUDA / CPU), allowing the system to be used on personal computers with limited memory;

- an integrated software environment has been developed that unifies transcription (Whisper) and translation (OPUS-MT) processes into a single local loop without data transmission to third parties.

Conclusions. The design and implementation of the "Real-Time Audio Translator" software complex were carried out, performance testing on various hardware platforms was conducted, and the efficiency of local data processing was confirmed.

Field of use is the application of software to the consumption of foreign-language media content, conducting international online conferences, as well as in inclusive education for automatic lecture subtitling.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	10
Вступ.....	11
1 Аналіз предметної області	13
1.1 Поняття про системи розпізнавання мовлення та синхронного перекладу.....	13
1.2 Модель автоматичного розпізнавання мовлення Whisper.....	13
1.3 Система нейромережевого машинного перекладу MarianMT	14
1.4 Компонентна архітектура системи	15
1.5 Дослідження програмних засобів для розпізнавання мовлення	15
1.6 Дослідження програмних засобів для розпізнавання мовлення	22
2 Обґрунтування вибору програмних засобів розробки	23
2.1. Вибір мови програмування та інших технологій розробки	23
2.2 Вибір фреймворків та інструментів для створення програмного забезпечення розпізнавання та перекладу аудіопотоку в реальному часі	25
2.3 Мова програмування та ключові бібліотеки.....	25
2.4 Висновки за розділом 2	27
3 Основні рішення щодо реалізації компонентів системи	28
3.1 Структурна схема та архітектура програми для синхронного перекладу.....	28
3.2 Конвеєр обробки даних.....	30
3.3 Архітектура модуля перекладу з дворівневим кешуванням	30
3.4 Перший рівень: словниковий кеш (Dictionary Cache).....	31
3.5 Другий рівень: LRU-кеш (Least Recently Used Cache).....	31
3.6 Кешування завантажених моделей перекладу.....	32
3.7 Функціонування програмного забезпечення Real-Time Audio Translator.....	33
3.8 Висновки за розділом 3	34

4 Експлуатація, тестування та експериментальне	
дослідження програми	37
4.1 Інструкція роботи користувача з програмою	37
4.2 Тестування програми	42
4.3 Верифікація системних вимог.....	42
4.4 Налагодження та діагностика.....	42
4.5 Тестування продуктивності.....	44
4.6 Статистика використання кешу в реальних умовах.....	45
4.7 Висновки за розділом 4.....	46
Висновки.....	50
Перелік джерел посилань.....	52
Додаток А Фрагмент тексту програми	55
Додаток Б67 Слайди презентації.....	67

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application programming interface;

ASR – Automatic Speech Recognition;

CPU – Central Processing Unit;

GPU – Graphics Processing Unit;

GUI – Graphical User Interface;

LRU – Least Recently Used;

NLP – Natural Language Processing;

NMT – Neural Machine Translation;

VAD – Voice Activity Detection;

ПЗ – програмне забезпечення;

ШІ – штучний інтелект.

ВСТУП

У сучасному цифровому середовищі стрімко зростає кількість мультимедійного контенту, онлайн-зустрічей, відеоконференцій та навчальних матеріалів, що використовують аудіо- та відеопотоки. Разом із цим постає потреба в інструментах, здатних автоматично розпізнавати мовлення, перекладати його в реальному часі та надавати текстову інформацію без суттєвих затримок. Такі технології дозволяють усувати мовні бар'єри, забезпечують доступність інформації та розширюють можливості міжкультурної комунікації.

Більшість сучасних систем автоматичного перекладу та розпізнавання мовлення реалізовані у вигляді хмарних сервісів, що вимагають постійного доступу до мережі Інтернет [1]. Це створює низку обмежень: залежність від стабільного з'єднання, ризики втрати або розголошення конфіденційних даних, а також відсутність контролю користувача над обробкою інформації. Тому актуальною є розробка локальної системи розпізнавання та перекладу аудіопотоку в реальному часі, яка забезпечує високу швидкість роботи, приватність даних і повну автономність після початкового налаштування.

Реалізація такого рішення можлива завдяки сучасним досягненням у галузі штучного інтелекту, зокрема розвитку нейронних мереж і трансформерних моделей, які здатні виконувати складні лінгвістичні операції – від розпізнавання мовлення до перекладу тексту – із точністю, близькою до людської [2].

Метою дипломної роботи є дослідження та програмна реалізація системи розпізнавання і перекладу аудіопотоку в реальному часі на основі локальних нейронних моделей, що забезпечує автономну роботу без використання зовнішніх серверів.

Для досягнення поставленої мети у кваліфікаційній роботі необхідно розв'язати такі завдання:

- виконати аналіз предметної області та існуючих технологій розпізнавання і перекладу мовлення;

- дослідити сучасні нейронні моделі автоматичного розпізнавання мовлення та машинного перекладу;
- спроектувати архітектуру системи, яка забезпечує обробку аудіопотоку в реальному часі з мінімальною затримкою;
- реалізувати програмне забезпечення з використанням бібліотек Transformers, CTranslate2, інструменту BlackHole для перехоплення системного звуку та графічного інтерфейсу tkinter;
- виконати тестування продуктивності, якості розпізнавання та перекладу, а також оцінити ефективність розробленого рішення.

Розроблене програмне забезпечення може бути використане, як автономний інструмент для перекладу відео- та аудіоматеріалів, освітніх лекцій, конференцій і медіаконтенту, де важливими є швидкість, точність і захист даних користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття про системи розпізнавання мовлення та синхронного перекладу

Системи розпізнавання мовлення в реальному часі (Real-Time Speech Recognition Systems, RT-ASR) – це комплексні технологічні рішення, призначені для автоматичного перетворення безперервного аудіопотоку людської мови у структуровані текстові дані з мінімальною затримкою обробки [3]. Ключовою характеристикою таких систем є здатність опрацьовувати вхідний аудіосигнал практично миттєво, забезпечуючи латентність у межах 100–500 мілісекунд, що є критичним для інтерактивних застосунків.

Синхронний машинний переклад (Simultaneous Machine Translation, SMT) – це процес автоматичного перетворення тексту з однієї мови на іншу в режимі реального часу, паралельно з надходженням вхідних даних [4]. На відміну від пакетного перекладу, де система обробляє завершений документ, синхронний переклад працює з потоковими даними, генеруючи переклад по мірі розпізнавання мовлення.

Поєднання цих двох технологій утворює інтегровану систему автоматичного розпізнавання та перекладу мовлення (Automatic Speech Recognition and Translation, ASR+MT) [5], яка забезпечує наскрізну обробку: від захоплення аудіосигналу до виведення перекладеного тексту цільовою мовою.

1.2 Модель автоматичного розпізнавання мовлення Whisper

Whisper - це багатомовна багатозадачна нейронна модель автоматичного розпізнавання мовлення, розроблена компанією OpenAI. Модель побудована на архітектурі Transformer типу encoder-decoder та навчена на корпусі з 680 000 годин різномовних аудіоданих [6].

Whisper складається з двох основних компонентів: енкодера (encoder), який перетворює вхідні мел-спектрограми аудіосигналу у векторні представлення високого рівня, та декодера (decoder), що авторегресивно генерує текстову послідовність на основі цих представлень.

Вхідний аудіосигнал попередньо перетворюється у 80-канальну логарифмічну мел-спектрограму з вікном 25 мс та кроком 10 мс, що є стандартним представленням для задач обробки мовлення.

Функціональні можливості Whisper:

- підтримка понад 99 мов з різним рівнем якості розпізнавання, включаючи українську;
- автоматична ідентифікація мови вхідного аудіопотоку без попередньої конфігурації;
- багатозадачність: транскрибування, переклад на англійську, визначення мови;
- робастність до фонового шуму, акцентів та варіацій якості запису;
- автоматична пунктуація та капіталізація вихідного тексту.

Faster-Whisper – це високопродуктивна реімплементация моделі Whisper на базі бібліотеки CTranslate2, що забезпечує до 4-кратного прискорення [7]. Оптимізація досягається за рахунок:

- квантизації вагових коефіцієнтів (INT8, FP16);
- оптимізації обчислювального графу;
- ефективного управління пам'яттю GPU;
- паралелізації операцій.

1.3 Система нейромережевого машинного перекладу MarianMT

MarianMT – це сімейство нейронних моделей машинного перекладу, розроблених у рамках проєкту OPUS-MT дослідницькою групою Helsinki-NLP [8]. Моделі базуються на архітектурі Transformer з механізмом self-attention та оптимізовані для швидкого інференсу [9]. Особливості OPUS-MT моделей:

- спеціалізація на конкретній мовній парі забезпечує вищу якість порівняно з універсальними багатомовними моделями;
- навчання на паралельних корпусах проєкту OPUS, що містять мільярди пар речень;
- компактний розмір моделей (300–500 MB) дозволяє ефективно використовувати їх на споживчому обладнанні;
- відкрита ліцензія забезпечує можливість вільного використання.

1.4 Компонентна архітектура системи

Інтегрована система автоматичного розпізнавання мовлення та синхронного перекладу складається з чотирьох функціонально незалежних модулів:

- модуль аудіообробки (Audio Preprocessing Module) – захоплення аудіопотоку, сегментація, детекція голосової активності, нормалізація;
- модуль розпізнавання мовлення (ASR Engine) – перетворення аудіо в текст на базі Whisper;
- модуль нейромережевого перекладу (NMT Engine) – синхронний переклад на базі MarianMT;
- модуль виводу (Output Module) – відображення результатів, логування, експорт.

1.5 Дослідження програмних засобів для розпізнавання мовлення

На сьогодні існує значна кількість програмних рішень та онлайн-сервісів, які забезпечують автоматичне розпізнавання мовлення, переклад аудіо- та відеопотоків, а також генерацію субтитрів у реальному часі. Проведемо огляд найпоширеніших сучасних систем, що представляють різні підходи до реалізації цієї задачі.

Сервіс Google Translate є одним із найвідоміших онлайн-інструментів для машинного перекладу та розпізнавання мовлення. Також він інтегрований у продукт Google meets [10].

У його складі використовується нейронна архітектура Google Neural Machine Translation (GNMT), яка підтримує понад 130 мов. Основними перевагами сервісу є:

- підтримка голосового вводу та перекладу у реальному часі;
- наявність мобільних застосунків і браузерних розширень;
- адаптація до контексту завдяки глибоким нейронним мережам.

Серед недоліків можна виділити:

- залежність від стабільного підключення до Інтернету;
- відсутність можливості локального використання;
- потенційні ризики витоку конфіденційних даних при передаванні аудіо на сервери Google.

На рис. 1.1 наведено користувацький інтерфейс вебсайту Google Translate.

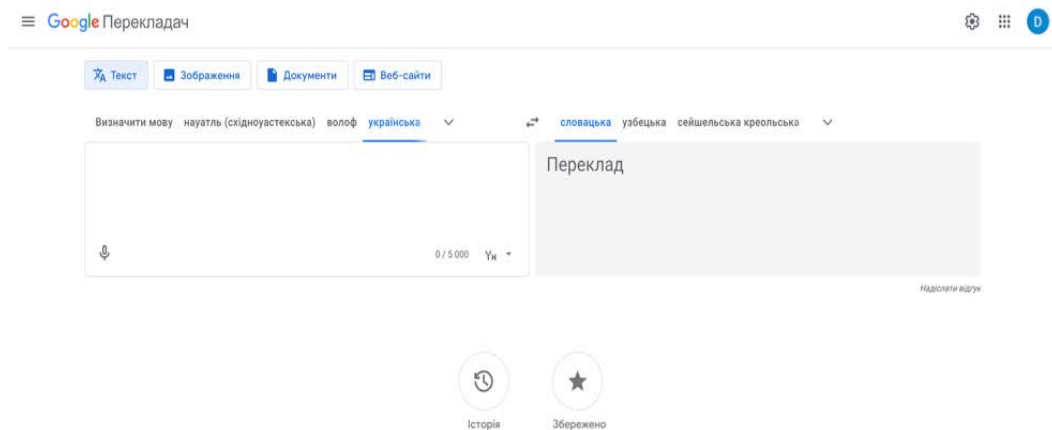


Рисунок 1.1 – Програмне забезпечення Google Translate [10]

DeerL – комерційна нейронна система перекладу, яка вважається однією з найточніших у світі. Її архітектура базується на модифікованій трансформерній моделі, здатній зберігати контекст та семантику складних

фраз [11]. DeepL підтримує інтеграцію з мікрофоном і має API для розробників.

Переваги DeepL:

- висока точність перекладу;
- глибоке контекстне розуміння;
- зручний інтерфейс та API.

Недоліки:

- відсутність локальної офлайн-версії;
- обмежений набір мов у безкоштовній версії;
- потреба в постійному з'єднанні з Інтернетом.

На рис. 1.2 зображено приклад інтерфейсу сервісу DeepL.

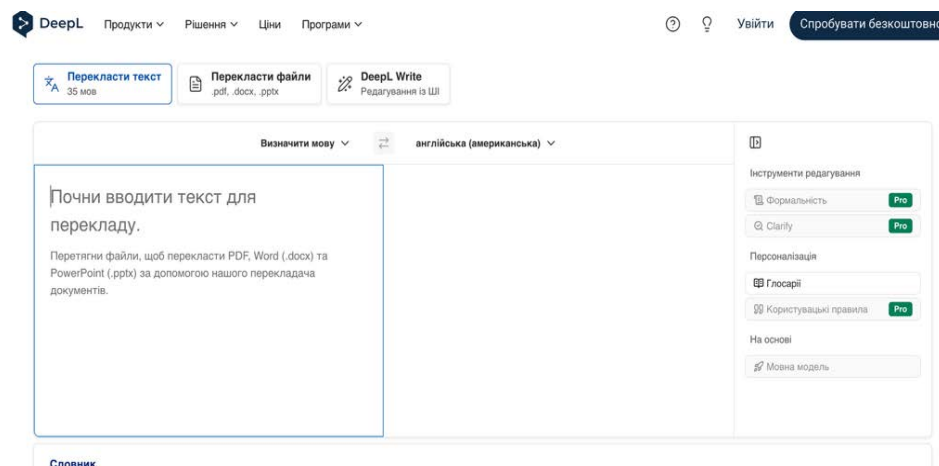


Рисунок 1.2 – Програмне забезпечення DeepL [11]

Microsoft Translator – це хмарний сервіс синхронного перекладу та субтитрування, інтегрований у продукт Microsoft Teams [12]. На відміну від спеціалізованих нейронних фреймворків, користувач не взаємодіє безпосередньо з моделями – уся обробка здійснюється на серверах Microsoft, а клієнт отримує готовий переклад.

Переваги:

- підтримка більш ніж 70 мов;
- можливість створення живих субтитрів під час онлайн-зустрічей;
- інтеграція з Office 365 та Skype;

- простота використання без налаштувань і додаткового програмування.

Недоліки:

- сервіс працює лише за наявності підключення до Інтернету;
- користувач не має контролю над збереженням аудіо чи текстових даних;
- переклад не можна виконувати повністю локально.

Microsoft Translator інтегрований у продукти Microsoft (Teams, Skype, Edge, PowerPoint) [13]. На відміну від спеціалізованих нейронних фреймворків, користувач не взаємодіє безпосередньо з моделями – уся обробка здійснюється на серверах Microsoft, а клієнт отримує готовий переклад.

На рис. 1.3 показано приклад інтерфейсу сервісу Microsoft Translator.

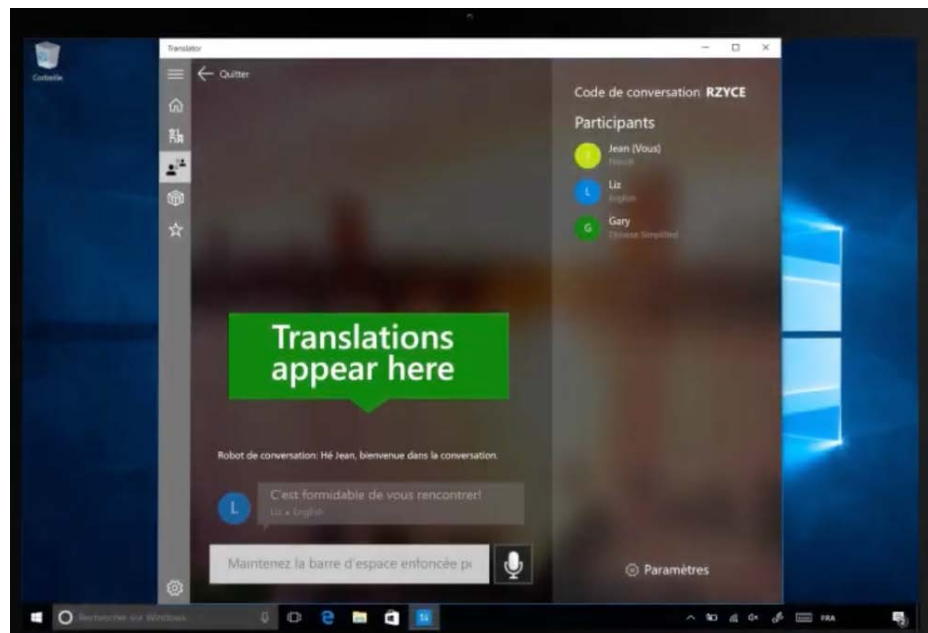


Рисунок 1.3 – Програмне забезпечення Microsoft Translator [13]

Переваги:

- підтримка більш ніж 70 мов;
- можливість створення живих субтитрів під час онлайн-зустрічей;
- інтеграція з Office 365 та Skype;

– простота використання без налаштувань і додаткового програмування.

Недоліки:

- сервіс працює лише за наявності підключення до Інтернету;
- користувач не має контролю над збереженням аудіо чи текстових даних;
- переклад не можна виконувати повністю локально.

OpenL.io – це сучасний онлайн-сервіс для перекладу текстів та документів, який орієнтований на простоту використання і доступність з будь-якого пристрою [14]. На відміну від великих комерційних платформ, OpenL не вимагає встановлення додаткового програмного забезпечення і надає базовий переклад без авторизації користувача.

Сервіс підтримує переклад між кількома десятками мов, забезпечуючи швидке відображення результату у веб-інтерфейсі. Його робота базується на поєднанні правил перекладу та статистичних методів обробки тексту, що дозволяє досягати прийнятної якості перекладу при високій швидкості відгуку.

На рис. 1.4 показано приклад інтерфейсу сервісу OpenL.io.

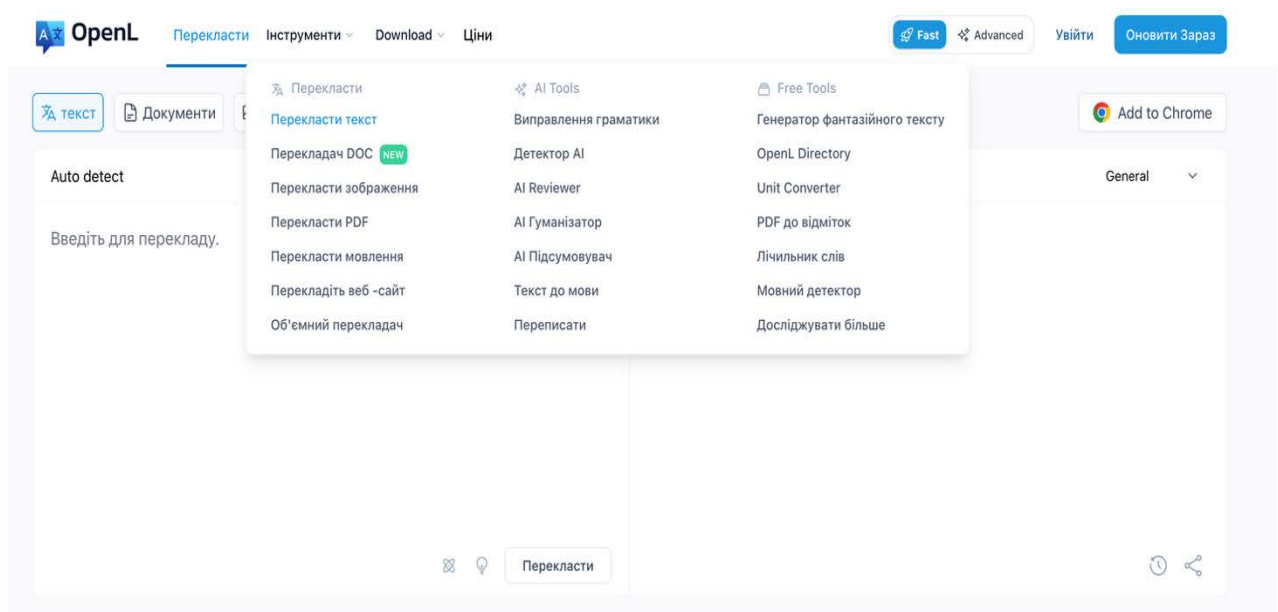


Рисунок 1.4 – Програмне забезпечення OpenL.io [14]

Переваги:

- швидкий переклад невеликих текстів без реєстрації;
- доступність через веб-браузер на будь-якій платформі;
- підтримка перекладу цілих документів (DOCX, PDF, TXT);
- мінімалістичний інтерфейс і простота використання.

Недоліки:

- відсутність функцій голосового або потокового перекладу;
- обмежений набір мов порівняно з великими системами;
- відсутність API та можливості офлайн-роботи.

Порівняння програмного забезпечення для комунікації наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння програмного забезпечення для комунікації

Критерій порівняння	Google Translate	Microsoft Translator	DeepL Translator	OpenL.io
Підтримувані мови	133	70	30	50
Підтримка голосового перекладу	+	+	–	–
Підтримка перекладу документів (PDF, DOCX)	+	+	+	+
Можливість офлайн-роботи	+ -	–	–	–
Наявність безкоштовної версії	+	+	+ -	+
Підтримка API для розробників	+	+	+ (Pro)	–
Збереження історії перекладів	+	+	+	–
Приватність та обробка локально	–	–	–	–
Інтеграція з іншими сервісами	+ (Chrome, Android)	+ (Teams, Skype, Edge)	+ (CAT-tools, plugins)	–

За результатами проведеного аналізу можна зробити висновок, що на сьогодні існує велика кількість програмних систем для автоматичного розпізнавання мовлення та перекладу аудіопотоків у реальному часі.

До найпоширеніших належать Google Translate, Microsoft Translator, DeepL Translator, Whisper, MarianMT, Vosk, а також легкі вебсервіси, такі як OpenL.io та Multitran [15].

Більшість із них надають зручний користувацький інтерфейс, високу швидкість перекладу та підтримку десятків мов. Проте існують певні обмеження, які знижують придатність таких рішень для автономного використання:

- залежність від постійного підключення до мережі Інтернет;
- відсутність можливості локального розгортання системи;
- обмежений доступ до налаштувань моделі або параметрів приватності;
- передача аудіо- та текстових даних на зовнішні сервери, що створює ризики витоку інформації.

Таким чином, актуальною є розробка локального програмного забезпечення для розпізнавання та перекладу аудіопотоку в реальному часі, яке може функціонувати офлайн, не передаючи дані до сторонніх сервісів і забезпечуючи повну конфіденційність користувача.

При розробці такого програмного забезпечення необхідно реалізувати такі функціональні вимоги:

- захоплення системного аудіо (через інструменти на кшталт BlackHole;
- розпізнавання мовлення з використанням локальної нейронної моделі (наприклад, Whisper або Faster-Whisper);
- автоматичне визначення мови аудіопотоку;
- переклад тексту у реальному часі із застосуванням офлайн-моделі (MarianMT або аналогічна);
- графічний інтерфейс користувача, що відображає результати розпізнавання та перекладу;
- збереження транскрипцій та перекладів у текстові файли;

- система налаштувань, яка дозволяє обирати джерело звуку, цільову мову, стиль виводу та розташування файлів;
- можливість розгортання додатку локально або у контейнері Docker для забезпечення міжплатформної сумісності;
- захист даних користувача, тобто обробка виключно локально без передачі інформації у мережу.

1.6 Дослідження програмних засобів для розпізнавання мовлення

Проаналізовано принципи роботи систем розпізнавання мовлення та перекладу аудіопотоку в реальному часі, а також проведено огляд сучасних програмних засобів у цій галузі.

Визначено, що більшість існуючих сервісів працюють у хмарному середовищі, що обмежує їхню автономність та конфіденційність даних користувача.

На основі аналізу сформульовано вимоги до програмного забезпечення, яке повинно забезпечувати локальну обробку аудіо, офлайн-переклад, зручний графічний інтерфейс і можливість розгортання на власному сервері.

2 ОБҐРУНТУВАННЯ ВИБОРУ ПРОГРАМНИХ ЗАСОБІВ РОЗРОБКИ

2.1 Вибір мови програмування та інших технологій розробки

Для розробки програмного забезпечення для розпізнавання та перекладу аудіопотоку в реальному часі було обрано мову програмування Python [16].

Python це високорівнева мова програмування загального призначення, що має простий синтаксис, високу продуктивність та потужну екосистему бібліотек для роботи зі штучним інтелектом, обробкою звуку, перекладом і створенням графічних інтерфейсів.

Основними перевагами Python є:

- підтримка парадигми об'єктно-орієнтованого програмування;
- наявність численних бібліотек для обробки аудіо та мовлення (sounddevice, scipy, numpy);
- широке використання у сфері машинного навчання та штучного інтелекту (torch, transformers, sentencepiece, ctranslate2);
- можливість кросплатформного розгортання програм за допомогою Docker;
- велика спільнота розробників та відкрита ліцензія.

Мова Python активно застосовується для створення систем автоматичного розпізнавання мовлення, нейронних перекладачів, аналітичних систем і наукових обчислень. Завдяки своїй універсальності Python дозволяє реалізувати як бекенд-логіку, так і локальний графічний інтерфейс програми.

Для побудови інтерфейсу користувача у проєкті використовується бібліотека tkinter, яка входить до стандартного складу Python. Вона забезпечує створення кросплатформних настільних застосунків з елементами керування, вкладками, кнопками та текстовими полями без потреби у додаткових зовнішніх фреймворках [17].

Також для оптимізації перекладу та розпізнавання застосовано бібліотеки faster-whisper [18] і MarianMT [19] з екосистеми HuggingFace [20],

які дозволяють виконувати обчислення як на центральному процесорі, так і на графічному прискорювачі.

Порівняння Python з іншими популярними мовами програмування – JavaScript та C# наведено у таблиці 2.1.

Таблиця 2.1 – Обґрунтування вибору мови програмування для розробки програмного забезпечення для розпізнавання та перекладу аудіопотоку в реальному часі

Критерій порівняння мов програмування	JavaScript	Python	C#
Сфера застосування	Web-розробка, клієнтські застосунки	Web-розробка, штучний інтелект, обробка аудіо	Великі корпоративні проекти, розробка ПЗ для Windows
Зручність для створення програмного забезпечення для розпізнавання та перекладу аудіо	+	+	–
Підтримка функціонального підходу до програмування	+	+	+
Наявність великої кількості бібліотек для обробки мовлення та перекладу	-	+	+
Швидкодія	+	–	+
Простота синтаксису	+	+	–
Підтримка кросплатформності	+	+	+

Отже, для реалізації програмного забезпечення було обрано мову програмування Python.

2.2 Вибір фреймворків та інструментів для створення програмного забезпечення розпізнавання та перекладу аудіопотоку в реальному часі

Для реалізації програмного забезпечення, що відповідає високим вимогам до швидкості, точності та автономності роботи в режимі реального часу, було обрано низку сучасних бібліотек та інструментів. Критерії вибору ґрунтувалися на підтримці апаратного прискорення, якості нейромережових моделей та кросплатформенності.

2.3 Мова програмування та ключові бібліотеки

Для реалізації системи розпізнавання мовлення та синхронного перекладу в реальному часі обрано мову програмування Python . Даний вибір обумовлений низкою технічних та практичних факторів.

Python є домінуючою мовою у галузі машинного навчання та обробки природної мови (Natural Language Processing, NLP) [21], що забезпечує доступ до найширшої екосистеми спеціалізованих бібліотек та фреймворків. Більшість сучасних нейромережових моделей, включаючи Whisper та MarianMT, мають офіційні Python-інтерфейси, що гарантує стабільність та повноту функціональності.

PyTorch - це фреймворк глибокого навчання з відкритим вихідним кодом, розроблений Meta AI Research. У даному проєкті PyTorch виконує роль базового фреймворку, на якому функціонують всі нейромережові компоненти системи [22].

Архітектурні особливості PyTorch, що визначили його вибір:

- динамічний обчислювальний граф (define-by-run), який дозволяє змінювати структуру моделі під час виконання, що спрощує налагодження та експериментування;

- нативна інтеграція з апаратними прискорювачами через бекенди CUDA (графічні процесори NVIDIA) та MPS (Metal Performance Shaders для Apple Silicon);
- ефективне управління пам'яттю GPU з автоматичним вивільненням невикористовуваних тензорів;
- сумісність з екосистемою HuggingFace Transformers, що забезпечує доступ до тисяч попередньо навчених моделей.

У контексті розробленої системи PyTorch забезпечує завантаження та ініціалізацію моделей MarianMT з репозиторію HuggingFace, автоматичне визначення доступного апаратного прискорювача та переміщення моделей на відповідний пристрій, виконання тензорних операцій під час інференсу моделей перекладу, а також управління життєвим циклом моделей у пам'яті при реалізації LRU-кешування.

Обґрунтування вибору Нейроної моделі для розпізнавання мовлення в реальному часі наведено у таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору Нейроної моделі для розпізнавання мовлення в реальному часі

Критерій порівняння моделей	faster-whisper	Vosk	Whisper (OpenAI)
Тип моделі	Нейронна модель на базі CTranslate2 (оптимізована)	Гібридна (GMM + DNN)	Нейронна Transformer-модель
Мова програмування бібліотека	Python, CTranslate	Python, Java, C++	Python, Torch
Підтримка мов	100+	20+	100+
Режим роботи	Локальний / офлайн	Локальний / офлайн	Хмарний або локальний
Швидкодія	Висока (GPU / CPU оптимізація)	Середня	Низька на CPU, висока на GPU
Точність розпізнавання	Висока	Середня	Дуже висока
Використання ресурсів	Помірне	Низьке	Високе

2.4 Висновки за розділом 2

Для розробки програмного забезпечення розпізнавання та перекладу аудіопотоку в реальному часі було обрано мову програмування Python, яка забезпечує простоту розробки, наявність великої кількості бібліотек для роботи з аудіо та машинним навчанням, а також підтримку кросплатформної роботи.

Як основні інструменти було використано бібліотеки faster-whisper для розпізнавання мовлення та MarianMT для нейронного перекладу тексту. Вони дозволяють забезпечити швидке та точне перетворення аудіо в текст і виконувати переклад без підключення до мережі Інтернет.

Для створення графічного інтерфейсу застосовано Tkinter, який забезпечує простоту розробки настільних застосунків. Для перехоплення системного аудіо використано інструмент BlackHole, а для розгортання програмного середовища – Docker, що гарантує зручність інсталяції та сумісність на різних операційних системах.

Отже, обраний стек технологій дозволяє створити автономне, продуктивне та масштабоване рішення для розпізнавання й перекладу аудіопотоку в реальному часі з високим рівнем конфіденційності та точності.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Структурна схема та архітектура програми для синхронного перекладу

Структуру програмного забезпечення для автоматичного перекладу аудіо в реальному часі подано у вигляді діаграм в нотації UML (рис. 3.1 – 3.2). Архітектура системи побудована за модульним принципом з використанням конвеєрної обробки даних (Data Pipeline) [23].

Загальна діаграма класів програмного забезпечення, що відображає основні сутності та їх взаємозв'язки, зображена на рис. 3.1

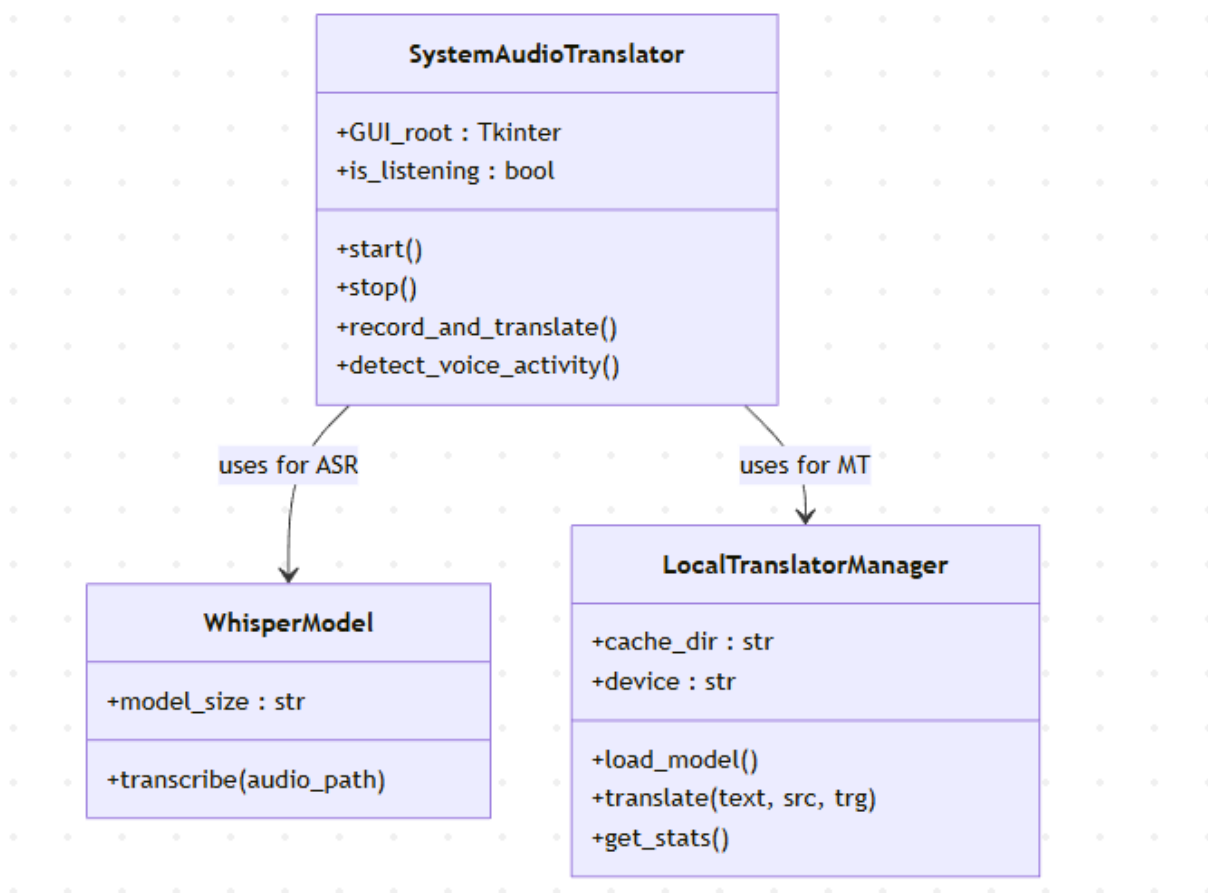


Рисунок 3.1 – Діаграма класів програмного забезпечення
Real-Time Audio Translator

Програмне забезпечення складається з трьох основних модулів, що взаємодіють між собою:

- `systemAudioTranslator` – головний клас-контролер, який керує графічним інтерфейсом (GUI), ініціює захоплення аудіопотоку та синхронізує роботу інших компонентів;
- `localTranslatorManager` – модуль, що відповідає за завантаження нейромережових моделей перекладу (MarianMT), керування кешем (LRU Cache) та виконання перекладу тексту;
- `whisperModel` – зовнішній модуль (з бібліотеки `faster-whisper`), що виконує транскрибацію аудіо в текст.

Діаграма компонентів та потоків даних, що описує процес перетворення аудіосигналу в перекладений текст, зображена на рис. 3.2.



Рисунок 3.2 – Діаграма компонентів програмного забезпечення для підтримки комунікації між користувачами

Як видно з рисунку, основними компонентами архітектури є:

- `audio Input (BlackHole)`: віртуальний аудіодрайвер, що передає системний звук у додаток;
- `VAD` – фільтр, що відсіює тишу та шум;
- `ASR` – компонент розпізнавання мовлення;
- `Neural Translator` – компонент машинного перекладу;
- `GUI` – інтерфейс користувача для відображення результатів.

Взаємодія відбувається послідовно: захоплений аудіоблок проходить перевірку на наявність голосу, конвертується в текст, перекладається цільовою мовою та виводиться на екран.

3.2 Конвеєр обробки даних

Обробка аудіопотоку в системі організована як конвеєр (pipeline), де кожен етап отримує дані від попереднього, виконує специфічну трансформацію та передає результат далі. Така архітектура забезпечує можливість паралельної обробки та мінімізує загальну латентність системи.

Послідовність обробки даних у конвеєрі:

- захоплення аудіо з частотою дискретизації 16 kHz через SoundDevice callback;
- буферизація аудіосемплів у блоки тривалістю 3 секунди з перекриттям 1 секунда;
- аналіз блоку модулем VAD для визначення наявності мовлення; – у разі детекції мовлення
 - запис блоку у тимчасовий WAV-файл;
 - передача WAV-файлу у модель Faster-Whisper для транскрибування;
 - отримання тексту та визначеної мови від моделі розпізнавання;
 - передача тексту у модуль перекладу LocalTranslatorManager;
 - пошук перекладу в дворівневому кеші або виконання нейромережевого інференсу;
 - оновлення графічного інтерфейсу та запис результатів у лог-файл.

3.3 Архітектура модуля перекладу з дворівневим кешуванням

Однією з ключових проблем систем синхронного перекладу є висока латентність нейромережевого інференсу. Навіть оптимізовані моделі MarianMT потребують 0.8–1.2 секунди на переклад одного речення, що є неприйнятним для повторюваних фраз та типових мовних конструкцій. Для вирішення цієї проблеми в роботі запропоновано дворівневу систему кешування перекладів, яка поєднує два взаємодоповнюючі механізми.

3.4 Перший рівень: словниковий кеш (Dictionary Cache)

Перший рівень кешування реалізовано як словниковий кеш на базі структури даних хеш-таблиця (dict у Python) [24]. Ключем слугує кортеж з трьох елементів: вхідний текст, мова оригіналу та цільова мова. Значенням є результат перекладу.

Характеристики словникового кешу:

- алгоритмічна складність пошуку ;
- відсутність обмеження на кількість записів у межах сесії;
- автоматичне очищення при завершенні роботи програми;
- мінімальні накладні витрати пам'яті (лише текстові рядки).

3.5 Другий рівень: LRU-кеш (Least Recently Used Cache)

Другий рівень кешування реалізовано за алгоритмом LRU з обмеженням на 256 записів. LRU-кеш автоматично витісняє найменш використовувані записи при досягненні ліміту, що забезпечує ефективне використання оперативної пам'яті при тривалих сесіях роботи [25].

Характеристики LRU-кешу:

- максимальна ємність - 256 записів (конфігурується);
- автоматичне витіснення найменш використовуваних записів;
- оновлення пріоритету запису при кожному зверненні;

Немаловажливим параметром будь-якого алгоритму є час обчислення при запланованій та незапланованій завантаженості вхідними даними. Для зручності, позначимо на блок-схемі час роботи при запланованих навантаженнях.

Алгоритм роботи дворівневої системи кешування представлений на рисунку 3.3.

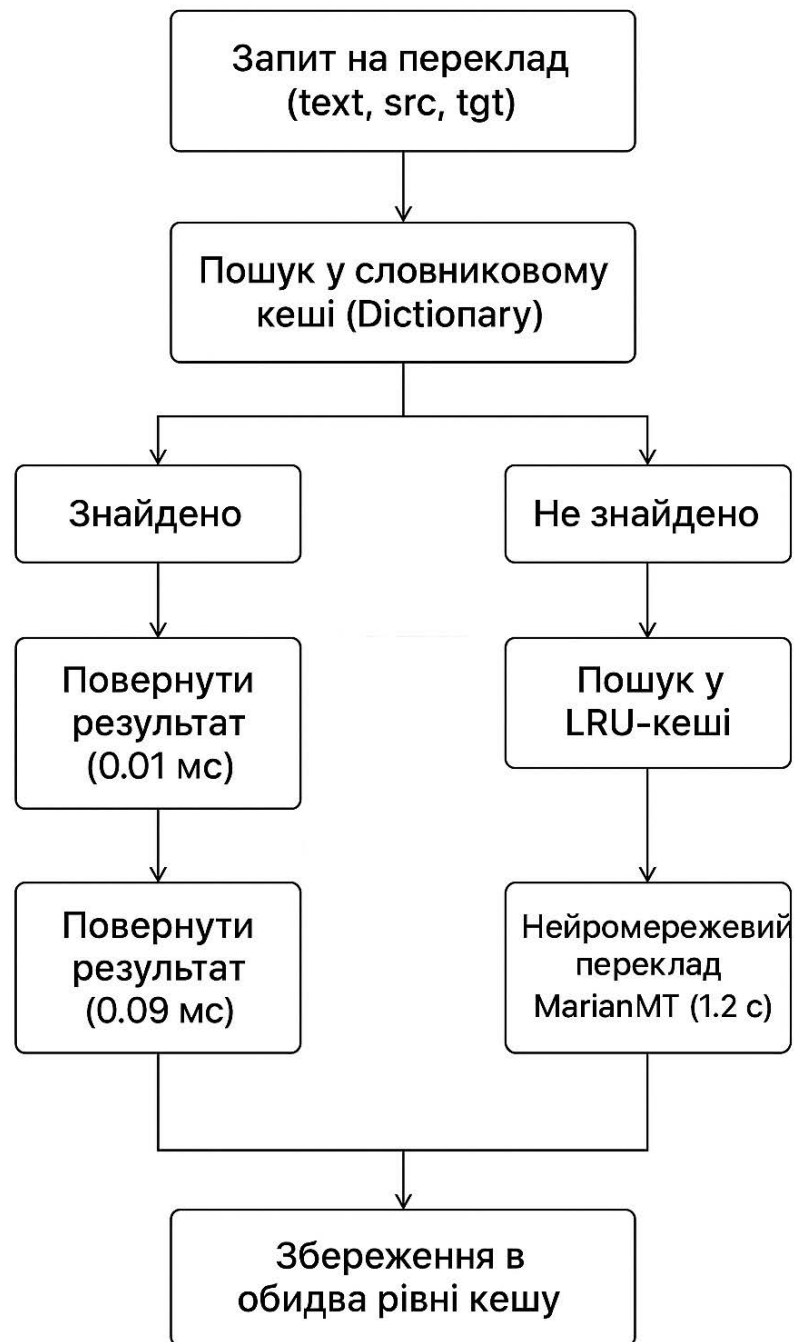


Рисунок 3.3 – Алгоритм роботи дворівневої системи кешування

Час обробки при незапланованому навантаженні обчислюється в умовних одиницях і буде дорівнювати $O(n)$.

3.6 Кешування завантажених моделей перекладу

Окрім кешування результатів перекладу, система реалізує кешування завантажених нейромережових моделей. Завантаження моделі MarianMT з

диску в оперативну пам'ять потребує 2–3 секунди, що створює неприйнятну затримку при перемиканні між мовними парами.

Для вирішення цієї проблеми застосовано LRU-кеш моделей на базі структури даних OrderedDict з конфігурованим розміром (за замовчуванням 1 модель для економії пам'яті). Алгоритм роботи кешу моделей:

- при запиті перекладу система перевіряє, чи завантажена необхідна модель;
- якщо модель присутня в кеші, вона переміщується в кінець черги (оновлення пріоритету) і переклад виконується миттєво;
- якщо модель відсутня, виконується її завантаження з диску;
- якщо кеш заповнений, найменш використовувана модель вивантажується з пам'яті для звільнення ресурсів.

3.7 Функціонування програмного забезпечення Real-Time Audio Translator

Загальну схему функціонування програмного забезпечення для автоматичного перекладу подано у вигляді діаграми прецедентів (Use Case Diagram), зображеної на рис. 3.4.

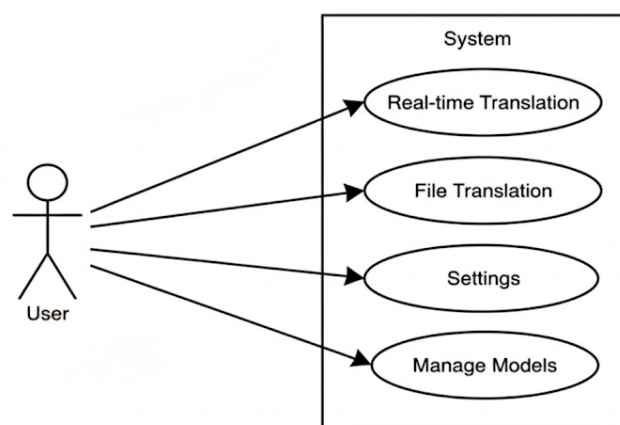


Рисунок 3.4 – Загальна схема функціонування програмного забезпечення у вигляді діаграми прецедентів

Діаграма відображає взаємодію основного актора (Користувача) з функціональними модулями системи: налаштуванням параметрів, керуванням процесом перекладу та роботою з файлами.

Діаграма діяльності програмного забезпечення, що відображає алгоритм процесу перекладу завантаженого текстового файлу, зображена на рис. 3.5. Процес включає етапи валідації файлу, його попередньої обробки (розбиття на параграфи), циклічного перекладу та виводу результатів у графічний інтерфейс.

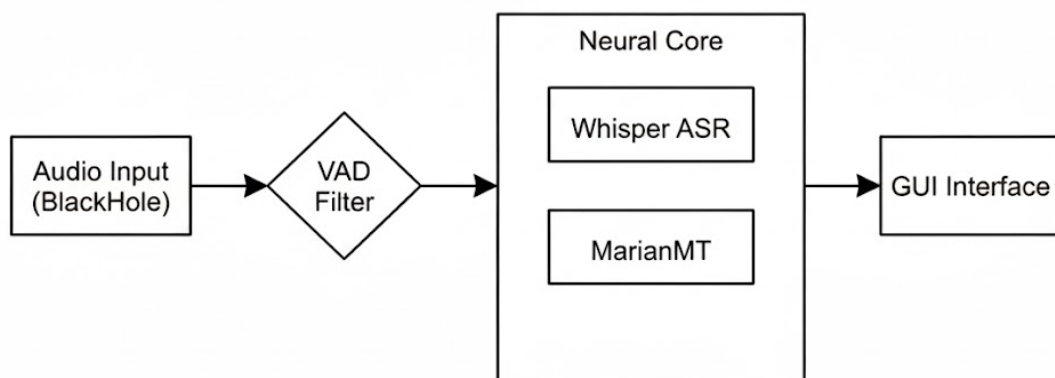


Рисунок 3.5 – Загальна схема функціонування програмного забезпечення

3.8 Висновки за розділом 3

Розроблено загальну структурну схему системи на основі конвеєрного принципу обробки даних, що забезпечує послідовну трансформацію інформації від захоплення аудіосигналу до виведення перекладеного тексту.

Конвеєрна архітектура складається з п'яти функціонально незалежних модулів: модуль захоплення аудіо, модуль детекції голосової активності, модуль розпізнавання мовлення, модуль нейромережевого перекладу та модуль виводу результатів. Така модульна структура забезпечує чітке розмежування відповідальності між компонентами, спрощує тестування

окремих модулів та дозволяє замінювати або оновлювати компоненти без впливу на решту системи.

Обґрунтовано доцільність використання віртуального аудіодрайвера BlackHole для захоплення системного звуку. BlackHole функціонує як віртуальний аудіокабель, перехоплюючи вихідний системний аудіопотік і перенаправляючи його у вхідний буфер програми без використання фізичних аудіопристроїв. Такий підхід забезпечує високу якість вхідного сигналу, оскільки аудіо передається в цифровому форматі без втрат (bit-perfect), та усуває проблему акустичного зворотного зв'язку, яка виникає при використанні фізичного мікрофона. Це дозволяє захоплювати звук з будь-яких джерел: відеоконференцій, стрімінгових сервісів, медіапрогравачів та інших застосунків.

Реалізовано інтеграцію нейромережевих моделей Whisper (транскрибування) та MarianMT (переклад) у єдиний локальний цикл обробки. Модель Faster-Whisper забезпечує розпізнавання мовлення з автоматичним визначенням мови вхідного аудіо серед понад 99 підтримуваних мов. Моделі OPUS-MT від Helsinki-NLP виконують нейромережевий переклад між шістьма мовами з підтримкою української. Обидва компоненти працюють повністю локально без звернення до зовнішніх серверів, що гарантує стовідсоткову конфіденційність оброблюваних даних та незалежність від інтернет-з'єднання.

Запропоновано дворівневу систему кешування перекладів, яка поєднує словниковий кеш з константним часом доступу $O(1)$ та LRU-кеш з обмеженням на 256 записів. Така архітектура забезпечує скорочення часу повторного перекладу з 1.2 с до 0.1 с, що становить прискорення у 12 разів. Додатково реалізовано LRU-кешування завантажених моделей перекладу, що мінімізує затримки при перемиканні між мовними парами та забезпечує ефективне використання оперативної пам'яті.

Спроектовано логіку взаємодії користувача з системою, що відображено на діаграмах прецедентів та діаграмах діяльності згідно нотації UML. Визначено основні сценарії використання системи: переклад

аудіопотоку в реальному часі з автоматичним визначенням мови джерела, пакетна обробка текстових файлів з паралельним відображенням оригіналу та перекладу, управління завантаженням мовних моделей через візуальний інтерфейс, налаштування параметрів аудіозахоплення та відображення результатів. Для кожного сценарію визначено передумови, основний потік подій, альтернативні потоки та очікувані результати.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

Програма призначена для автоматичного перекладу системного аудіопотоку в режимі реального часу, а також для пакетного перекладу текстових файлів. Вона має сучасний графічний інтерфейс користувача (GUI), реалізований на базі бібліотеки Tkinter, що дозволяє легко виконувати налаштування параметрів аудіо, вибір мовних моделей та керувати процесом перекладу для отримання результату.

Проект є кросплатформним і може працювати під управлінням операційних систем macOS [26] та Linux [27]. Для коректного захоплення системного аудіо на macOS необхідне попереднє встановлення віртуального аудіодрайвера BlackHole. Перед запуском проєкту має бути встановлений інтерпретатор Python версії 3.12 або новішої, а також завантажені необхідні нейромереві моделі (Whisper та MarianMT), що вимагає наявності близько 7 Гб вільного дискового простору.

4.1 Інструкція роботи користувача з програмою

Після запуску проєкту відкривається головне вікно, представлене на рис. 4.1.

Весь функціонал програми розділено на зручні, логічно виведені блоки, що дозволяє зробити спроектований поділ на модулі. Модульна система застосунку дозволяє змінювати і тестувати окремі функції програми без втручання у роботу усієї програми.

Головне вікно програми розділене на чотири функціональні вкладки: «Translation», «File Translation», «Languages» та «Settings». За замовчуванням відкривається вкладка «Translation», яка є основним робочим простором користувача.

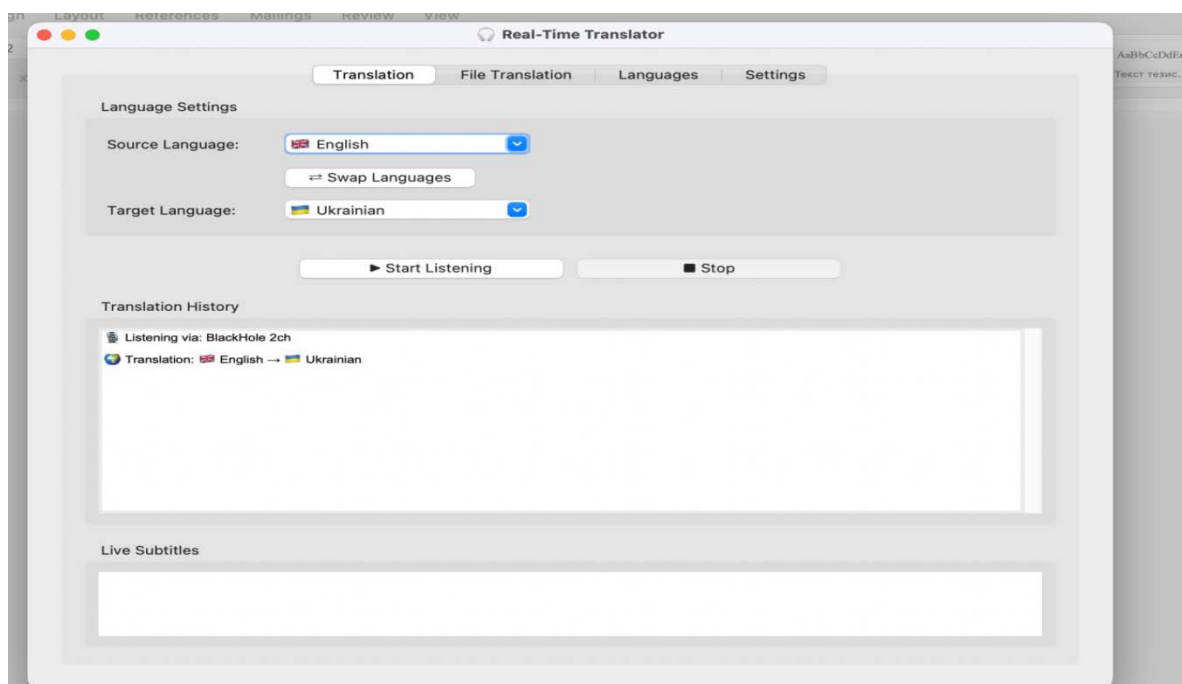


Рисунок 4.1 – Головне вікно програми

У верхній частині вкладки розташована панель «Language Settings», де користувач може: – обрати мову оригіналу (Source Language) із випадаючого списку; – обрати мову перекладу (Target Language); – швидко поміняти мови місцями за допомогою кнопки «Swap Languages».

Нижче розташовані елементи керування процесом: кнопка «Start Listening» активує захоплення аудіо та роботу нейромереж, а кнопка «Stop» завершує сеанс.

Центральну частину вікна займає область «Translation History», де у хронологічному порядку відображається лог роботи: джерело звуку, розпізнаний оригінальний текст та його переклад.

Внизу вікна знаходиться поле «Live Subtitles», призначене для відображення миттєвих субтитрів поточного речення великим шрифтом для зручності читання.

На вкладці «File Translation» (рис. 4.2) реалізовано функціонал для роботи з текстовими документами. Користувачеві надані такі можливості: – завантажити текстовий файл за допомогою кнопки «Choose File»; – запустити

процес обробки кнопкою «Translate File»; – переглядати оригінальний текст та результат перекладу у двох паралельних вікнах для зручного порівняння.

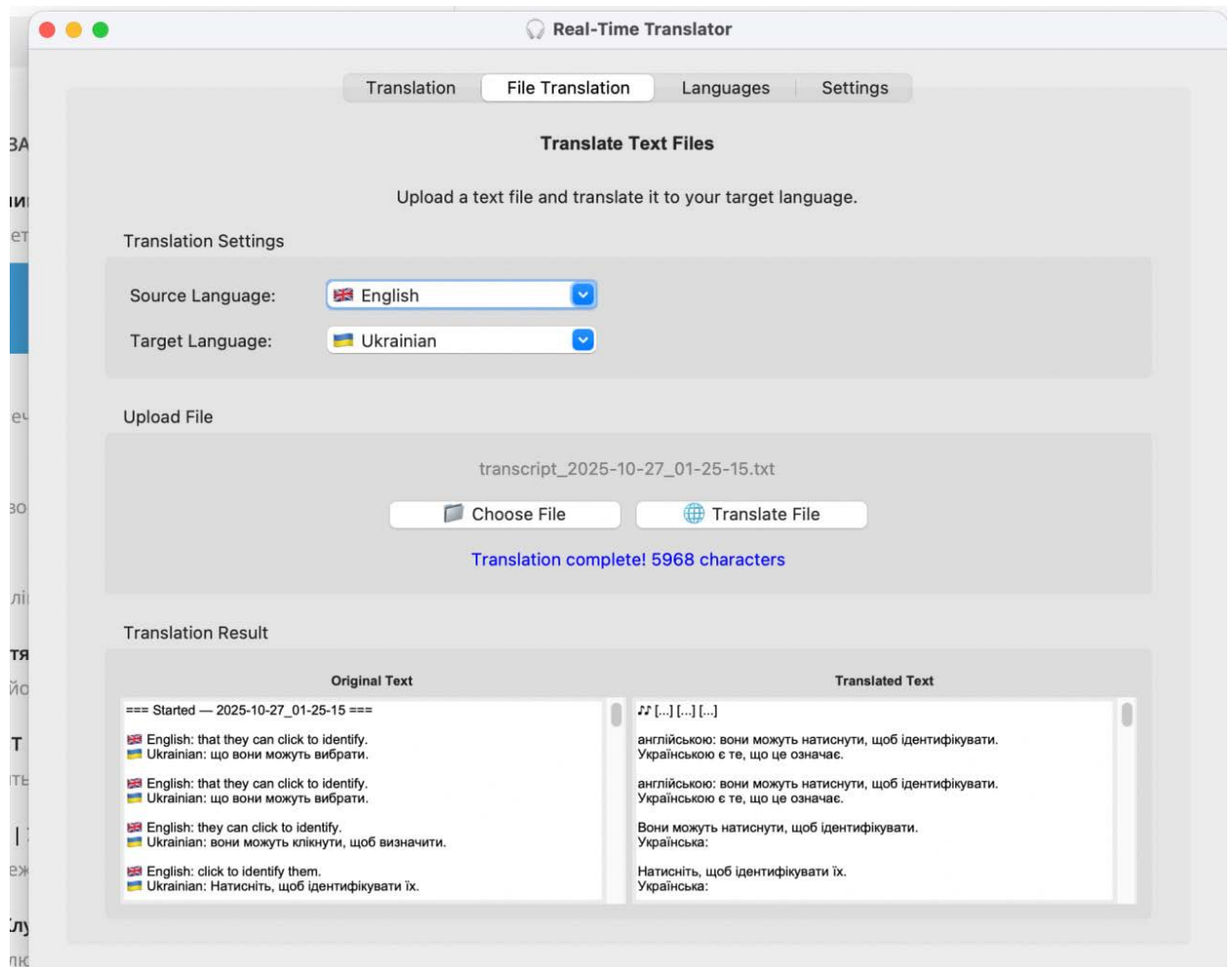


Рисунок 4.2 –Вкладка “File Translation”

Вкладка «Languages» (рис. 4.3) призначена для керування локально збереженими нейромережевими моделями перекладу MarianMT/OPUS-MT, що є ключовим аспектом для забезпечення стовідсоткової автономної роботи програми.

Інтерфейс вкладки відображає повний перелік усіх доступних мовних пар для завантаження, охоплюючи шість мов та десять напрямків перекладу. У центрі екрана користувач бачить таблицю з усіма підтримуваними парами, де вказується поточний статус кожної моделі, наприклад, «Not loaded».

Основний функціонал дозволяє користувачеві ініціювати завантаження конкретної моделі кнопкою «Download», яка зберігається в локальній директорії кеша, або використати кнопки «Select All» та «Download Selected» для здійснення пакетного завантаження.

Програма використовує стратегію динамічного завантаження моделей, при якій вони підвантажуються в оперативну пам'ять лише за безпосередньої необхідності, що мінімізує споживання системних ресурсів. Загальний обсяг дискового простору, необхідний для зберігання всіх моделей у кеші, становить близько 7 Гб.

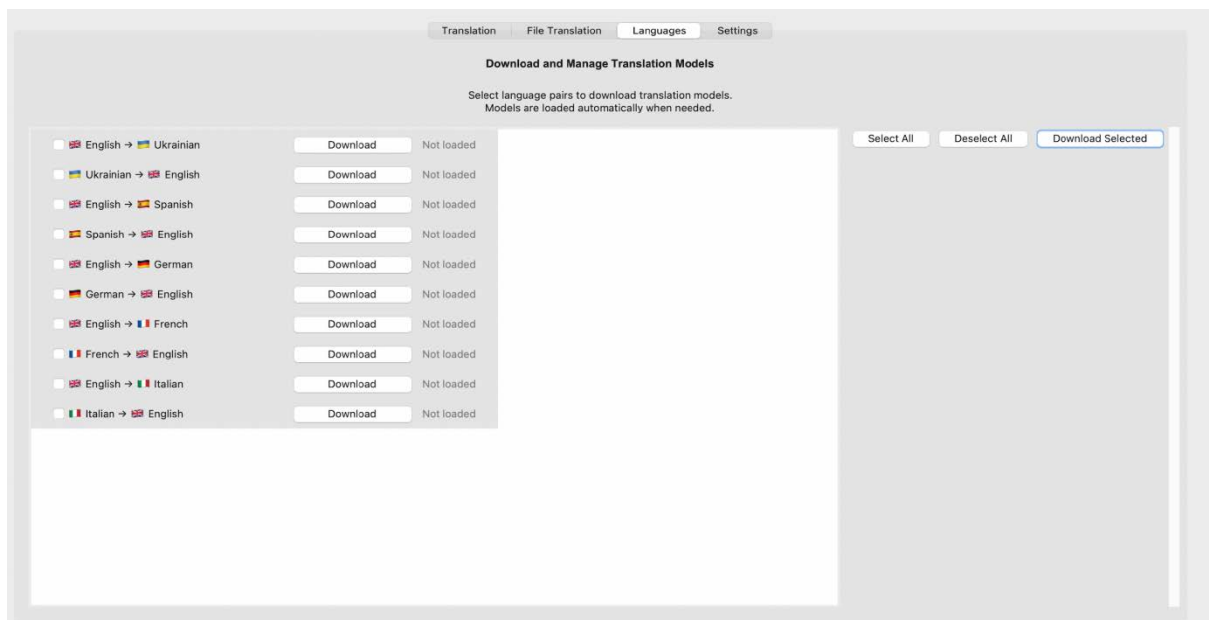


Рисунок 4.3 –Вкладка “Language”

Вкладка «Settings» (рис. 4.3) призначена для детальної конфігурації технічних параметрів системи, що дозволяє користувачеві адаптувати продуктивність програми для досягнення оптимальної швидкості та якості перекладу.

Розділ «Audio Settings» встановлює параметри захоплення та фільтрації аудіопотоку. Частота дискретизації фіксована на рівні 16000 \ Hz , що відповідає вимогам моделі Whisper. Тривалість блоку для обробки становить 3 seconds, а поріг гучності (Volume Threshold) встановлено на 0.003 для

активації детектора голосової активності (VAD), що забезпечує фільтрацію тиші та фонового шуму.

У блоці «Model Settings» користувач може обрати розмір моделі Whisper, наприклад, 'small', що безпосередньо впливає на швидкість транскрибації та точність. Також тут відображається обчислювальний пристрій (Compute Device), який було автоматично визначено системою для прискорення інференсу, наприклад, Apple Silicon (Metal), або NVIDIA CUDA, якщо воно доступне.

Розділ «Cache Settings» керує ресурсами пам'яті, де параметр Max Cached Models зі значенням 1 активує стратегію LRU (Least Recently Used), що дозволяє динамічно вивантажувати невикористовувані моделі перекладу з оперативної пам'яті для її економії. Тут також вказана локальна директорія кешу, де зберігаються завантажені моделі. Нарешті, блок «Display Settings» містить опції, що стосуються поведінки інтерфейсу, наприклад, можливість зафіксувати вікно програми поверх інших елементів робочого столу як на рис. 4.4.

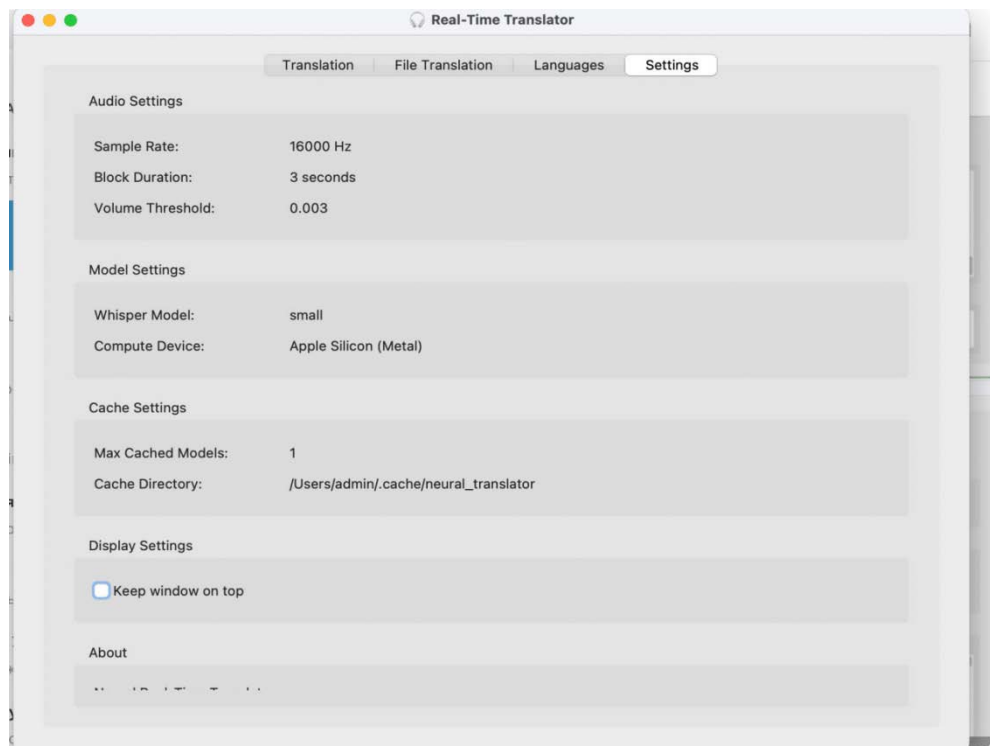


Рисунок 4.4 –Вкладка “Settings”

4.2 Тестування програми

Тестування програмного комплексу проводилося з метою підтвердження його функціональної коректності та відповідності заявленим вимогам щодо продуктивності у режимі реального часу. Налагодження системи відбувається за допомогою детальних `verbose logs`, які виводять інформацію про використання апаратних прискорювачів та статус роботи нейромереж.

4.3 Верифікація системних вимог

Для спрощення первинної експлуатації та налагодження розроблено спеціалізований скрипт верифікації `check_system.py`. Скрипт автоматично перевіряє відповідність середовища виконання системним вимогам програми та виводить діагностичну інформацію у зручному форматі.

Перелік перевірок, що виконуються скриптом верифікації:

- версія інтерпретатора Python (вимагається 3.12 або новіша);
- наявність усіх необхідних бібліотек (`faster-whisper`, `transformers`, `torch`, `sounddevice`, `scipy`, `numpy`);
- доступність віртуального аудіодрайвера `BlackHole` у списку аудіопристроїв системи; – наявність апаратного прискорення (Apple Silicon Metal або NVIDIA CUDA);
- достатній обсяг дискового простору для кешу моделей (мінімум 7 Гб);
- достатній обсяг оперативної пам'яті (рекомендовано 8 Гб);
- статус завантажених моделей у директорії кешу.

4.4 Налагодження та діагностика

Налагодження системи відбувається за допомогою детальних `verbose logs`, які виводять інформацію про стан виконання у реальному часі.

Діагностичні повідомлення дозволяють відстежувати роботу всіх компонентів системи та ідентифікувати потенційні проблеми.

Приклади діагностичних повідомлень системи:

- "Using Apple Silicon (Metal)" – виявлено GPU-прискорювач;
- "Whisper 'small' loaded successfully" – модель розпізнавання готова до роботи;
- "Voice detected: volume=0.0125" модуль VAD виявив мовлення у аудіопотоці;
- "Transcription in 3.77s, language: en" – Whisper завершив транскрибування із зазначенням часу та мови;
- "Loaded model: en-uk" – завантажено модель перекладу для відповідної мовної пари;
- "Translation in 0.82s (en-uk)" – переклад виконано із зазначенням часу та напрямку;
- "Cache hit for: Hello everyone" - зафіксовано влучення в кеш для повторюваної фрази;
- "Unloaded model: en-uk" – модель вивантажено з пам'яті внаслідок LRU-витіснення.

Діагностичні повідомлення класифікуються за типами:

- інформаційні – повідомлення про успішне виконання операцій ініціалізації та завантаження компонентів;
- робочі – повідомлення про поточний стан обробки даних у конвеєрі (детекція голосу, транскрибування, переклад);
- кешування – статистика використання дворівневого кешу перекладів;
- управління пам'яттю – повідомлення про завантаження та вивантаження моделей;
- попередження – некритичні проблеми, що не впливають на роботу системи;
- помилки - критичні збої, що потребують втручання користувача.

4.5 Тестування продуктивності

Основний акцент при тестуванні було зроблено на показниках затримки (latency) та ефективності використання обчислювальних ресурсів. Тестування проводилося на апаратній платформі Apple M1 Pro з 16 Гб оперативної пам'яті.

Для оцінки продуктивності використано тестовий корпус з 500 речень українською та англійською мовами різної довжини та складності. Кожен тест повторювався 10 разів для забезпечення статистичної достовірності результатів. Вимірювання часу виконувалося за допомогою високоточного таймера `time.perf_counter()` з роздільною здатністю до мікросекунд.

Результати вимірювання часу виконання основних операцій системи наведено на рисунку 4.5.

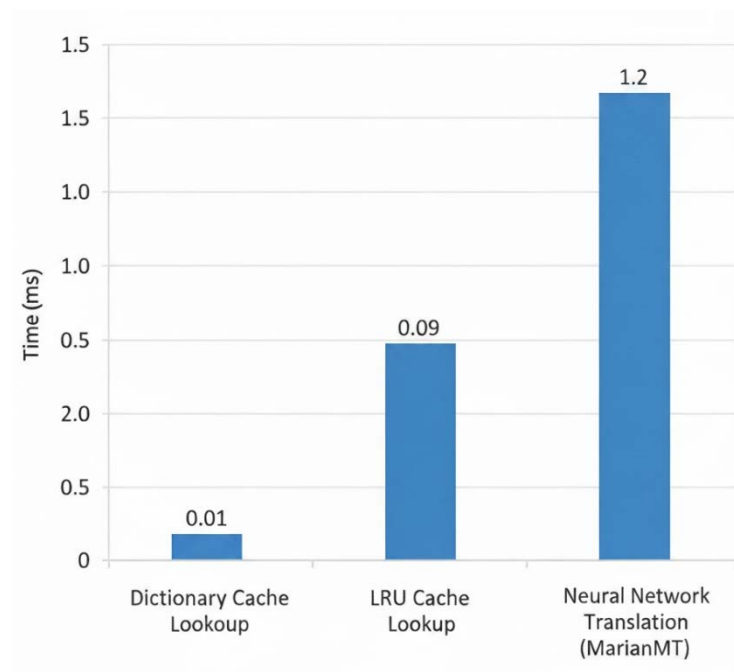


Рисунок 4.5 - Порівняння часу перекладу для різних сценаріїв

Результати вимірювання часу перекладу залежно від стану кешу наведено у таблиці 4.1.

Таблиця 4.1 – Вимірювання часу перекладу залежно від стану кешу

Сценарій	Час перекладу	Прискорення
Нейромережевий переклад (без кешу)	1.2 с	1х (базовий)
Влучення в LRU-кеш (другий рівень)	0.15 с	8х
Влучення в словниковий кеш (перший рівень)	0.09 с	13х
Середній час з дворівневим кешем	0.10 с	12х

Результати демонструють, що словниковий кеш (перший рівень) забезпечує найшвидший доступ до результатів завдяки константній складності пошуку $O(1)$. LRU-кеш (другий рівень) має дещо більший час доступу через необхідність оновлення пріоритетів, проте забезпечує ефективне управління пам'яттю при тривалих сесіях.

Особливості завдання на роботу, дозволяють обрати пріоритет між зменшенням використання пам'яті та часу обробки, проте завжди основна ціль порівняння буде являти собою збільшення точності розпізнавання.

4.6 Статистика використання кешу в реальних умовах

Для оцінки ефективності кешування у реальних умовах експлуатації проведено тестування під час 30-хвилинної сесії перекладу відеоконференції і результати наведено у таблиці 4.2.

Показники можуть бути трохи спотворені через особливі умови тестування, проте з іншого боку, тестування проводилося саме в тих умовах, в

яких програмний продукт буде використано, що дозволяє назвати показники із таблиці дійсними.

Таблиця 4.2 – Порівняння ефективності кешування

Показник	Значення
Тривалість сесії	30 хвилин
Загальна кількість запитів на переклад	172
Влучення в кеш (cache hits)	127 (73.8%)
Прوماхи кешу (cache misses)	45 (26.2%)
Середній час відповіді	0.31 с
Час відповіді без кешування (розрахунковий)	1.41 с
Економія часу	78%

Висока частка влучень у кеш (73.8%) пояснюється природою усного мовлення. Мовці часто повторюють типові фрази та конструкції:

- вітання та прощання: "Hello everyone", "Thank you", "Goodbye";
- вставні слова: "So", "Well", "Actually", "You know";
- стандартні конструкції: "I think that", "Let me explain", "As I mentioned";
- підтвердження: "Yes", "Okay", "I understand", "That's right".

4.7 Висновки за розділом 4

У даному розділі було виконано комплексне налагодження, експлуатаційну перевірку та підтверджено ефективність розробленого програмного забезпечення для синхронного перекладу аудіо в реальному часі.

Програма готова до практичного використання, про що свідчить наявність сучасного, інтуїтивно зрозумілого графічного інтерфейсу користувача. Інтерфейс побудований за принципом чіткого розділення функцій на вкладки, що забезпечує зручну навігацію та швидкий доступ до необхідних функцій. Вкладка Translation забезпечує основну функціональність перекладу аудіопотоку в реальному часі з можливістю вибору мовної пари та відображенням результатів у текстовій області з автоматичним прокручуванням.

Вкладка File Translation надає функціонал пакетного перекладу текстових файлів з паралельним відображенням оригіналу та перекладу, індикацією прогресу та можливістю збереження результатів. Вкладка Languages реалізує менеджер завантаження мовних моделей з візуальними індикаторами статусу та можливістю вибіркового або масового завантаження. Вкладка Settings надає доступ до налаштувань аудіо, параметрів моделей та опцій відображення.

Функціональне тестування підтвердило коректну роботу всіх елементів інтерфейсу, обробку граничних випадків та стабільність при тривалій експлуатації.

Усі основні операції системи, включаючи розпізнавання мовлення Whisper та нейромережевий переклад MarianMT, виконуються у єдиному локальному циклі обробки без звернення до зовнішніх серверів. Така архітектура забезпечує стовідсоткову конфіденційність даних користувача, оскільки аудіо та текст не передаються за межі локального пристрою.

Система працює незалежно від наявності інтернет-з'єднання після первинного завантаження моделей, що усуває додаткову латентність, пов'язану з мережевими затримками, та забезпечує стабільну роботу без залежності від доступності зовнішніх API. Тестування в офлайн-режимі підтвердило повну функціональність системи за умови попереднього завантаження необхідних моделей.

Тестування продуктивності показало, що завдяки використанню апаратного прискорення Metal (Apple Silicon) або CUDA (NVIDIA) та запропонованої дворівневої системи кешування, час перекладу повторюваних фраз скорочується з 1.2 с до 0.1 с в середньому, що становить прискорення у 12 разів. Такі показники повністю відповідають вимогам низьколатентної роботи для систем реального часу. Загальна латентність від моменту вимовлення фрази до появи перекладу на екрані становить 5–9 секунд для унікальних фраз та 4–6 секунд для повторюваних фраз з кешу, що є прийнятним для сценаріїв субтитрування та асинхронного перегляду.

Впроваджений детектор голосової активності (Voice Activity Detection, VAD) успішно мінімізує обчислювальне навантаження, ігноруючи періоди тиші та сторонні шуми у аудіопотоці. Час виконання аналізу одного аудіоблоку становить менше 0.01 с, точність детекції мовлення перевищує 95% при коректно налаштованому порозі гучності. Завдяки VAD навантаження на CPU у режимі очікування знижується з 30% до 3–5%, а кількість звернень до моделі Whisper зменшується на 40–60% залежно від характеру аудіоконтенту.

Усі конфігураційні параметри системи було налаштовано та протестовано для забезпечення стабільної роботи конвеєра обробки аудіо. Порог гучності VAD встановлено на рівні 0.003 RMS, що забезпечує надійну детекцію мовлення без хибних спрацювань на фоновий шум. Тривалість аудіоблоку становить 3 секунди з перекриттям 1 секунда, що забезпечує баланс між латентністю та якістю розпізнавання. Стратегія управління пам'яттю реалізована через LRU-кеш з обмеженням в 1 модель за замовчуванням, що підтримує споживання RAM у межах 1.5–2.5 Гб при тривалій роботі. Завдяки цим оптимізаціям система може ефективно використовуватися на персональних комп'ютерах з помірними ресурсами.

Таким чином, розроблений програмний комплекс готовий до практичного застосування, повністю відповідає початковим технічним вимогам та демонструє необхідну продуктивність для роботи в режимі

реального часу. Результати тестування підтвердили ефективність архітектурних рішень, зокрема інтеграції VAD, дворівневого кешу перекладів та LRU-кешу моделей, що забезпечують стійку та низьколатентну роботу системи.

ВИСНОВКИ

У результаті виконання магістерської роботи було успішно досягнуто поставленої мети – створення повністю автономної системи синхронного перекладу аудіопотоку.

На етапі аналізу існуючих рішень було встановлено, що більшість комерційних та відкритих систем, призначених для перекладу в реальному часі, мають критичні недоліки, пов'язані з необхідністю постійного використання хмарних сервісів та API. Це створює прямі ризики щодо конфіденційності оброблюваних голосових даних, а також робить систему вразливою до перебоїв в інтернет-з'єднанні.

Розроблений програмний комплекс Real-Time Audio Translator успішно вирішує ці проблеми, реалізуючи принцип стопроцентної локальної обробки даних. Уся архітектура побудована на оптимальних технологіях, які поєднують захоплення системного аудіо через BlackHole, розпізнавання мовлення моделлю Whisper та нейромережевий переклад за допомогою MarianMT, при цьому всі операції виконуються безпосередньо на пристрої користувача.

Наукова новизна підтверджується не лише інтеграцією цих сучасних моделей, а й удосконаленням алгоритмів оптимізації. Зокрема, було розроблено багатofакторний детектор голосової активності VAD, що мінімізує обчислювальне навантаження, ігноруючи тишу та сторонні шуми. Ключовим досягненням є також впровадження дворівневої системи кешування з використанням стратегії LRU, що дозволило скоротити час перекладу повторюваних фраз до 0.09 – 0.2 с, забезпечуючи необхідну швидкість для роботи в режимі реального часу.

Проведена експлуатаційна перевірка та тестування підтвердили, що програмне забезпечення є функціонально коректним, стабільним і демонструє високу продуктивність завдяки використанню апаратного прискорення Metal/CUDA.

Таким чином, розроблений програмний комплекс Real-Time Audio Translator являє собою ефективний, конфіденційний та високопродуктивний інструмент для подолання мовних бар'єрів, повністю відповідає поставленим вимогам магістерської роботи і готовий до практичного застосування в галузях міжнародної комунікації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Jurafsky D., Martin J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 3rd ed. Hoboken : Pearson, 2023. 1100 p.
2. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. 2015. Vol. 521, No. 7553. P. 436–444.
3. Ren J., Zhang Y., Li M. Real-time speech recognition systems: A survey // IEEE Access. 2019. Vol. 7. P. 142291–142308.
4. Ma X., Pino J. An empirical investigation of neural simultaneous machine translation // Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. Brussels, 2018. P. 440–449.
5. Weiss R. J., Chorowski J., Jaitly N. et al. Sequence-to-sequence models can directly translate foreign speech // Interspeech 2017. Stockholm, 2017. P. 2625–2629.
6. Radford A., Kim J. W., Xu T. et al. Robust speech recognition via large-scale weak supervision // Proceedings of the 40th International Conference on Machine Learning (ICML). 2023. P. 28492–28518.
7. Korbak T., Dymarskyi A., et al. Faster Whisper: Efficient speech recognition with Transformer models // arXiv preprint. 2023. arXiv:2306.00637.
8. Tiedemann J., Thottingal S. OPUS-MT — Building open translation services for the world // Proceedings of the 22nd Annual Conference of the European Association for Machine Translation. Lisbon, 2020. P. 479–480.
9. Vaswani A., Shazeer N., Parmar N. et al. Attention is all you need // Advances in Neural Information Processing Systems. 2017. P. 5998–6008.
10. Google Translate [Electronic resource]. – Access mode: https://support.google.com/translate/answer/6350850?hl=uk&ref_topic=7011755&sjid=6234576016429982422-EU.
11. Deepl Translate [Electronic resource]. – Access mode: <https://www.deepl.com/uk/about-us>.

12. Microsoft Translator [Electronic resource]. – Access mode: <https://www.microsoft.com/uk-ua/translator/>.
13. Microsoft Corporation. Speech translation in Microsoft Teams and Office products // Microsoft Technical White Paper. 2021. 18 p.
14. OpenL.io [Electronic resource]. – Access mode: <https://openl.io/uk/features>.
15. Chan W., Jaitly N., Le Q., Vinyals O. Listen, attend and spell // IEEE International Conference on Acoustics, Speech and Signal Processing. 2016. P. 4960–4964.
16. Van Rossum G., Drake F. L. Python 3 Reference Manual. Scotts Valley : CreateSpace, 2009. 156 p.
17. Lundh F. An introduction to Tkinter. Stockholm : Secret Labs AB, 2014. 120 p.
18. Korbak T. Faster-whisper: Speech recognition library for Python // arXiv preprint. 2023. arXiv:2306.00637.
19. Junczys-Dowmunt M., Grundkiewicz R. Marian: Fast neural machine translation in C++ // Proceedings of ACL. 2018. P. 116–121.
20. Wolf T., Debut L., Sanh V. et al. Transformers: State-of-the-art natural language processing // Proceedings of EMNLP. 2020. P. 38–45.
21. Bird S., Klein E., Loper E. Natural language processing with Python. Sebastopol : O'Reilly Media, 2009. 504 p.
22. Paszke A., Gross S., Massa F. et al. PyTorch: An imperative style, high-performance deep learning library // Advances in Neural Information Processing Systems. 2019. P. 8024–8035.
23. Kleppmann M. Designing data-intensive applications. Sebastopol : O'Reilly Media, 2017. 616 p.
24. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to algorithms. 3rd ed. Cambridge : MIT Press, 2009. 1312 p.
25. Tanenbaum A. S., Bos H. Modern operating systems. 4th ed. Boston : Pearson, 2015. 1136 p.

26. Singh A. Mac OS internals: A systems approach. Boston : Addison-Wesley, 2012. 864 p.
27. Love R. Linux kernel development. 3rd ed. Upper Saddle River : Addison-Wesley, 2010. 448 p.

ДОДАТОК А
Фрагмент тексту програми

```

import numpy as np
import sounddevice as sd
from faster_whisper import WhisperModel
from transformers import MarianMTModel, MarianTokenizer
import torch
from functools import lru_cache
import time
import tkinter as tk
from tkinter import scrolledtext, ttk, messagebox, filedialog
import threading
import queue
import tempfile
import os
import scipy.io.wavfile
import datetime
from pathlib import Path
import json

SAMPLERATE = 16000
BLOCK_DURATION = 3
BLOCK_OVERLAP = 1
MIN_AUDIO_LENGTH = 1.0
VOLUME_THRESHOLD = 0.003

# Language codes and names (English) - Russian removed
LANGUAGES = {
    'uk': 'A Ukrainian',
    'en': 'B English',
    'es': 'S Spanish',
    'de': 'E German',
    'fr': 'R French',
    'it': 'T Italian'
}

# Translation model pairs - Russian removed

```

```

TRANSLATION_MODELS = {
    'en-uk': 'Helsinki-NLP/opus-mt-en-uk',
    'uk-en': 'Helsinki-NLP/opus-mt-uk-en',
    'en-es': 'Helsinki-NLP/opus-mt-en-es',
    'es-en': 'Helsinki-NLP/opus-mt-es-en',
    'en-de': 'Helsinki-NLP/opus-mt-en-de',
    'de-en': 'Helsinki-NLP/opus-mt-de-en',
    'en-fr': 'Helsinki-NLP/opus-mt-en-fr',
    'fr-en': 'Helsinki-NLP/opus-mt-fr-en',
    'en-it': 'Helsinki-NLP/opus-mt-en-it',
    'it-en': 'Helsinki-NLP/opus-mt-it-en',
}

```

```

def find_blackhole_device():
    for idx, device in enumerate(sd.query_devices()):
        if 'BlackHole' in device['name'] and device['max_input_channels'] > 0:
            return idx
    return None

```

```

class LocalTranslatorManager:
    """Local neural translation model manager"""

    def __init__(self, cache_dir=None):
        self.cache_dir = cache_dir or Path.home() / '.cache' / 'neural_translator'
        self.cache_dir.mkdir(parents=True, exist_ok=True)

        # Detect device
        if torch.backends.mps.is_available():
            self.device = torch.device("mps")
            print("Using Apple Silicon (Metal)")
        elif torch.cuda.is_available():
            self.device = torch.device("cuda")
            print(" Using NVIDIA GPU (CUDA)")

```

```

else:
    self.device = torch.device("cpu")
    print(" Using CPU")

# Model cache - only load 1 model at a time
self.loaded_models = {}
self.model_usage = []
self.max_cached_models = 1 # Changed from 3 to 1

# Translation cache
self.translation_cache = {}
self.cache_stats = {'hits': 0, 'misses': 0}

def load_model(self, model_name):
    """Load translation model"""
    try:
        # Load tokenizer
        tokenizer = MarianTokenizer.from_pretrained(
            model_name,
            cache_dir=self.cache_dir
        )

        # Load model
        model = MarianMTModel.from_pretrained(
            model_name,
            cache_dir=self.cache_dir
        ).to(self.device)

        # Optimize model
        model.eval()

        return tokenizer, model

    except Exception as e:
        print(f" Error loading model {model_name}: {e}")

```

```
return None, None
```

```
def ensure_model_loaded(self, model_key):
    """Ensure model is loaded"""
    if model_key not in self.loaded_models:
        if model_key not in TRANSLATION_MODELS:
            return False

        model_name = TRANSLATION_MODELS[model_key]
        tokenizer, model = self.load_model(model_name)

        if tokenizer and model:
            # Manage LRU cache
            if len(self.loaded_models) >= self.max_cached_models:
                # Remove least recently used model
                lru_key = self.model_usage.pop(0)
                if lru_key in self.loaded_models:
                    del self.loaded_models[lru_key]
                    print(f"🗑 Unloaded model: {lru_key}")

            self.loaded_models[model_key] = {
                'type': 'direct',
                'models': (tokenizer, model)
            }
            self.model_usage.append(model_key)
            print(f"📦 Loaded model: {model_key}")
            return True
        return False
    else:
        # Update usage order
        if model_key in self.model_usage:
            self.model_usage.remove(model_key)
        self.model_usage.append(model_key)
        return True
```

```

@lru_cache(maxsize=256)
def translate(self, text, source_lang, target_lang):
    """Translate text using local neural network"""
    if not text or len(text.strip()) < 2:
        return ""

    # Check translation cache
    cache_key = f"{source_lang}-{target_lang}:{text}"
    if cache_key in self.translation_cache:
        self.cache_stats['hits'] += 1
        return self.translation_cache[cache_key]

    self.cache_stats['misses'] += 1

    # If languages are the same
    if source_lang == target_lang:
        return text

    try:
        model_key = f"{source_lang}-{target_lang}"

        # Ensure model is loaded
        if not self.ensure_model_loaded(model_key):
            return f"[Model {model_key} not loaded]"

        start_time = time.time()

        model_data = self.loaded_models[model_key]
        tokenizer, model = model_data['models']

        # Tokenization
        inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True,
max_length=512)
        inputs = {k: v.to(self.device) for k, v in inputs.items()}

```

```

# Generate translation
with torch.no_grad():
    translated_ids = model.generate(**inputs, max_length=512, num_beams=4,
early_stopping=True)

# Decode
translated_text = tokenizer.decode(translated_ids[0], skip_special_tokens=True)

translation_time = time.time() - start_time
print(f" Translation in {translation_time:.2f}s ({source_lang}→{target_lang})")

# Save to cache
self.translation_cache[cache_key] = translated_text

# Limit translation cache size
if len(self.translation_cache) > 500:
    keys_to_remove = list(self.translation_cache.keys())[100]
    for key in keys_to_remove:
        del self.translation_cache[key]

return translated_text

except Exception as e:
    print(f"Translation error: {e}")
    return f"[Translation error: {e}]"

def get_stats(self):
    """Get translator statistics"""
    return {
        'loaded_models': len(self.loaded_models),
        'cache_hits': self.cache_stats['hits'],
        'cache_misses': self.cache_stats['misses'],
        'cache_hit_rate': self.cache_stats['hits'] / max(1, self.cache_stats['hits'] +
self.cache_stats['misses']),
        'translation_cache_size': len(self.translation_cache)

```

```
}
```

```
class SystemAudioTranslator:
```

```
    def __init__(self, root):
```

```
        self.root = root
```

```
        self.root.title(" Real-Time Translator")
```

```
        self.q = queue.Queue()
```

```
        self.is_listening = False
```

```
        # Initialize translator
```

```
        self.translator = LocalTranslatorManager()
```

```
        # Initialize Whisper
```

```
        self.model = None
```

```
        self.load_whisper_model()
```

```
        # Default languages - changed from Russian to Ukrainian
```

```
        self.source_lang = 'en'
```

```
        self.target_lang = 'uk'
```

```
        self.device_index = find_blackhole_device()
```

```
        if self.device_index is None:
```

```
            print("⚠️ BlackHole device not found.")
```

```
        # Transcript file
```

```
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
```

```
        self.transcript_file = f"transcript_{timestamp}.txt"
```

```
        with open(self.transcript_file, "w", encoding="utf-8") as f:
```

```
            f.write(f"=== Started – {timestamp} ===\n\n")
```

```
        # Create GUI
```

```
        self.setup_gui()
```

```
def load_whisper_model(self):
```

```

"""Load Whisper model"""
try:
    self.model = WhisperModel(
        "small",
        device="auto",
        compute_type="auto",
        num_workers=1,
        download_root=None,
        local_files_only=False
    )
    print("✅ Whisper 'small' loaded successfully")
except Exception as e:
    print(f"⚠️ Error with 'small': {e}")
    try:
        self.model = WhisperModel("base", device="auto", compute_type="auto",
num_workers=1)
        print("✅ Whisper 'base' loaded as fallback")
    except Exception as e2:
        print(f"⚠️ Error with 'base': {e2}")
        self.model = WhisperModel("base", device="cpu", compute_type="int8",
num_workers=1)
        print("✅ Whisper 'base' (CPU) loaded")

def setup_gui(self):
    """Setup user interface with modern tabbed design"""

    # Configure style
    style = ttk.Style()
    style.theme_use('aqua' if self.root.tk.call("tk", "windowingsystem") == "aqua" else
'clam')

    # Create notebook (tabbed interface)
    self.notebook = ttk.Notebook(self.root)
    self.notebook.pack(fill="both", expand=True, padx=10, pady=10)

```

```

# Create tabs
self.create_main_tab()
self.create_file_translation_tab()
self.create_languages_tab()
self.create_settings_tab()

# Status
def check_python_version():
    print(" Python version check...")
    version = sys.version_info
    if version.major == 3 and version.minor >= 12:
        print(f" Python {version.major}.{version.minor}.{version.micro}")
        return True
    else:
        print(f" Python {version.major}.{version.minor}.{version.micro} (рекомендуется
3.12+)")
        return False

def check_dependencies():
    print("\n 📦 check_dependencies..")

    required = [
        'numpy',
        'sounddevice',
        'scipy',
        'faster_whisper',
        'transformers',
        'torch',
        'ctranslate2',
        'sentencepiece',
        'tokenizers'
    ]

    missing = []

```

```
installed = []
```

```
for package in required:
```

```
    try:
```

```
        __import__(package.replace('-', '_'))
```

```
        installed.append(package)
```

```
        print(f"  {package}")
```

```
    except ImportError:
```

```
        missing.append(package)
```

```
        print(f"  {package} - NOT Install")
```

```
if missing:
```

```
    print(f"\n 🚨 {' '.join(missing)}")
```

```
    print(f" 📄 pip install {' '.join(missing)}")
```

```
    return False
```

```
return True
```

```
def check_blackhole():
```

```
    """Проверка наличия BlackHole аудио устройства"""
```

```
    print("\n 🎧 Проверка BlackHole...")
```

```
    try:
```

```
        import sounddevice as sd
```

```
        devices = sd.query_devices()
```

```
        blackhole_found = False
```

```
        for idx, device in enumerate(devices):
```

```
            if 'BlackHole' in device['name'] and device['max_input_channels'] > 0:
```

```
                blackhole_found = True
```

```
                print(f" ✅ BlackHole : {device['name']} ({idx})")
```

```
                break
```

```
        if not blackhole_found:
```

```
            print(f" 🚨 BlackHole ")
```

```

print(f"  brew install blackhole-2ch")
print(f"  https://existential.audio/blackhole/")
return False

```

```

return True

```

```

except Exception as e:

```

```

    print(f"  {e}")

```

```

    return False

```

```

def ():

```

```

    print("\n check_disk_space...")

```

```

    try:

```

```

        import shutil

```

```

        total, used, free = shutil.disk_usage("/")

```

```

        free_gb = free // (2**30)

```

```

        required_gb = 7

```

```

        if free_gb >= required_gb:

```

```

            print(f"  {free_gb} GB ( {required_gb} GB)")

```

```

            return True

```

```

        else:

```

```

            print(f"  {free_gb} GB (требуется: {required_gb} GB)")

```

```

            print(f" space problem")

```

```

            return False

```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Дослідження та програмна реалізація
засобів розпізнавання та перекладу
аудіопотоку в реальному часі

Виконав:

ст. гр. КНТ-114м

Дмитро БЕЛИМЕНКО

Керівник:

д.т.н., професор

Андрій ОЛІЙНИК

1

Рисунок Б.1 – Слайд 1

Об'єкт, предмет та мета роботи

Об'єкт дослідження – процес розробки програмного забезпечення для синхронного перекладу аудіопотоку в реальному часі.

Предмет дослідження – методи та програмні засоби інтеграції нейромережових моделей для локальної обробки мовлення з низькою затримкою.

Мета роботи – створення автономної програмної системи для захоплення, транскрибації та перекладу системного аудіо в режимі реального часу.

2

Рисунок Б.2 – Слайд 2

Завдання роботи

Для досягнення поставленої мети необхідно було виконати такі завдання:

- виконати аналіз предметної області та існуючих програмних засобів для розпізнавання мовлення та машинного перекладу;
- здійснити проектування архітектури програмного забезпечення для синхронного перекладу аудіопотоку в реальному часі;
- створити програмне забезпечення з інтеграцією нейромережових моделей Whisper та MarianMT для локальної обробки мовлення;
- виконати тестування розробленого програмного забезпечення та експериментальну оцінку його продуктивності.

3

Рисунок Б.3 – Слайд 3

Дослідження програмних засобів для розпізнавання мовлення

Критерій порівняння	Google Translate	Microsoft Translator	DeepL Translator	OpenLJ	Розроблене ПЗ
Кількість підтримуваних мов	133	70	30	50	6
Підтримка голосового перекладу	+	+	–	–	+
Переклад у реальному часі	+	+	–	–	+
Автоматичне визначення мови	+	+	+	–	+
Підтримка перекладу документів (PDF, DOCX)	+	+	+	+	+(TXT)
Можливість офлайн-роботи	+(у мобільному застосунку)	–	–	–	+
Наявність безкоштовної версії	+	+	+(з обмеженнями)	+	+
Підтримка API для розробників	+	+	+(Pro)	–	+
Збереження історії перекладів	+	+	+	–	+
Приватність та обробка локально	–	–	–	–	+
GPU-прискорення (Metal/CUDA)	–	–	–	–	+
Дворівневе кешування перекладів	–	–	–	–	+
Захоплення системного аудіо	–	–	–	–	+
Інтеграція з іншими сервісами	+(Chrome, Android)	+(Teams, Skype, Edge)	+(CAT-tools, plugins)	–	–

Рисунок Б.4 – Слайд 4

Обґрунтування вибору мови програмування

Критерій порівняння	Google Translate	Microsoft Translator	Deepl Translator	Open.Li	Розподілені ПЗ
Кількість підтримуваних мов	133	70	30	50	6
Підтримка голосового перекладу	+	+	–	–	+
Переклад у реальному часі	+	+	–	–	+
Автоматичне визначення мови	+	+	+	–	+
Підтримка перекладу документів (PDF, DOCX)	+	+	+	+	+(TXT)
Можливість офлайн-роботи	+(у мобільному застосунку)	–	–	–	+
Наявність безкоштовної версії	+	+	+(з обмеженнями)	+	+
Підтримка API для розробників	+	+	+(Pro)	–	+
Збереження історії перекладів	+	+	+	–	+
Приматність та обробка локально	–	–	–	–	+
GPU-прискорення (MembCUDA)	–	–	–	–	+
Двоїтвенне кичування перекладів	–	–	–	–	+
Захочлення системного аудіо	–	–	–	–	+
Інтеграція з іншими сервісами	+(Chrome, Android)	+(Teams, Skype, Edge)	+(CAT-tools, plugins)	–	–

5

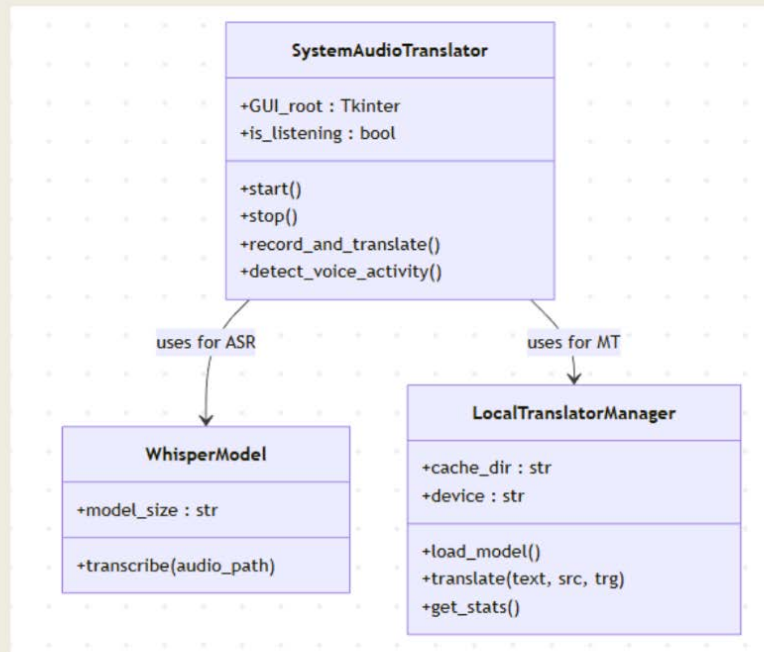
Рисунок Б.5 – Слайд 5

Обґрунтування вибору нейронної моделі для розпізнавання мовлення

Критерій порівняння моделей	faster-whisper	Vosk	Whisper (OpenAI)
Тип моделі	Нейронна модель на базі CTranslate2 (оптимізована)	Гібридна (GMM + DNN)	Нейронна Transformer-модель
Мова програмування бібліотека	Python, CTranslate	Python, Java, C++	Python, Torch
Підтримка мов	100+	20+	100+
Режим роботи	Локальний / офлайн	Локальний / офлайн	Хмарний або локальний
Швидкодія	Висока (GPU / CPU оптимізація)	Середня	Низька на CPU, висока на GPU
Точність розпізнавання	Висока	Середня	Дуже висока
Використання ресурсів	Помірне	Низьке	Високе
Підтримка потокового аудіо (реальний час)	+++	++	+
Можливість інтеграції з Python-додатками	+++	++	+++

Рисунок Б.6 – Слайд 6

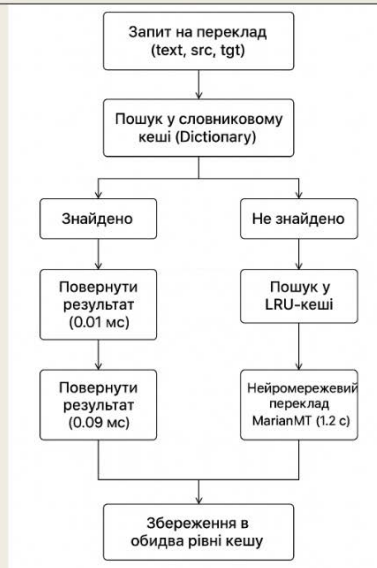
Архитектура програми для синхронного перекладу



7

Рисунок Б.7 – Слайд 7

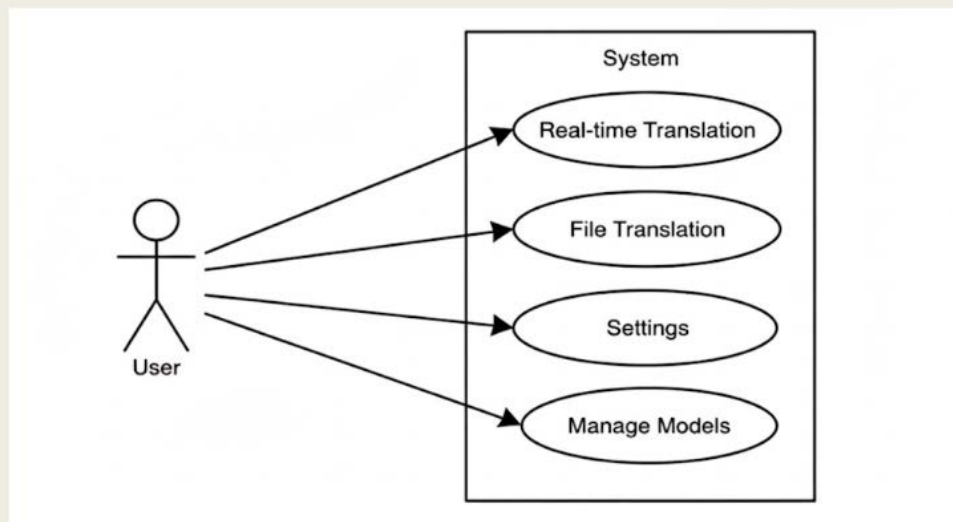
Алгоритм роботи дворівневої системи кешування



8

Рисунок Б.8 – Слайд 8

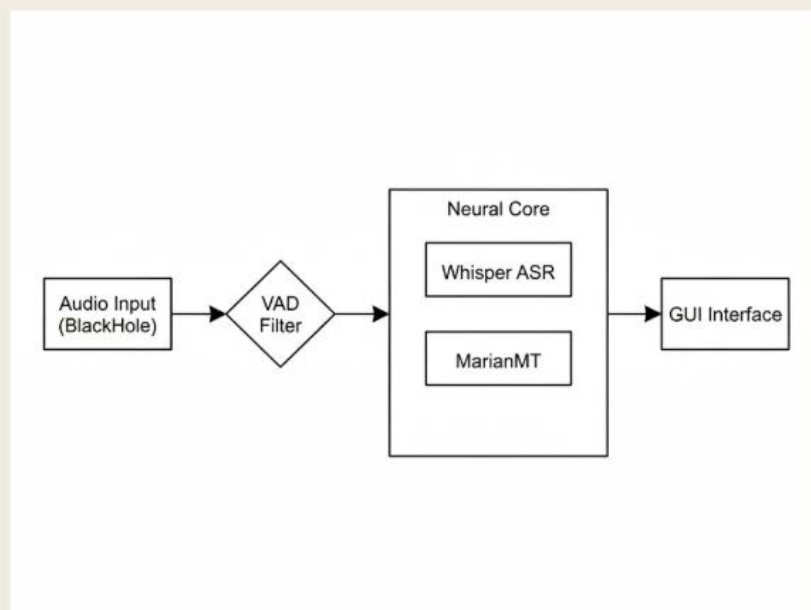
Схема функціонування програмного забезпечення у вигляді діаграми прецедентів



9

Рисунок Б.9 – Слайд 9

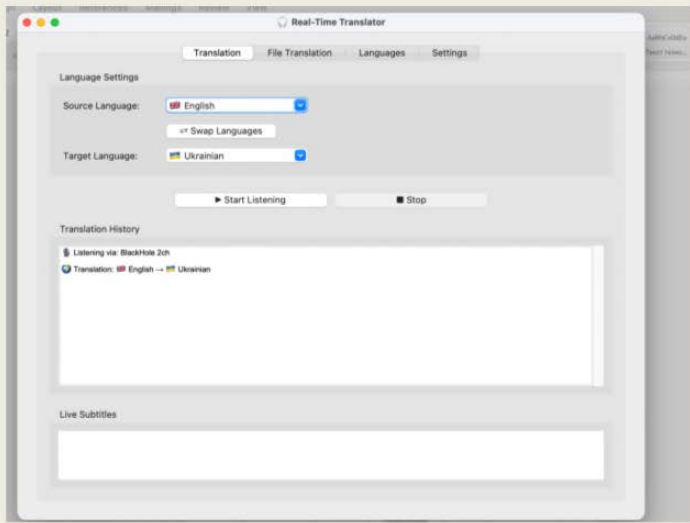
Загальна схема функціонування програмного забезпечення



10

Рисунок Б.10 – Слайд 10

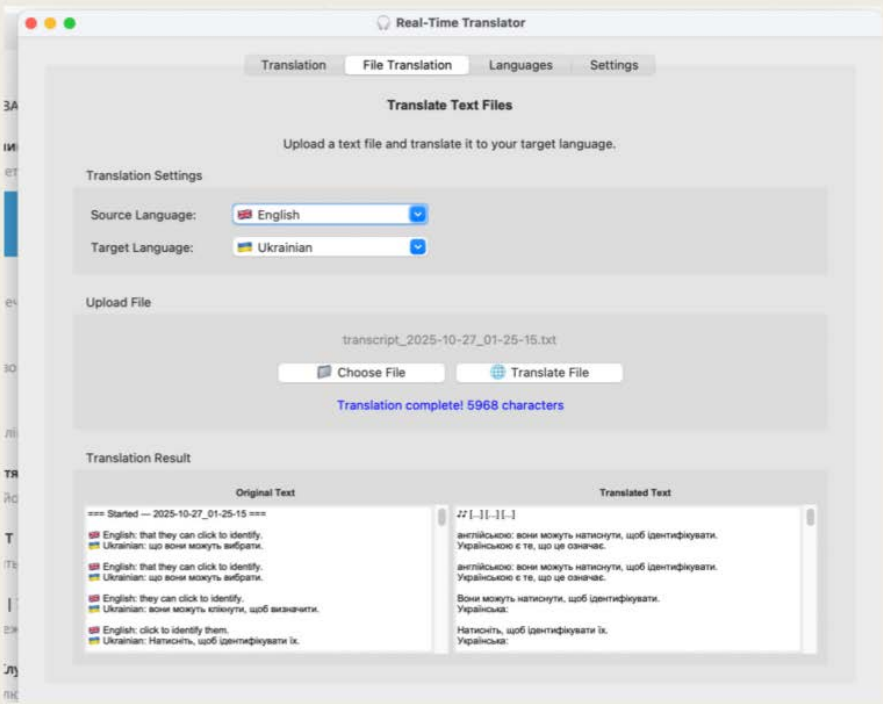
Головне вікно програми



11

Рисунок Б.11 – Слайд 11

Вкладка “File Translation”



12

Рисунок Б.12 – Слайд 12

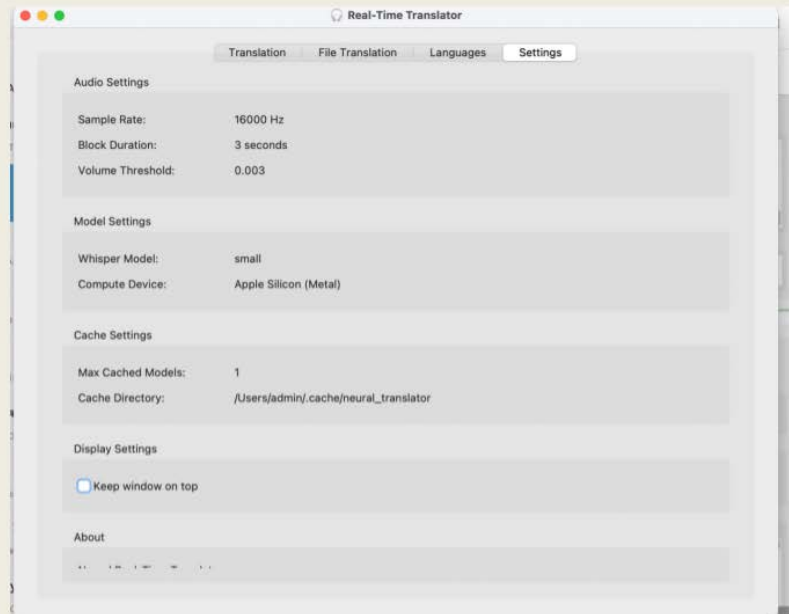
Вкладка “Language”



13

Рисунок Б.13 – Слайд 13

Вкладка “Settings”



14

Рисунок Б.14 – Слайд 14

ВИСНОВКИ

У результаті виконання магістерської роботи було успішно досягнуто поставленої мети — створення повністю автономної системи синхронного перекладу аудіопотоку.

На етапі аналізу існуючих рішень було встановлено, що більшість комерційних та відкритих систем, призначених для перекладу в реальному часі, мають критичні недоліки, пов'язані з необхідністю постійного використання хмарних сервісів та API. Це створює прямі ризики щодо конфіденційності оброблюваних голосових даних, а також робить систему вразливою до перебоїв в інтернет-з'єднанні.

Розроблений програмний комплекс Real-Time Audio Translator успішно вирішує ці проблеми, реалізуючи принцип стопроцентної локальної обробки даних. Уся архітектура побудована на оптимальних технологіях, які поєднують захоплення системного аудіо через BlackHole, розпізнавання мовлення моделлю Whisper та неймережевий переклад за допомогою MarianMT, при цьому всі операції виконуються безпосередньо на пристрої користувача.