Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

ВИКОРИСТАННЯ МАШИННОГО НАВЧАННЯ ДЛЯ МОДЕЛІ ОБРОБЛЕННЯ ЗАПИТІВ ГОЛОСОВОЇ ПОМІЧНИЦІ У СФЕРІ ГРОМАДСЬКОЇ ДІЯЛЬНОСТІ
(тема)

Виконала:
здобувачка 4 року навчання,
групи ІТІНФ-22-2
Єфімова В. В.
(прізвище, ініціали)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір С. В.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Єфімовій Валерії Вікторівні
(прізвище, ім'я, по батькові)

1. Тема роботи Використання машинного навчання для моделі оброблення запитів голосової помічниці у сфері громадської діяльності

затверджена наказом університету від 26 травня 2026 року № 435Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 19 червня 2026 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, технічна документація, бібліотека розпізнавання мовлення Vosk, бібліотека gRPC, FastAPI, scikit-learn, transformers, глосарії та тезауруси, джерела з проєктування голосових систем.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз голосових помічників та їхньої архітектури.
2. Проєктування та реалізація голосової помічниці для обробки питань, пов'язаних з громадською діяльністю.
3.Тестування, порівняння методів та оцінка ефективності системи.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми обробки зображень, постановка задачі, тестові дані.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	13.04.2026	
2	Аналіз завдання, підбір літератури	14.04.26-16.04.26	
3	Аналіз літератури з проблеми, що вирішується	17.04.26-19.04.26	
4	Аналіз існуючих методів розпізнавання	20.04.26-22.04.26	
5	Формування кроків вирішення проблеми	23.04.26-05.05.26	
6	Програмна реалізація	26.04.26-20.05.26	
7	Аналіз отриманих результатів	21.05.26-04.06.26	
8	Оформлення пояснювальної записки	05.06.26-09.06.26	
9	Перевірка на нормоконтроль	10.06.26-19.06.26	
10	Перевірка на плагіат	11.06.26-30.06.26	
11	Рецензування	20.06.26-30.06.26	
12	Підготовка презентації та доповіді	20.06.26-30.06.26	
13	Занесення роботи в електронний архів	30.06.26-09.07.26	
14	Попередній захист кваліфікаційної роботи	30.06.26-09.07.26	

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Машталір С. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 83 с., 9 табл., 30 рис., 1 дод., 31 джерело.

ГОЛОСОВІ ПОМІЧНИКИ, КЛАСИФІКАЦІЯ НАМІРІВ, МАШИННЕ НАВЧАННЯ, ОБРОБКА ПРИРОДНОЇ МОВИ, РОЗПІЗНАВАННЯ МОВЛЕННЯ, ASR, BERT, FASTAPI, GRPC, NLU, TF-IDF, WINUI 3.

Об’єктом роботи є процес голосової взаємодії користувача з інформаційною системою у сфері громадської діяльності.

Метою роботи є розроблення моделі обробки запитів користувача, що використовує сучасні методи машинного навчання для класифікації намірів у сфері громадської діяльності, а також створення програмного прототипу голосової помічниці для реалізації цієї моделі.

У межах роботи розглянуто підходи до створення інтелектуальних голосових помічників, можливості застосування методів TF-IDF та BERT у задачі класифікації намірів, визначено їхні переваги, недоліки та сфери доцільного використання.

Створено функціональний прототип голосової помічниці, який включає модуль автоматичного розпізнавання мовлення (ASR), модуль семантичного аналізу (NLU), клієнтський застосунок WinUI 3 та серверну частину на Python для обробки запитів користувача щодо термінів громадської діяльності. Взаємодію між компонентами реалізовано за допомогою gRPC та FastAPI, а для розпізнавання мовлення використано бібліотеку Vosk.

ASR, BERT, FASTAPI, GRPC, INTENT CLASSIFICATION, MACHINE LEARNING, NATURAL LANGUAGE PROCESSING, NLU, SPEECH RECOGNITION, TF-IDF, VOICE ASSISTANTS, WINUI 3.

The object of the work is the process of voice-based interaction between a user and an information system in the field of civic engagement.

The aim of the work is the development of a user query processing model that applies modern machine learning methods for intent classification in the domain of civic engagement, as well as the creation of a software prototype of a voice assistant.

The work examines approaches to building intelligent voice assistants, the applicability of TF-IDF and BERT methods to the intent classification task, and identifies their advantages, limitations, and appropriate use cases.

A functional prototype of a voice assistant was developed, including an ASR module, a NLU module, a WinUI 3 client application, and a Python-based server to assistant processes user queries related to terminology in the field of civic engagement. The system components interact via gRPC and FastAPI, while the Vosk library is used for real-time Ukrainian speech recognition.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Ембединги – вектор дробових чисел, що містить найважливіші ознаки слів

Фреймворк – програмний каркас (структурований набір інструментів і компонентів для розроблення програмного забезпечення)

API – Application Programming Interface (інтерфейс програмного застосунку)

ASR – Automatic Speech Recognition (розпізнавання мовлення)

BERT – Bidirectional Encoder Representations From Transformers (двонапрямні кодувальні представлення на основі трансформерів)

CPU – Central Processing Unit (центральний процесор)

DNN – Deep Neural Network (глибока нейронна мережа)

F1 – гармонійне середнє між точністю та повнотою

GPU – Graphics Processing Unit (графічний процесор)

gRPC – Google Remote Procedure Call (високопродуктивний фреймворк віддаленого виклику процедур)

HMM – Hidden Markov Model (прихована марковська модель)

LLM – Large Language Models (великі мовні моделі)

MFCC – Mel-frequency Cepstral Coefficients (коефіцієнти мел-частотного кепстру)

NLG – Natural Language Generation (генерація природномовних відповідей)

NLP – Natural Language Processing (обробка природної мови)

NLU – Natural Language Understanding (семантичне розуміння природної мови)

NGO – Non-Government Organisation (неурядова організація)

Q&A – Question And Answer (запитання та відповідь)

RAG – Retrieval-Augmented Generation (генерування, доповнене пошуком)

RMS – Root Mean Square (середньоквадратичне значення)

SRP – Single Responsibility Principle (принцип єдиної відповідальності)

STT – Speech-To-Text (перетворення мовлення на текст)

TF-IDF – Term Frequency-Inverse Document Frequency (частота терміна – обернена частота документа)

TTS – Text-To-Speech (перетворення тексту на мовлення)

UI – User Interface (користувацька оболонка)

ВСТУП

У сучасних умовах цифрової трансформації суспільства зростає потреба у зручних та доступних інструментах взаємодії з інформаційними системами. Особливої важливості це набуває у сфері громадської діяльності, де ефективна комунікація, доступ до достовірної інформації та швидке отримання відповідей на суспільно важливі питання є критичними для розвитку інклюзивності, правозахисних ініціатив та залучення громадян до соціальних процесів.

Попри наявність великої кількості цифрових ресурсів, користувачі часто стикаються з труднощами у пошуку необхідної інформації через складність термінології, фрагментованість джерел або низький рівень цифрової грамотності. Це створює бар’єри для участі у громадській діяльності, особливо для молоді, людей старшого віку та представників вразливих груп. У таких умовах голосові помічники стають перспективним інструментом, оскільки забезпечують природний спосіб взаємодії та знижують поріг входу до цифрових сервісів. Сучасні голосові системи активно розвиваються завдяки прогресу в галузі машинного навчання, автоматичного розпізнавання мовлення та обробки природної мови. Проте якість роботи таких систем значною мірою залежить від точності визначення наміру користувача. Невірна інтерпретація запиту призводить до некоректних відповідей і знижує ефективність помічника. Тому постає потреба у дослідженні різних підходів до класифікації намірів, зокрема порівнянні класичних статистичних методів і сучасних трансформерних моделей.

У межах даної роботи буде розглянуто сучасні підходи до створення голосових помічників та методи класифікації намірів користувача. Буде проаналізовано можливості застосування TF-IDF та BERT у задачах семантичної обробки тексту, а також буде визначено їхні переваги, недоліки та сфери доцільного використання.

Для практичної реалізації системи буде розроблено модуль автоматичного розпізнавання мовлення на основі технології ASR, буде

створено спеціалізований набір даних із термінологіями, необхідними для участі в громадській діяльності, інклюзії та гендерної рівності, а також буде реалізовано NLU-сервіс для класифікації намірів користувача.

Крім того, буде спроєктовано архітектуру програмної системи голосової помічниці, що включатиме клієнтський застосунок із використанням WinUI 3 (C#, XAML), серверну частину на Python та механізми взаємодії між компонентами за допомогою gRPC й FastAPI. Буде проведено експериментальне оцінювання ефективності моделей за метриками точності, F1-міри, часу обробки та матрицями плутанини, а також буде виконано порівняльний аналіз отриманих результатів.

Таким чином, робота спрямована на створення та дослідження інтелектуальної голосової системи, здатної забезпечити доступний та ефективний спосіб отримання інформації у сфері громадської діяльності, що є актуальним завданням у контексті розвитку цифрової інклюзії та громадянської участі.

1 ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Голосові помічники та їх архітектура

Голосові помічники або асистенти є програмними системами, що забезпечують взаємодію користувача з комп'ютером за допомогою природної мови. Вони широко застосовуються у мобільних пристроях, системах «розумного дому», інформаційних сервісах та інших сферах, де важливо забезпечити зручний та інтуїтивний інтерфейс взаємодії.

Перші голосові системи були побудовані на основі жорстко визначених правил (rule-based systems), де кожна команда відповідала конкретному шаблону [1]. Такі системи мали низьку гнучкість і не могли обробляти природні формулювання. Подальший розвиток став можливим завдяки появі статистичних методів ASR та NLP, що дозволило переходити від фіксованих команд до аналізу мовлення у вільній формі.

Сучасний етап розвитку пов'язаний із використанням глибоких нейронних мереж та трансформерних моделей. Помічники на кшталт Siri, Google Assistant, Alexa та сучасні LLM-асистенти (ChatGPT Voice, Gemini Voice, Copilot Voice) здатні розуміти контекст діалогу, підтримувати багатокрокові розмови, працювати з мультимодальними даними (текст, голос, зображення), адаптувати відповіді до користувача, виконувати складні запити [2].

Типова архітектура голосового помічника включає декілька основних компонентів: модуль розпізнавання мовлення (ASR), модуль обробки природної мови (NLU), модуль керування діалогом (Dialogue Management) та модуль генерації відповіді (NLG) [3].

Модуль ASR виконує перетворення аудіосигналу у текстове представлення. Сучасні ASR-системи використовують гібридні моделі (HMM + DNN), рекурентні мережі або трансформери (Whisper, DeepSpeech). До основних задач ASR входить сегментація аудіо, виділення акустичних ознак, визначення ймовірної послідовності слів, обробка шумів та акцентів.

Отримані дані передаються до модуля NLU, який аналізує його зміст та може включати такі підзадачі як: класифікація намірів (intent classification), вилучення сутностей (slot filling), визначення зв’язків між сутностями (relation extraction). Саме цей модуль забезпечує «розуміння» запиту системою. На основі цих даних модуль керування діалогом визначає логіку подальшої взаємодії, наприклад: вибір відповіді, керування станами діалогу, виклик зовнішніх API або сервісів, обробка помилок та уточнень. Після цього модуль NLG формує текст відповіді на основі наміру системи та може бути реалізований як шаблонна генерація, статистична модель або нейронна генеративна модель (GPT-подібні). Завершальним етапом є представлення отриманого результату користувачеві у вигляді тексту або природного мовлення [4]. Сучасні TTS-моделі (Tacotron 2, WaveNet) забезпечують високу природність голосу.

Як показано на рисунку 1.1, модуль NLU може включати підзадачі класифікації намірів, вилучення сутностей та визначення зв’язків між ними. Після аналізу запиту модуль керування діалогом (Dialogue management) визначає відповідну реакцію системи, а модуль генерації відповіді (NLG) формує текст, який передається до TTS або повертається у текстовому вигляді.

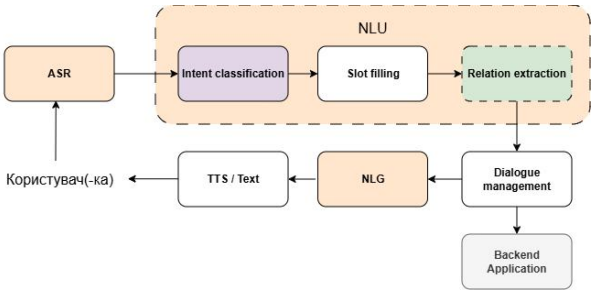


Рисунок 1.1 − Стандартна архітектура голосового помічника

Таким чином, голосові помічники стали повноцінними інтелектуальними системами, що поєднують розпізнавання мовлення, семантичний аналіз, генерацію відповідей та інтеграцію з зовнішніми сервісами. Класифікація намірів є ключовим етапом у роботі голосового помічника, оскільки саме вона

визначає подальшу поведінку системи та впливає на точність виконання користувацьких запитів.

1.2 Актуальність задачі для громадської діяльності

У сучасному суспільстві громадська діяльність відіграє важливу роль у формуванні демократичних процесів, забезпеченні прав людини та розвитку соціальної взаємодії. Ефективна комунікація між громадянами, неурядовими організаціями та державними структурами є ключовим фактором успішної реалізації соціальних ініціатив.

З розвитком цифрових технологій значна частина цієї взаємодії переходить в онлайн-середовище. Проте, незважаючи на велику кількість інформаційних ресурсів, користувачі часто стикаються з труднощами у пошуку необхідної інформації, особливо коли вона представлена у складній або неструктурованій формі. Це створює бар’єри для залучення громадян до суспільних процесів.

Одним із перспективних рішень цієї проблеми є використання інтелектуальних систем, зокрема голосових помічників, які забезпечують природну взаємодію з користувачем. Завдяки використанню обробки природної мови такі системи дозволяють формулювати запити у довільній формі та отримувати релевантні відповіді без необхідності навігації складними інтерфейсами.

Особливої актуальності набуває використання таких систем у сфері інклюзії та громадської активності, де важливо забезпечити доступність інформації для різних груп населення, включаючи людей з обмеженими можливостями або низьким рівнем цифрової грамотності. Голосові інтерфейси спрощують доступ до інформації, роблячи взаємодію інтуїтивною та універсальною. Завдяки цьому вони відіграють важливу роль у підвищенні

доступності цифрових сервісів для людей з інвалідністю, сприяючи формуванню більш інклюзивного середовища [5].

Водночас ключовою проблемою при створенні таких систем залишається точне визначення наміру користувача. Невірна інтерпретація запиту може призвести до надання некоректної або нерелевантної інформації, що знижує довіру до системи та її ефективність.

Додатково варто зазначити, що зростання обсягів інформації у сфері громадської діяльності створює потребу в інструментах, здатних швидко структурувати та пояснювати складні поняття у доступній формі. Голосові помічниці можуть виконувати роль посередника між громадянами та інформаційними ресурсами, забезпечуючи оперативний доступ до знань без необхідності спеціальної підготовки чи навичок роботи з цифровими платформами. Це особливо важливо для молоді, людей старшого віку та представників вразливих груп, які можуть відчувати труднощі у взаємодії з традиційними інформаційними системами. Інтелектуальні системи, що коректно визначають намір користувача, можуть стати важливим інструментом підтримки громадських організацій, допомагаючи автоматизувати відповіді на типові запити та зменшувати навантаження на волонтерів і консультантів.

З урахуванням вищезазначеного, дослідження методів підвищення точності класифікації намірів, зокрема шляхом використання сучасних моделей машинного навчання, є не лише технічною, а й соціально значущою, оскільки безпосередньо впливає на якість комунікації, доступність інформації та ефективність громадської взаємодії. Це дозволяє створювати більш ефективні та доступні системи для підтримки громадської діяльності.

1.3 Аналіз існуючих рішень у сфері громадської діяльності

Одним із ключових напрямів цифрових технологій є використання чат-ботів і голосових помічників для взаємодії з громадянами, поширення

інформації та залучення до суспільних процесів. Зростання кількості різних цифрових сервісів у громадській сфері формує потребу в інтелектуальних помічниках, які можуть спрощувати навігацію, пояснювати складні терміни та забезпечувати доступність інформації для широких груп населення [6].

Одним із найбільш поширених рішень є ШІ-чат-боти для державних сервісів, які забезпечують автоматизовану обробку запитів користувачів. Крім того, чат-боти широко застосовуються у сфері громадської участі. Наприклад, платформи на кшталт Polis [7] дозволяють збирати думки громадян та аналізувати їх за допомогою алгоритмів машинного навчання, формуючи узагальнені позиції суспільства щодо певних питань. Інші рішення, такі як Resistbot [8], автоматизують процес комунікації громадян із представниками влади, спрощуючи участь у демократичних процесах.

У діяльності неурядових організацій (NGOs) також активно використовуються ШІ-рішення. Такі системи дозволяють ефективніше обробляти великий обсяг звернень та забезпечувати персоналізовану взаємодію з користувачами.

Важливу роль у сучасних рішеннях відіграє обробка природної мови (NLP) [9], яка дозволяє системам розуміти запити користувачів у природній формі. Завдяки цьому чат-боти можуть не лише відповідати на питання, а й аналізувати настрої громадян, виділяти ключові проблеми та підтримувати прийняття управлінських рішень [10].

Окремим напрямом є створення спеціалізованих інформаційних чат-ботів, наприклад для пояснення державного бюджету або юридичних процедур. Такі системи поєднують методи пошуку інформації та генерації відповідей (RAG-підхід), що дозволяє підвищити точність і релевантність відповідей у вузькій предметній області.

Разом із перевагами існуючі рішення мають і низку обмежень. Основними проблемами є недостатня точність класифікації запитів у разі складних або неоднозначних формулювань; залежність від якості навчальних

даних; ризики упередженості алгоритмів; можливість генерації некоректних або недостовірних відповідей.

Нижче детально розглянемо, який функціонал пропонують вище згадані застосунки, зокрема Polis [7], Resistbot [8], Decidim [11], Дія [12], єВорог [13], GovBot [14] і запропонованого в даній роботі Voice Assistant (табл. 1.1).

Таблиця 1.1 − Порівняння існуючих систем у сфері громадської діяльності

Критерії	**Тип**	**Інтеракція**	**NLP**	**Персоналізація**	**Голос**
Polis	Платформа	Висока	Частково	Низька	Ні
Resistbot	Чат-бот	Середня	Так	Низька	Ні
Decidim	Платформа	Висока	Ні	Часткова	Ні
Дія	Сервіс	Середня	Ні	Висока	Ні
єВорог	Чат-бот	Низька	Ні	Низька	Ні
GovBot	Чат-бот	Середня	Так	Часткова	Ні
Запропо-нований Voice Assistant	Голосовий помічник	Висока	Так	Часткова	Так

Аналіз існуючих рішень показує, що більшість сучасних систем у сфері громадської діяльності орієнтовані на текстову взаємодію та виконують обмежений набір функцій, таких як надання довідкової інформації або збір зворотного зв’язку від користувачів. Хоча платформи типу Polis [7] забезпечують ефективний аналіз громадської думки, вони не підтримують природну діалогову взаємодію. У свою чергу, чат-боти, такі як Resistbot [8], спрощують комунікацію з органами влади, але мають обмеження у розумінні контексту та складних запитів.

Таким чином, існує потреба у створенні більш інтелектуальних систем, які поєднують голосову взаємодію, обробку природної мови та здатність до точного визначення намірів користувача.

1.4 Методи обробки природної мови у задачі класифікації намірів

Класифікація намірів користувача є однією з ключових задач обробки природної мови (NLP) у системах голосових помічників. Її основною метою є визначення семантичного змісту запиту та віднесення його до однієї з попередньо визначених категорій (класів), що дозволяє системі коректно реагувати на запити користувача. Однією з основних складових задачі класифікації намірів є представлення тексту у вигляді числових векторів, придатних для обробки алгоритмами машинного навчання. Такий процес називається векторизацією або побудовою ознак (relation extraction) [15].

У загальному випадку методи обробки тексту для задач класифікації намірів можна поділити на кілька основних груп:

– статистичні методи, які базуються на частотному аналізі слів (наприклад, Bag-of-Words, TF-IDF);

– методи на основі розподільних представлень (word embeddings), що враховують семантичну близькість слів (Word2Vec, GloVe);

– контекстні моделі, які формують представлення слів і текстів з урахуванням контексту (BERT, RoBERTa тощо).

Традиційні статистичні підходи, такі як TF-IDF, є відносно простими у реалізації та забезпечують високу швидкодію. Вони добре працюють у задачах із обмеженим словниковим запасом та чітко визначеними формулюваннями запитів, але не враховують контексту використання слів.

На відміну від цього, сучасні методи, засновані на нейронних мережах і трансформерних архітектурах, дозволяють отримувати більш точні семантичні представлення тексту. Контекстні моделі, зокрема BERT, здатні враховувати

залежності між словами у реченні та краще інтерпретувати зміст запиту, що підвищує якість класифікації намірів [16].

У задачах голосових помічників вибір методу обробки тексту залежить від ряду факторів, зокрема вимог до точності, швидкодії, обсягу доступних даних та обчислювальних ресурсів [17]. Тому доцільним є проведення порівняльного аналізу різних підходів з метою визначення оптимального рішення для конкретної предметної області.

У межах даної роботи для класифікації намірів використано два підходи: традиційний метод TF-IDF та сучасну трансформерну модель BERT, які розглянуто детальніше у наступних підрозділах.

1.4.1 Метод TF-IDF

TF-IDF є одним із базових методів представлення текстових даних у задачах обробки природної мови. Його основна ідея полягає у визначенні важливості слова в документі відносно колекції документів [18].

Значення TF-IDF для терміна t у документі d обчислюється як [19]:

$$TF - IDF(t,\ d) = TF(t,d) \bullet IDF(t), \tag{1.1}$$

де *TF(t, d)* − частота входження терміна у документ;

IDF(t) − обернена частота документів, у яких зустрічається термін.

Метод TF-IDF дозволяє перетворити текст у вектор числових ознак, що можуть використовуватися у моделях машинного навчання. У контексті громадської діяльності це особливо корисно для класифікації коротких запитів, визначення ключових слів у зверненнях громадян, аналізу коментарів або автоматичного групування схожих повідомлень.

Окрім цього, TF-IDF добре масштабується для великих корпусів текстів, що робить його придатним для систем та обробки значних обсягів звернень −

наприклад, у державних сервісах, волонтерських ініціативах або інформаційних платформах. Завдяки своїй простоті та швидкодії TF-IDF часто використовується як базовий метод у чат-ботах, де важливо забезпечити оперативну відповідь без значних обчислювальних витрат.

До переваг методу належать: простота реалізації, висока швидкість обробки, низькі обчислювальні витрати та інтерпретованість результатів. Вектор TF-IDF легко аналізувати, оскільки кожна його компонента відповідає конкретному слову, а значення відображає його важливість у тексті.

Основним недоліком TF-IDF є нечутливість до контексту, що особливо проблематично з огляду на притаманну природній мові багатозначність. Модель не розрізняє різні смисли слова залежно від синтаксичної ролі чи оточення, тому запити на кшталт «не люблю кішок» і «кішок не люблять» для неї виглядають подібними. Крім того, різні граматичні форми («люблю», «люблять») трактуються як окремі терміни [18]. Усе це обмежує здатність TF-IDF коректно інтерпретувати контекстно залежні висловлювання, що є критичним у задачах, де точність формулювань може суттєво впливати на зміст звернення.

1.4.2 Метод BERT

Сучасні підходи до обробки природної мови базуються на використанні нейронних мереж, зокрема трансформерних моделей. Архітектура трансформера базується на механізмі уваги, який дозволяє моделі враховувати взаємозв’язки між словами у реченні [19]. На відміну від рекурентних мереж, трансформери не обробляють текст послідовно, що забезпечує паралельність обчислень та ефективніше врахування довготривалих залежностей. Завдяки цьому трансформерні моделі здатні працювати з великими обсягами текстових даних та демонструють високу точність у широкому спектрі задач NLP.

Однією з найбільш відомих моделей є BERT. Особливістю BERT є двонаправлене навчання: модель аналізує слово одночасно з урахуванням як лівого, так і правого контексту. Це означає, що значення слова визначається не лише попередніми словами, а й тими, що йдуть після нього, що дозволяє точніше відображати семантику висловлювання.

Модель BERT формує контекстно залежні векторні представлення (ембединги) тексту, у яких одна й та сама лексема може мати різні вектори залежно від контексту [1, 17]. Це робить модель особливо ефективною для задач класифікації намірів, де важливо враховувати не лише окремі слова, а й їхнє значення у конкретному реченні. У таких задачах ембединги BERT можуть використовуватися для визначення найбільш подібного класу (домену) або для пошуку найближчого семантичного запиту.

До переваг BERT належать: врахування контексту, висока точність у широкому спектрі задач NLP, здатність працювати з різними формами мовних конструкцій, універсальність − модель успішно застосовується для класифікації тексту, семантичного пошуку, аналізу тональності, відповіді на запитання тощо.

Основними недоліками є значні обчислювальні витрати, потреба у більш потужному апаратному забезпеченні та більший час обробки порівняно з традиційними статистичними методами, такими як TF-IDF. Крім того, великі трансформерні моделі можуть вимагати оптимізації або кешування ембедингів для використання у системах реального часу.

1.5 Метрики оцінювання ефективності моделей

Для оцінювання якості моделей класифікації намірів використовуються стандартні метрики машинного навчання. Їхнє комплексне використання дозволяє оцінити не лише точність моделі, але й її стабільність, здатність працювати з різними класами, а також придатність для використання у системах реального часу.

Точність (accuracy) визначається як частка правильно класифікованих прикладів серед усіх об’єктів вибірки [20]. Вона дає загальне уявлення про якість роботи моделі, проте може бути недостатньо інформативною у випадку незбалансованих даних.

Точність позитивних прогнозів (precision) показує, яка частка об’єктів, віднесених моделлю до певного класу, дійсно належить до цього класу. Ця метрика є важливою у випадках, коли помилкові позитивні результати є критичними.

Повнота (recall) визначає, яка частка об’єктів певного класу була правильно виявлена моделлю [1, 20]. Вона характеризує здатність моделі знаходити всі релевантні приклади.

F1-міра є гармонійним середнім між точністю та повнотою, та дозволяє врахувати баланс між цими двома показниками [19, 20]. Вона є особливо корисною у задачах, де важливо одночасно мінімізувати як хибнопозитивні, так і хибнонегативні результати.

Окрім зазначених метрик, для детального аналізу якості класифікації використовується матриця плутанини, яка дозволяє оцінити розподіл правильних і неправильних передбачень для кожного класу окремо.

Для голосових помічників важливою є також швидкодія (latency) − час, необхідний для обробки запиту та формування відповіді.

Також у задачах багатокласової класифікації (як у випадку визначення намірів) використовуються узагальнені метрики macro- та micro-average.

Macro-average − обчислюється як середнє значення метрик для кожного класу, де усі класи мають однакову вагу. Micro-average − обчислюється глобально по всіх прикладах (враховує дисбаланс класів). Ці підходи дозволяють більш об’єктивно оцінити якість моделі у випадку нерівномірного розподілу даних [16].

У сучасних моделях, зокрема BERT, використовується також оцінка впевненості передбачення − ймовірність того, що запит належить до певного

класу. Цей показник дозволяє відфільтровувати ненадійні відповіді та реалізовувати механізми обробки невизначених запитів.

1.6 Постановка задачі

Розробка голосової помічниці для підтримки громадської діяльності є актуальним завданням у контексті цифровізації суспільних процесів та зростання потреби у зручних інструментах взаємодії між громадянами й інформаційними системами. У зв’язку з цим постає задача створення програмного застосунку Voice Assistant, здатного забезпечувати природну голосову взаємодію, коректне визначення намірів користувача та формування змістовних відповідей на запити. Практичне значення роботи полягає у створенні функціонального прототипу голосової помічниці, який може бути інтегрований у діяльність громадських організацій, інформаційно-просвітницьких ініціатив та сервісів підтримки користувачів.

Об’єктом роботи є процес голосової взаємодії користувача з інформаційною системою у сфері громадської діяльності.

Метою роботи є розроблення моделі обробки запитів користувача, що використовує сучасні методи машинного навчання для класифікації намірів у сфері громадської діяльності, а також створення програмного прототипу голосової помічниці для реалізації цієї моделі.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз сучасних підходів до побудови голосових помічників та методів класифікації намірів;

– дослідити можливості використання методів TF-IDF та BERT у задачах обробки природної мови;

– проаналізувати існуючі цифрові рішення та чат-боти, що застосовуються у сфері громадської діяльності;

– спроєктувати архітектуру програмної системи голосової помічниці;

– реалізувати модуль розпізнавання мовлення на основі технології ASR;

– підготувати набір даних для навчання моделей, що охоплює термінологію у сфері громадської діяльності (зокрема з питань гендерної рівності та інклюзії);

– реалізувати сервіс NLU для класифікації намірів користувача із застосуванням моделей машинного навчання;

– забезпечити взаємодію між компонентами системи за допомогою gRPC;

– реалізувати користувацький інтерфейс застосунку для взаємодії з голосовою помічницею;

– провести експериментальне оцінювання ефективності реалізованих моделей за метриками точності, F1-міри та швидкодії;

– виконати порівняльний аналіз отриманих результатів та сформулювати рекомендації щодо вибору моделі.

2 МАТЕМАТИЧНА МОДЕЛЬ ОБРОБЛЕННЯ ЗАПИТІВ ГОЛОСОВОЇ ПОМІЧНИЦІ

2.1 Формалізація задачі класифікації намірів

Задача класифікації намірів (intent classification) є ключовою складовою систем обробки природної мови та відіграє центральну роль у функціонуванні голосових помічників. Її метою є автоматичне визначення семантичного наміру користувача на основі введеного текстового або голосового запиту [16].

Нехай задано множину вхідних запитів користувача:

$$X = \{x_1, x_2, \dots, x_n\}, \tag{2.1}$$

де x_i − окремий текстовий запит.

Також визначено скінченну множину можливих намірів:

$$Y = \{y_1, y_2, \dots, y_k\}, \tag{2.2}$$

де кожен y_i відповідає певній категорії запитів (наприклад, визначення терміну, запит інформації тощо).

Тоді задача класифікації намірів полягає у побудові відображення:

$$f: X \to Y, \tag{2.3}$$

яке кожному запиту $x \in X$ ставить у відповідність найбільш імовірний намір $y \in Y$. У рамках статистичного підходу ця задача часто формулюється як задача максимізації умовної ймовірності [21]:

$$y^* = \underset{y \in Y}{arg\,max}\, P(y|x), \tag{2.4}$$

де $P(y|x)$ − ймовірність того, що запит x належить до класу y.

Оскільки алгоритми машинного навчання працюють з числовими даними, текстовий запит необхідно перетворити у формалізоване представлення. Для цього вводиться відображення:

$$g: X \to R^m, \tag{2.5}$$

яке переводить текст у вектор простору ознак розмірності m.

У загальному випадку простір можливих формулювань запитів є дуже великим, а самі запити можуть бути неоднозначними, неповними або містити синонімічні конструкції. З урахуванням цього, загальна модель класифікації набуває вигляду композиції двох функцій:

$$f(x) = h(g(x)), \tag{2.6}$$

де $g(x)$ − функція векторизації тексту;

$h(\cdot)$ − класифікатор, що визначає намір на основі векторного представлення.

Процес навчання моделі класифікації намірів полягає у мінімізації функції втрат [1]:

$$L = \sum_{i=1}^{n} l(y_i, \hat{y_i}), \tag{2.7}$$

де y_i − істинний клас;

$\hat{y_i}$ − передбачений моделлю клас.

У випадку TF-IDF використовується логістична регресія, тоді як у BERT-моделі класифікація здійснюється через пошук найближчого вектора за косинусною подібністю.

Таким чином, задача класифікації намірів у голосовій помічниці зводиться до побудови ефективного відображення текстових запитів у простір ознак та подальшого визначення найбільш ймовірного класу. Якість цього

процесу залежить як від способу представлення тексту, так і від обраного алгоритму класифікації, що обґрунтовує необхідність дослідження різних підходів, зокрема TF-IDF та BERT.

2.2 Математична модель представлення тексту

У задачі класифікації намірів одним із ключових етапів є перетворення текстового запиту користувача у числове представлення, придатне для подальшої обробки алгоритмами машинного навчання. Такий процес називається векторизацією тексту або побудової ознак [16, 19].

Нехай задано текстовий запит користувача $x \in X$, який складається з послідовності слів:

$$x = \{w_1, w_2, \ldots, w_n\}, \tag{2.8}$$

де w_i – окремі токени.

Перед векторизацією виконується попередня обробка тексту, яка включає токенізацію, нормалізацію та видалення стоп-слів. Токенізацією є процес розбиття тексту на елементарні одиниці (токени) [22]:

$$Tokenize(x) = [t_1, t_2, \ldots, t_m], \tag{2.9}$$

де t_i – токени, отримані з тексту.

У запропонованій системі використовуються два типи токенізаторів:

– класичний токенізатор TF-IDF, що розбиває текст за пробілами та пунктуацією;

– WordPiece-токенізатор BERT, який перетворює слова на підслова, забезпечуючи стійкість до рідкісних термінів.

Для ілюстрації процесу попередньої обробки тексту на рисунку 2.1 наведено приклад токенізації одного з користувацьких запитів, що демонструє відмінності між простим розбиттям на слова та підсловною сегментацією, характерною для трансформерних моделей. Наприклад, слово «гендерно-нейтральний» у разі простої токенізації може бути розбите на окремі частини: [гендерно,-,нейтральний]; тоді як BERT застосовує підсловну сегментацію (WordPiece), утворюючи такі токени: [гендер,#но,##нейтральний].

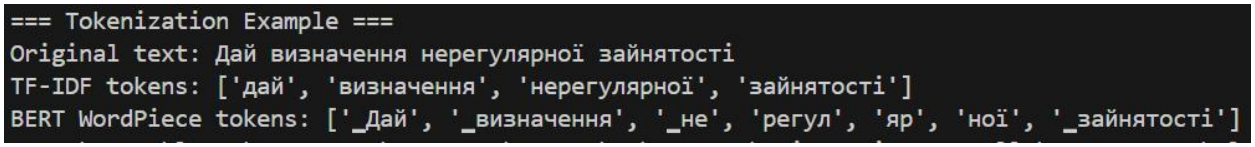


Рисунок 2.1 − Приклад токенізації текстового запиту за допомогою TF-IDF токенізатора та BERT WordPiece

Це дозволяє моделі коректно працювати з рідкісними або складними словами, що є важливим для предметної області, пов’язаної з інклюзією, гендерною рівністю та громадською діяльністю.

Також відбувається видалення стоп-слів, що не несуть суттєвого семантичного навантаження у задачі класифікації [22]. У розробленій системі додатково використовується множина стоп-слів:

$$S = \text{«ок», «ага», «ні», «гаразд», «так», «хах».} \tag{2.10}$$

Якщо запит складається лише зі стоп-слів або має довжину менше двох токенів, система повертає нульову впевненість:

$$x' \in S \cup \left|x'\right| < 2 \rightarrow f(x') = (\emptyset, 0). \tag{2.11}$$

Це дозволяє уникати хибних спрацьовувань на короткі або неінформативні фрази, що є критично важливим у голосових інтерфейсах.

2.3 Математична модель TF-IDF

Основна ідея методу TF-IDF полягає у визначенні важливості слова в документі відносно всієї колекції документів. Він дозволяє зменшити вагу часто вживаних слів та підсилити вагу рідкісних, що робить його ефективним для задач класифікації текстів [21]. Нехай задано корпус текстів:

$$D = \{d_1, d_2, \ldots, d_N\}, \tag{2.12}$$

де N − кількість документів (запитів користувача).

Для кожного слова t у документі d обчислюється показник *TF* (частота слова), який визначається як кількість входжень слова t у документ d [23]. Чим частіше слово зустрічається в документі, тим більш важливим воно вважається для опису його змісту:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}, \tag{2.13}$$

де $f_{t',d}$ − кількість входжень слова t у документ d.

Показник IDF (зворотна частота документа) відображає, наскільки рідкісним є слово в усій колекції документів. Він обчислюється на основі кількості документів, у яких зустрічається слово t [18]. Чим рідше слово зустрічається в усіх документах, тим більшу вагу воно отримує [24]:

$$idf(t) = log\frac{N}{df(t)}, \tag{2.14}$$

де $df(t)$ − кількість документів, у яких зустрічається слово t.

З урахуванням цих величин обчислюється TF-IDF. Цей показник дозволяє одночасно врахувати як важливість слова в конкретному документі, так і його поширеність у всій колекції [19]:

$$tfidf(t,d) = tf(t,d) \cdot idf(t). \quad (2.15)$$

Таким чином, кожен документ *d* представляється у вигляді вектора ознак, де кожна компонента відповідає значенню TF-IDF для певного слова. Отримане векторне представлення використовується як вхід для моделі класифікації намірів:

$$v_d = (tfidf(t_1,d), tfidf(t_2,d), ..., tfidf(t_m,d)), \quad (2.16)$$

де m − розмір словника.

Для кращого розуміння принципу роботи TF-IDF розглянемо приклад. Нехай документ d містить текст: «гендерна рівність важлива». У цьому документі три слова, і кожне з них зустрічається один раз. Тоді частота терміна (*TF*) для кожного слова становить:

$$tf(гендерна, d) = \frac{1}{3}, tf(рівність, d) = \frac{1}{3}, tf(важлива, d) = \frac{1}{3}. \quad (2.17)$$

Тепер припустимо, що корпус складається з 10 документів, і слово «гендерна» зустрічається лише у 2 документах з них. Тоді зворотна частота (*IDF*) для цього слова дорівнює:

$$idf(гендерна) = log\frac{10}{2} = log\,5\,. \quad (2.18)$$

Аналогічно, якщо слово «рівність» зустрічається у 5 документах, а «важлива» − у 8, то:

$$idf(рівність) = log\frac{10}{5} = log\,2\,, \quad (2.19)$$

$$idf(важлива) = log\frac{10}{8} = log\,1{,}25\,. \quad (2.20)$$

Тепер обчислимо TF-IDF для кожного слова:

$$tfidf(гендерна, d) = \frac{1}{3} log\, 5\,, \tag{2.21}$$

$$tfidf(рівність, d) = \frac{1}{3} log\, 2\,, \tag{2.22}$$

$$tfidf(важлива, d) = \frac{1}{3} log\, 1{,}25\,. \tag{2.23}$$

Отже, слово «гендерна» отримає найбільшу вагу, оскільки воно є рідкісним у корпусі, а слова «рівність» і «важлива» − меншу, бо зустрічаються частіше. Таким чином, документ представиться у вигляді вектора:

$$v_d = (\frac{1}{3} log\, 5\,,\, \frac{1}{3} log\, 2\,, \frac{1}{3} log\, 1{,}25\,), \tag{2.24}$$

де кожна компонента відповідає важливості відповідного слова.

TF-IDF дозволяє: підсилювати вагу рідкісних, але значущих термінів; зменшувати вплив часто вживаних слів; формувати інформативне векторне представлення документа.

2.4 Математична модель BERT

Модель BERT є сучасним підходом до обробки природної мови, що базується на архітектурі трансформерів та дозволяє отримувати контекстно залежні представлення тексту [18]. На відміну від статистичних методів, які розглядають слова незалежно, BERT формує щільні векторні представлення, що враховують взаємозв’язки між словами у всьому реченні.

Нехай задано вхідний текст:

$$x = \{\omega_1, \omega_2, ..., \omega_n\}, \tag{2.25}$$

де ω_i − токени, отримані після токенізації тексту (WordPiece).

У моделі BERT токени можуть відповідати як окремим словам, так і їх частинам (наприклад, «гендер», «##но»). Перед подачею в модель кожен токен перетворюється у вектор:

$$e_i = Embed(\omega_i), \tag{2.26}$$

де вектор включає токен-ембединг, позиційний ембединг та сегментний ембединг. Таким чином формується матриця вхідних представлень:

$$E = (e_1, e_2, \ldots, e_n). \tag{2.27}$$

Основою BERT є механізм багатоголової самоуваги (Multi- Head Self- Attention), який дозволяє моделі визначати, які слова є важливими одне для одного [18]. Для кожного токена обчислюються три вектори:

$$q_i = W_Q e_i, \tag{2.28}$$

$$k_i = W_K e_i, \tag{2.29}$$

$$v_i = W_V e_i, \tag{2.30}$$

де W_Q, W_K, W_V − матриці параметрів.

Вони дозволяють моделі враховувати контекст у всьому реченні, а не лише локальні залежності. Після проходження через l шарів трансформера формується матриця прихованих станів:

$$H = \{h_1, h_2, \ldots, h_n\}. \tag{2.31}$$

Для задачі класифікації використовується вектор спеціального токена CLS, який узагальнює інформацію про весь текст [20]:

$$h_{CLS} \in R^m, \tag{2.32}$$

де R^m − m-вимірний простір дійсних чисел, тобто числовий вектор довжини m (наприклад, $m = 768$ для базової версії BERT).

Далі застосовується класифікаційний шар. Намір визначається як той, для якого значення подібності є максимальним:

$$\hat{y} = softmax(W \cdot h_{CLS} + b), \tag{2.33}$$

де W − матриця ваг;

b − вектор зміщення;

$\hat{y}$ − вектор ймовірності класів.

Навчання моделі здійснюється шляхом мінімізації функції втрат крос-ентропії [9, 25]:

$$L = -\sum_{i=1}^{N} y_i log(p_i), \tag{2.34}$$

де y_i − фактична мітка (0 або 1);

p_i − передбачувана ймовірність для цієї мітки.

У даній роботі модель BERT використовується не як класичний класифікатор із softmax-шаром, а як модель семантичного пошуку (Sentence-BERT) [18]. Такий підхід широко застосовується у сучасних системах обробки природної мови, оскільки дозволяє ефективно визначати намір користувача на основі подібності векторних представлень. Спочатку формується векторне представлення v_q, що містить узагальнену інформацію про зміст запиту:

$$v_q = BERT(x). \tag{2.35}$$

Після отримання векторного представлення запиту виконується порівняння цього вектора з ембедингами всіх запитань, що містяться у навчальному наборі даних:

$$v_i = BERT(d_i). \tag{2.36}$$

Для визначення ступеня близькості між запитом користувача та кожним еталонним прикладом використовується косинусна подібність [22]:

$$cos(v_q, v_i) = \frac{v_q \cdot v_i}{\|v_q\|\|v_i\|}, \tag{2.37}$$

де значення косинусної подібності лежить у межах від -1 до 1, де 1 означає повну семантичну відповідність, а значення, близькі до нуля, свідчать про відсутність подібності [26].

Після обчислення подібності між вектором запиту та всіма векторами корпусу визначається найближчий намір. Це здійснюється шляхом вибору того прикладу, для якого значення косинусної подібності є максимальним:

$$\hat{i} = arg\max_i cos(v_q, v_i)\,. \tag{2.38}$$

Такий підхід дозволяє моделі коректно працювати навіть у випадках, коли формулювання запиту користувача суттєво відрізняється від формулювань у навчальних даних, але має подібний зміст.

2.5 Критерії оцінювання якості моделі

У межах даної роботи для порівняння моделей TF-IDF та BERT використовуються такі метрики: точність класифікації, F1-міра, час обробки запиту, оцінка впевненості та матриця плутанини. Оцінювання проходить методом навчання моделі на тестовій вибірці та передбачає попередній поділ даних на тренувальну та тестову вибірки [27].

Точність (*accuracy*) визначає частку правильно класифікованих прикладів серед усіх прикладів тестової вибірки. Формально воно обчислюється як:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.39}$$

де *TP* − істинно позитивні результати;

TN − істинно негативні;

FP − хибно позитивні;

FN − хибно негативні.

У задачах класифікації намірів точність є корисною метрикою, проте вона не завжди відображає реальну якість моделі, особливо якщо деякі класи представлені значно частіше за інші. У таких випадках більш інформативною є F1-міра, яка враховує баланс між влучністю (*precision*) та повнотою (*recall*), що враховує баланс між пропущеними та помилково визначеними класами [13]. Влучність та повнота визначаються як:

$$Precision = \frac{TP}{TP + FP}, \tag{2.40}$$

$$Recall = \frac{TP}{TP + FN}. \tag{2.41}$$

F1-міра є гармонійним середнім між влучністю та повнотою:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \tag{2.42}$$

Окрім якості класифікації, важливим показником є час обробки запиту, що вимірюється як різниця між моментом надходження запиту та моментом формування відповіді:

$$T = t_{response} - t_{request}\,. \tag{2.43}$$

У моделі BERT додатково використовується оцінка впевненості (*Confidence*).

Показник впевненості використовується для фільтрації нерелевантних відповідей та визначення порогу довіри:

$$Confidence(x) = \max_i cos(v_q, v_i), \tag{2.44}$$

де v_q – вектор запиту;

v_i – вектор i-го прикладу.

Для детального аналізу якості класифікації використовується матриця плутанини (confusion matrix). Матриця плутанини C визначається як квадратна матриця розмірності $k \times k$ де k – кількість класів. Елемент матриці C_{ij} показує, скільки разів модель передбачила клас j, коли істинним був клас i:

$$C_{ij} = |\{x : y_x = i \cap \hat{y}_x = j\}|. \tag{2.45}$$

Діагональні елементи матриці C_{ii} відповідають кількості правильно класифікованих прикладів класу i. Позадіагональні елементи $C_{ij,}\ i \neq j$ показують кількість помилок, коли клас i був неправильно віднесений до класу j [26]. Отримані матриці візуалізуються у вигляді теплових карт (heatmap), що дозволяє наочно оцінити, які саме наміри моделі плутають між собою, чи є класи, що систематично неправильно визначаються, та наскільки рівномірно модель працює з різними типами запитів.

2.6 Узагальнена математична модель системи

Голосову помічницю у межах даної роботи можна представити як композицію функцій, що перетворюють вхідний аудіосигнал у відповідь системи. Загальна модель системи має вигляд:

$$Y = F(A), \tag{2.46}$$

де A − аудіосигнал користувача;

Y − відповідь системи.

Аудіосигнал є неперервною функцією часу, оскільки фізичний звук існує у вигляді хвиль, що змінюються плавно, без стрибків. Математично це описується як функція:

$$A(t): R \to R, \tag{2.47}$$

де $A(t)$ − значення звукового тиску в момент часу t.

Однак комп'ютер не може працювати з неперервними величинами, тому аудіосигнал проходить дискретизацію − перетворення у послідовність чисел. Це робиться з певною частотою, наприклад 16 000 разів на секунду (16 кГц):

$$A = \{a_1, a_2, \ldots, a_N\}, \tag{2.48}$$

$$a_i = A(\frac{i}{f_s}), \tag{2.49}$$

де f_s − частота дискретизації (16 кГц).

Таким чином, аудіосигнал стає масивом чисел, кожне з яких описує амплітуду звуку у конкретний момент часу. Для подальшої обробки аудіосигнал розбивається на короткі вікна тривалістю 20-30 мс, у межах яких мовний сигнал вважається стаціонарним. Перетворення мовлення у текст відбувається за допомогою модуля розпізнавання мовлення (ASR):

$$X = f_{ASR}(A), \tag{2.50}$$

де X − текстовий запит.

Класифікація наміру в модулі обробки природної мови (NLU) за допомогою методів TF-IDF та BERT, і задається функцією:

$$y = f_{NLU}(X), \tag{2.51}$$

де $y \in Y_{intents}$ – передбачений намір користувача.

Генерація відповіді здійснюється за допомогою функції:

$$R = f_{resp}(y), \tag{2.52}$$

де R – текст відповіді, що відповідає наміру y;

f_{resp} – функція реалізована у вигляді словника відповідей.

Озвучення відповіді виконується за функцією:

$$S = f_{TTS}(R), \tag{2.53}$$

де S – аудіовідповідь системи.

Об’єднавши всі етапи, отримаємо повну модель системи:

$$S = f_{TTS}(f_{resp}(f_{NLU}(f_{ASR}(A)))). \tag{2.54}$$

Взаємодія між компонентами системи реалізована за допомогою протоколу gRPC, що забезпечує ефективний обмін даними між сервісами.

Для моделі BERT система додатково враховує поріг довіри [28, 29]:

$$y = \begin{cases} \arg\max P(y|X), & \text{якщо } confidence \geq \theta \\ y_{unknown}, & \text{інакше} \end{cases}, \tag{2.55}$$

де θ – поріг впевненості (у межах роботи він дорівнює $\theta = 0{,}75$);

$y_{unknown}$ – невизначений намір, у випадку якого система пропонує користувачу повторити запит.

Таким чином, запропонована узагальнена математична модель дозволяє формалізувати роботу голосової помічниці як послідовність взаємопов’язаних перетворень.

3 ПРОЄКТУВАННЯ ТА РОЗРОБЛЕННЯ ГОЛОСОВОЇ СИСТЕМИ

3.1 Специфікація вимог до застосунку

У рамках кваліфікаційної роботи був розроблений застосунок Voice Assistant для обробки голосових запитів. Розроблення будь-якого програмного забезпечення починається з формування чітких вимог до майбутньої системи, що дозволяє визначити функціональність, критерії якості та сценарії взаємодії користувача із застосунком. Щоб створити голосову систему, яка буде зручною, зрозумілою та корисною для користувача, необхідно заздалегідь визначити, які можливості вона буде надавати та якими характеристиками володіти.

Створимо діаграму варіантів використання (рис. 3.1) системи Voice Assistant, аби сформувати бачення системи з точки зору використання системи користувачем, а саме:

– початок або закінчення прослуховування;
– надання голосового запиту системі;
– отримання відповіді.

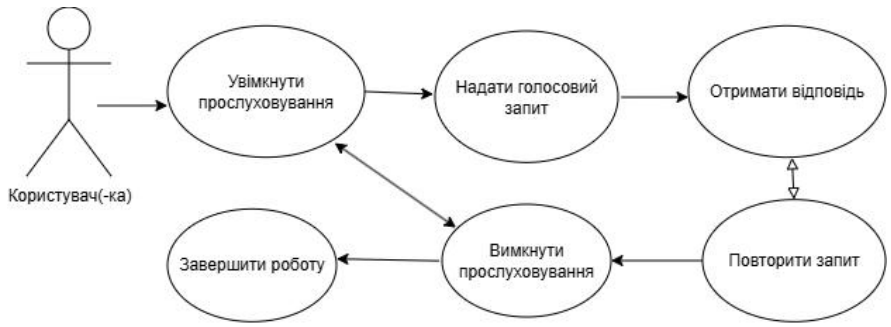


Рисунок 3.1 − Діаграма варіантів використання голосової системи Voice Assistant

На підставі проведеного аналізу існуючих голосових помічників, а також відповідно до цілей кваліфікаційної роботи, було сформовано перелік функціональних, системних та нефункціональних вимог до застосунку.

Системні вимоги:

– клієнт з .NET ≥ 8, XAML, підтримка мікрофона та аудіовиходу;

– сервер з Python ≥ 3,9 , gRPC, FastAPI, Vosk, NumPy, Docker.

Функціональні вимоги:

– забезпечувати запис аудіосигналу користувача з мікрофона;

– розпізнавати мовлення за допомогою аудіоданих;

– відображати текст розпізнаного запиту;

– відтворювати базові аудіовідповіді;

– виконувати класифікацію наміру за допомогою TF-IDF або BERT;

– визначати текст відповіді відповідно до визначеного наміру;

– відображати визначеного наміру та гучність;

– визначати рівень впевненості;

– обробляти випадки невизначеного наміру.

Нефункціональні вимоги до системи:

– система повинна підтримувати обробку запитів у реальному часу;

– точність розпізнавання мовлення повинна бути ≥ 85%;

– система повинна забезпечувати стабільність при повторних запитах;

– інтерфейс повинен бути інтуїтивно зрозумілим для користувача;

– інтерфейс повинен реагувати до 300 мс (запуск та зупинка прослуховування, реакція анімованої сфери);

– візуальні елементи повинні відображати стан системи (прослуховування, обробка, відповідь).

3.2 Обґрунтування вибору технологій

3.2.1 Мови програмування та середовище для розробки

Розроблення голосової системи Voice Assistant передбачає використання різних технологій для клієнтської та серверної частин, оскільки вони виконують різні функції та мають різні вимоги до продуктивності, інтеграції та

обробки даних. Перед початком виконання проєкту необхідно обрати технологічний стек, що забезпечить продуктивність, гнучкість та зручність у подальшій підтримці й масштабуванні системи.

Клієнтська частина застосунку реалізована з використанням WinUI 3 − сучасного фреймворку від Microsoft, що дозволяє створювати нативні Windows-застосунки з використанням XAML та .NET 8. На відміну від застарілого WPF або кросплатформеного MAUI, WinUI 3 забезпечує доступ до новітніх API Windows, підтримує апаратне прискорення рендерингу та дозволяє створювати інтерфейси, які відповідають сучасним стандартам дизайну Windows 11. Під час розроблення використовувалися бібліотеки, доступні через NuGet: NAudio забезпечив можливість запису та обробки аудіосигналу з мікрофона, Microsoft.Graphics.Win2D була застосована для рендерингу графічних елементів та анімацій. Для взаємодії клієнта із сервером застосовувався Grpc.Net.Client, що дозволило реалізувати передачу аудіоданих у режимі реального часу.

Серверна частина системи була реалізована мовою Python, яка є стандартом у сфері машинного навчання та обробки природної мови, адже забезпечує широкий вибір бібліотек для роботи з даними, побудови моделей та інтеграції алгоритмів класифікації. Для створення API та організації мікросервісної архітектури було використано FastAPI, який дозволяє обробляти запити з мінімальною затримкою та легко інтегрується з gRPC. У модулі розпізнавання мовлення застосовувалася бібліотека Vosk, яка забезпечує розпізнавання української мови. Для обробки текстових даних у NLU-модулі використовувалися NumPy та pandas, що дозволило ефективно працювати з векторами ознак та набором даних намірів. Алгоритми TF-IDF та BERT були реалізовані за допомогою scikit-learn, а для побудови графіків під час тестування застосовувалася бібліотека matplotlib.

Для наочності наведено порівняльні таблиці з альтернативними технологіями для розробки клієнтської (табл. 3.1) та серверної частини застосунку (табл. 3.2).

Таблиця 3.1 − Порівняння технологій для клієнтської частини

Технологія	**Продуктивність**	**Сучасність**	**Підтримка мультимедіа**	**Підтримка бібліотек**
WinUI 3 (.NET 8)	Висока	Сучасний стек Windows	Відмінна	Дуже широка
WPF (.NET Framework)	Середня	Застарілий	Обмежена	Широка
MAUI (.NET)	Середня	Новий, але нестабільний	Слабка	Обмежена
Electron (JS)	Низька	Залежить від браузера	Слабка	Немає
Qt (C++)	Дуже висока	Стабільний, але складний	Відмінна	Обмежена

Таблиця 3.2 − Порівняння технологій для серверної частини

Технологія	**Підтримка ML/NLP**	**Підтримка ASR**	**Продуктивність**	**Вимоги до ресурсів**
Python	Найкраща	Повна	Середня	Низькі
Node.js	Слабка	Немає	Висока	Низькі
Java	Середня	Часткова	Висока	Середні
Go	Дуже слабка	Немає	Дуже висока	Низькі
C++	Низька	Є, але складно	Максимальна	Високі

Таким чином, вибір WinUI 3 для клієнтської частини та Python із FastAPI, Vosk і scikit-learn для серверної частини забезпечив оптимальний баланс між продуктивністю, гнучкістю та простотою розробки. Обидві частини системи були побудовані на сучасних технологіях, які добре підтримуються та дозволяють розширювати функціональність у майбутньому.

3.2.2 Методи обробки тексту та передавання даних

Обробка тексту в системі Voice Assistant є ключовим етапом, що забезпечує коректне визначення наміру користувача та формування відповідної відповіді. У межах роботи було застосовано два різні підходи до побудови NLU-модуля: TF-IDF та BERT. Обидва методи мають різні властивості, вимоги до ресурсів та поведінку під час роботи з короткими голосовими командами, тому їхнє порівняння є важливим для обґрунтування вибору технологій.

Різні моделі суттєво відрізняються за точністю, швидкодією, здатністю враховувати контекст та вимогами до обчислювальних ресурсів, що особливо важливо для роботи в режимі реального часу. Деякі алгоритми забезпечують високу продуктивність, але не враховують семантику, тоді як інші демонструють кращу якість, проте потребують значно більше ресурсів.

Саме тому було розглянуто кілька альтернативних підходів до класифікації тексту, які потенційно могли бути використані в системі, та проведено їх порівняння за ключовими характеристиками (табл. 3.3).

Таблиця 3.3 − Порівняння варіантів класифікації тексту

Метод	**Точність**	**Швидкодія**	**Урахування контексту**	**Вимоги до ресурсів**
TF-IDF + Logistic Regression / SVM	Середня-висока	Дуже висока	Немає	Низькі
BERT + косинусна подібність	Висока	Середня	Є	Середні
Bag-of-Words + Naive Bayes	Низька	Дуже висока	Немає	Низькі
Fine-tuning BERT	Дуже висока	Низька	Є	Високі (GPU)
GPT-подібні моделі	Дуже висока	Дуже низька	Максимальне	Дуже високі

3.2.3 Технології передавання даних

Передавання даних між клієнтською та серверною частинами системи реалізовано за допомогою gRPC − протоколу від Google, який забезпечує низьку затримку, двосторонню передачу даних та серіалізацію даних у форматі Protocol Buffers. gRPC працює поверх HTTP/2. Це дозволяє передавати аудіосигнал у вигляді потоків невеликих фрагментів, що є критично важливим для роботи у режимі реального часу.

FastAPI базується на стандарті HTTP/1.1 та використовує REST, що забезпечує простоту інтеграції з клієнтськими застосунками та іншими мікросервісами. В роботі він використовується як серверний фреймворк, який забезпечує асинхронну обробку запитів, просту інтеграцію з мікросервісною архітектурою та можливість масштабування. У поєднанні з uvicorn це дозволяє обробляти кілька запитів одночасно та підтримувати стабільну роботу системи.

Для обґрунтування вибору протоколів нижче наведено порівняння альтернатив (табл. 3.4).

Таблиця 3.4 − Порівняння технологій передавання даних

Технологія	**Затримка**	**Підтримка потокової передачі**	**Серіалізація**	**Продуктивність**
gRPC	Дуже низька	Повна	Protocol Buffers (дуже ефективна)	Висока
REST	Висока	Відсутня	JSON (повільніша, об’ємніша)	Середня
WebSocket	Низька	Є	Залежить від реалізації	Середня
TCP	Низька	Є	Сирі байти	Дуже висока

Як видно з порівняння, gRPC забезпечує найкраще співвідношення швидкодії, ефективності серіалізації та підтримки потокової передачі даних. REST-API був відхилений через високу затримку та відсутність потокової передачі даних, WebSocket − через менш ефективну серіалізацію та складнішу підтримку, а TCP-сокети − через необхідність створення власного протоколу та значну складність реалізації.

3.2.4 Бібліотеки та моделі для розпізнавання мовлення

Розпізнавання мовлення є одним із ключових компонентів голосової системи, оскільки саме цей модуль забезпечує перетворення аудіосигналу на текст, який надалі обробляється NLU-сервісом. Для реалізації ASR-модуля було обрано бібліотеку Vosk, яка підтримує офлайн-розпізнавання, працює на широкому спектрі платформ та має готові моделі для української мови.

На відміну від хмарних сервісів, Vosk не потребує підключення до інтернету, що дозволяє забезпечити стабільну роботу системи навіть за умов обмеженого мережевого доступу, а також гарантує конфіденційність аудіоданих користувача. У межах проєкту використовувалася модель model-uk-v3-lgraph, яка є однією з найбільших і найточніших українських моделей у Vosk-екосистемі. Вона забезпечує високу якість розпізнавання, стійкість до фонового шуму та коректну обробку природного мовлення.

Хмарні сервіси, такі, як Azure Speech Services та Google Speech-to-Text, забезпечують високу точність, однак потребують стабільного інтернет-з’єднання, мають затримки, пов’язані з передаванням аудіо на сервер, та можуть створювати ризики для конфіденційності даних. DeepSpeech, попри відкритий код, демонструє нижчу якість розпізнавання української мови та потребує значних ресурсів для донавчання.

Нижче у таблиці 3.5 наведено порівняння основних альтернативних бібліотек і моделей розпізнавання мовлення.

Таблиця 3.5 − Порівняння бібліотек та моделей розпізнавання мовлення за характеристиками

Бібілотека	**Режим роботи**	**Підтримка української мови**	**Затримка обробки**	**Вартість**
Vosk	Офйлан	Є, якісна	Дуже низька	Безкоштовно
Google Speech-to-Text	Онлайн	Є, висока	Середня (мережеві затримки)	Платно
Azure Speech Services	Онлайн	Є, середня	Середня	Платно
Mozilla DeepSpeech	Офлайн	Часткова, слабка	Низька	Безкоштовно

У порівнянні з альтернативами Vosk виявився оптимальним рішенням завдяки поєднанню офлайн-роботи, високої точності, простоти інтеграції та доступності готових моделей.

3.3 Структура клієнт-серверної взаємодії

3.3.1 Основні компоненти системи

Голосова система Voice Assistant побудована за клієнт-серверною архітектурою. Така структура забезпечує розподіл відповідальності між компонентами, мінімізацію затримок, можливість паралельної обробки запитів та гнучке масштабування.

Клієнтська частина відповідає за взаємодію з користувачем, запис аудіоданих та відображення результатів, тоді як серверна частина виконує обчислювально складні операції-розпізнавання мовлення, класифікацію намірів та формування відповіді. Кожен компонент системи виконує окрему роль у

загальному процесі обробки запиту, що дозволяє підтримувати прозору та керовану структуру всієї системи.

Серверна частина складається з двох мікросервісів − NLU та ASR. ASR-сервіс виконує розпізнавання мовлення за допомогою, приймаючи аудіодані у потоковому режимі через gRPC. NLU-сервіс обробляє текстові запити, застосовуючи TF-IDF та BERT, визначає рівень впевненості та формує відповідь на основі внутрішнього набору даних.

Узагальнена структура компонентів системи наведена у таблиці 3.6, де показано відповідність між компонентами, використаними технологіями та їх функціональним призначенням.

Таблиця 3.6 − Відповідність функціоналу та компонентів системи

№	Компонент	Технологія	Функція
1	Модуль аудіозапису	NAudio (WinUI 3, .NET 8)	Запис аудіосигналу з мікрофона, передача даних у реальному часі.
2	Візуальний індикатор	WinUI 3, Win2D	Візуалізація гучності та активності прослуховування.
3	Інформаційна панель	WinUI 3, XAML	Відображення наміру, рівня впевненості, гучності, розпізнаного тексту та відповіді.
4	gRPC-клієнт	Grpc.Net.Client	Передавання аудіо на сервер, отримання результатів.
5	ASR-сервіс	Python, Vosk	Розпізнавання мовлення, формування partial/final результатів.
6	NLU-сервіс	Python, scikit-learn, pandas, transformers	Класифікація намірів, обчислення рівня впевненості та формування відповіді.
7	API-шлюз	FastAPI, uvicorn	Прийом запитів, маршрутизація між сервісами, повернення результатів.

3.3.2 Взаємодія між сервісами та обробка даних

Взаємодія між компонентами голосової системи Voice Assistant побудована на основі потокового обміну даними та мікросервісної архітектури, що забезпечує мінімальні затримки, паралельну обробку запитів та гнучкість масштабування.

Передавання даних між клієнтом і сервером здійснюється через двосторонню передачу даних gRPC, тоді як взаємодія між серверними мікросервісами реалізована за допомогою HTTP-запитів через FastAPI. Така комбінація дозволяє ефективно розподілити навантаження між ASR- та NLU-сервісами та забезпечити обробку аудіо й тексту в режимі реального часу.

Клієнтський застосунок встановлює двостороннє потокове з'єднання з ASR-сервісом відповідно до специфікації, визначеної у .proto-файлі. Кожен аудіофрагмент надсилається у вигляді повідомлення *AudioChunk.* ASR-сервіс отримує ці фрагменти послідовно, передає їх у *KaldiRecognizer* та формує два типи результатів:

– часткові результати − проміжні гіпотези розпізнавання, що дозволяють клієнту відображати текст у реальному часі;

– кінцеві результати − остаточний текст після завершення фрази.

Структура відповіді визначена у повідомленні *RecognizeResponse*, яке містить текст (*text*), намір (*intent*), відповідь (*answer*) та рівень впевненості (*confidence*).

Лістинг 3.1 Приклад кінцевого результату після повного розпізнавання та обробки NLU:

{"text": "Що таке громадський бюджет",

"answer": "Громадський бюджет − це інструмент участі громадян у розподілі коштів громади.",

"intent": "budget_definition",

"confidence": 0.91}

ASR-сервіс не зберігає стану між запитами, що відповідає принципам мікросервісної архітектури. Він виконує лише одну функцію − розпізнавання мовлення − і делегує семантичну обробку NLU-модулю. NLU-сервіс отримує текстові запити від ASR-модуля через FastAPI у форматі JSON.

Лістинг 3.2 Приклад повернення відповіді від NLU- до ASR-сервісу:

{"intent": "ask_definition",
"answer": "Цей термін означає...",
"confidence": 0.92}

Взаємодія між сервісами є асинхронною та незалежною та відбувається за наступною схемою:

Крок 1. ASR отримує аудіодані, обробляє їх та формує текст.

Крок 2. ASR надсилає текст у NLU через HTTP- запит.

Крок 3. NLU повертає намір, відповідь та впевненість.

Крок 4. ASR об’єднує ці дані у RecognizeResponse і надсилає клієнту.

Детальніше взаємодію корисувач − клієнт − сервер можна розглянути на відповідній діаграмі (рис. 3.2).

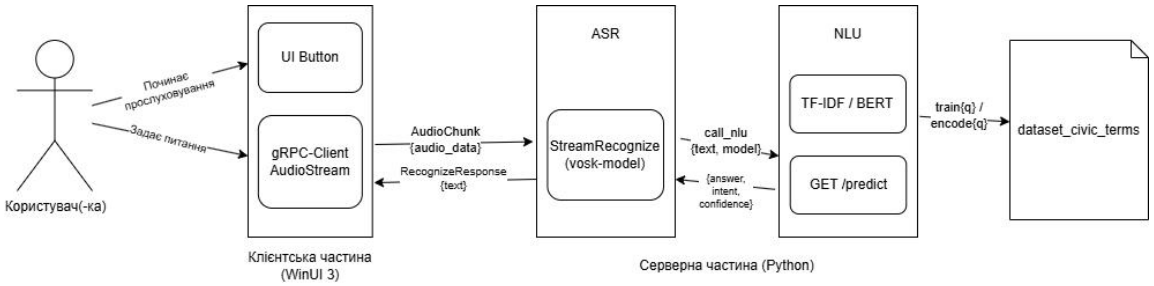


Рисунок 3.2 − Діаграма взаємодії між користувачем, клієнтом та сервером

Завдяки чіткому розподілу відповідальностей між компонентами, використанню двосторонню передачу даних gRPC та мікросервісній архітектурі система залишається масштабованою, стійкою до навантажень і здатною швидко реагувати на дії користувача.

3.3.3 Підготовка та структура набору даних

Для коректної роботи модуля NLU необхідним є наявність якісного набору даних, який містить запитання, відповіді та відповідні наміри. Під час дослідження не було знайдено готових наборів даних, що охоплювали б тематику громадської діяльності. Доступні відкриті дані, зокрема з порталу «Дія.Відкриті дані», містили лише сирі реєстрові записи про громадські організації з деяких регіонів України у форматі CSV − дані про які не є метою роботи. Тому було прийнято рішення сформувати власний набір даних обсягом приблизно 1000-2000 записів.

Основою для створення набору даних стали авторитетні джерела термінології, зокрема «Глосарій і тезаурус Європейського інституту з гендерної рівності» [30] та «Глосарій з гендерно чутливого відновлення» [31]. На основі цих джерел було сформовано перелік ключових термінів та їх визначень. Для кожного терміна створювалося кілька варіантів запитань у форматі Q&A, що дозволяє моделі краще розпізнавати різні формулювання одного й того ж наміру.

Наприклад, для терміна «інклюзія» було створено понад 5 варіантів запитань: «Що таке інклюзія?», «Поясни термін інклюзія», «Що означає інклюзія?» тощо. Це підвищує стійкість моделі до варіативності мовлення користувачів.

У результаті опрацювання термінологічних джерел та формування варіативних запитань було підготовлено власний набір даних *dataset_civic_terms* обсягом 1682 записи, що охоплює тематику громадської діяльності, інклюзії, гендерної рівності та безбар’єрності.

Набір даних було підготовлено у форматі CSV, де кожен запис містить такі поля:

– *id* − унікальний ідентифікатор;
– *question* − запитання у природній мові;
– *answer* − текст відповіді;

– *intent* − визначений намір (класифікаційна мітка);

– *category* − тематична категорія (наприклад: «соціальна політика», «права людини», «освіта»);

– *notes* − додаткові примітки (необов’язкове поле).

Такий формат забезпечує зручність подальшої обробки даних, можливість використання як статистичних методів (TF-IDF), так і контекстних моделей (BERT), а також дозволяє легко розширювати набір даних новими термінами та категоріями. Створення власного набору даних дало змогу адаптувати систему до специфіки громадської діяльності та забезпечити високу точність класифікації намірів у межах обраної тематики.

3.4 Реалізація клієнтської частини

Клієнтська частина програмної системи реалізована як настільний застосунок на платформі WinUI 3, що забезпечує сучасний інтерфейс користувача, високу продуктивність рендерингу та можливість гнучкого компонування елементів UI. Архітектура клієнта побудована відповідно до принципів чистого коду та розділення відповідальностей, що дозволило сформувати зрозумілу, підтримувану та масштабовану структуру застосунку.

Основна логіка клієнта реалізована мовою C#, тоді як інтерфейс користувача описано за допомогою XAML, що забезпечує декларативний підхід до побудови UI. Такий поділ дозволив чітко відокремити візуальну частину від бізнес-логіки та спростив подальше розширення функціональності.

Функціонально клієнт складається з таких компонентів:

– модуль захоплення та обробки аудіо (*AudioRecorder*), реалізований на основі бібліотеки NAudio;

– модуль мережевої взаємодії (*StreamingGrpcProcessManager*), побудований на основі Grpc.Net.Client;

– модуль візуалізації та анімації, реалізований за допомогою бібліотеки Microsoft.Graphics.Win2D;

– модуль управління станами (*VoiceRecognitionService*), який координує взаємодію між UI, аудіопотоком та мережею;

– модуль обробки результатів (*SpeechResultHandler*), який приймає часткові (*partial*) та кінцеві (*final*) результати розпізнавання, фільтрує дублікати, нормалізує текст та передає його до інтерфейсу користувача.

Взаємодія клієнта з сервером здійснюється через потоковий gRPC-виклик, де кожен фрейм містить масив байтів PCM- аудіо. Взаємодія з користувачем реалізована через інтуїтивний інтерфейс, що включає:

– кнопку керування увімкнення мікрофону;

– анімований індикатор голосової активності;

– панель відображення часткових і фінальних результатів, рівень гучності, класифікований намір, впевненість.

Завдяки використанню WinUI 3 та XAML інтерфейс є адаптивним, підтримує стилізацію та легко розширюється. Логіка обробки подій реалізована в окремих класах, що відповідає принципу єдиної відповідальності (SRP) та забезпечує чистоту коду.

3.4.1 Архітектура клієнтського застосунку та шаблони

Під час розробки клієнтської частини системи Voice Assistant було застосовано принципи чистого коду та об’єктно-орієнтованого проєктування, що забезпечило модульність, розширюваність і зручність підтримки програмного забезпечення. Архітектура застосунку побудована на чітко розмежованих відповідальностях між компонентами, використанні інтерфейсів та інверсії залежностей, що відповідає принципам SOLID.

Наприклад, клас *AudioRecorder* відповідає виключно за роботу з мікрофоном і запис аудіо, не містячи логіки обробки чи передачі даних.

Використанням інтерфейсів, таких як *IAudioPlayer*, дозволило додавати нові способи відтворення звуку без зміни існуючого коду. Використанно абстракції, наприклад *IProcessManager*, реалізації якого можуть взаємозамінюватися без впливу на роботу застосунку. Компонент *RecognitionManager*, що отримує *IRecorder*, *IProcessManager* та *IRecognitionStateManager*, не залежить від конкретних реалізацій.

Використано низку структурних і поведінкових шаблонів, які підвищили гнучкість архітектури. Шаблон Стану застосовано для керування станами розпізнавання (Idle, Listening, Processing). Шаблон Фасаду використано у класі *VoiceRecognitionService*. Шаблон Спостерігача в *IRecorder* та події *AudioCaptured*, а шаблон Стратегії використано в *VoiceOutputParser*, який інкапсулює *ITextParser* та *IVolumeParser*. Шаблон Компонувальник реалізовано в класі *SphereAnimator*.

Для кращого розуміння структури клієнтської частини та взаємозв’язків між її складовими було створено діаграму класів (рис. 3.3). Застосування принципів SOLID та шаблонів проєктування дозволило створити гнучку, модульну та легко підтримувану архітектуру клієнтської частини.

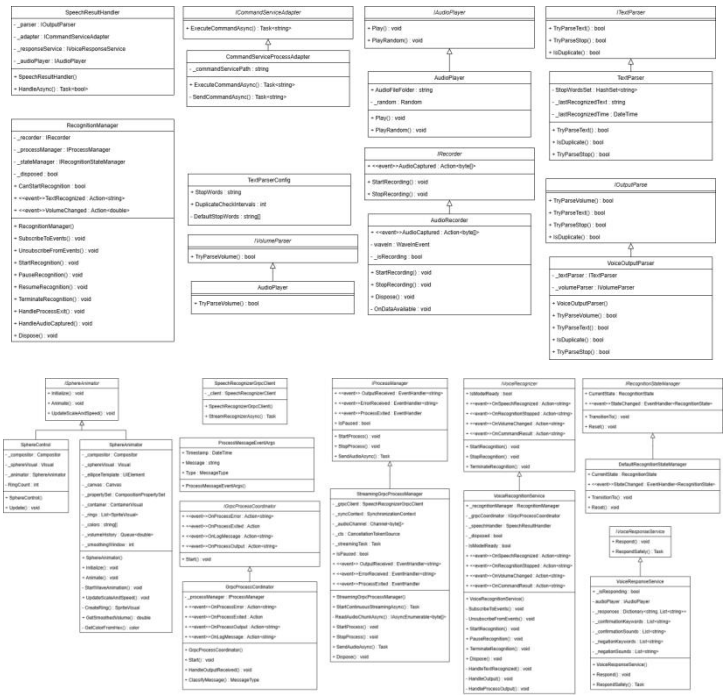


Рисунок 3.3 − Діаграма класів клієнтської частини

3.4.2 Модуль аудіозапису та обробки звуку

Модуль аудіозапису є одним із ключових компонентів клієнтської частини системи, оскільки саме він забезпечує захоплення голосового сигналу користувача та передачу його у потоковому режимі до серверного ASR-сервісу. У проєкті реалізація аудіозапису побудована на основі бібліотеки NAudio, яка надає низькорівневий доступ до мікрофона та дозволяє працювати з аудіопотоками у форматі PCM.

Модуль складається з двох основних частин: інтерфейсу *IRecorder*, який визначає контракт для будь-якої реалізації запису аудіо; та класу *AudioRecorder*, який реалізує цей контракт і відповідає за фактичне захоплення звуку.

Лістинг 3.3 Інтерфейс *IRecorder*:

```
public interface IRecorder
{event Action<byte[]> AudioCaptured;
void StartRecording();
void StopRecording();
double CalculateRMSVolume(byte[] audioData);}
```

Клас *AudioRecorder* реалізує логіку захоплення аудіо з мікрофона за допомогою *WaveInEvent*. Запис здійснюється у форматі 16 kHz, 16-bit PCM, mono − це стандартний формат для моделей розпізнавання мовлення, включно з Vosk. Під час запуску запису створюється екземпляр *WaveInEvent*, який генерує подію *DataAvailable* щоразу, коли доступний новий фрагмент аудіо та викликається подія *AudioCaptured*, яка передає аудіодані іншим модулям (наприклад, мережевому менеджеру *IProcessManager*).

Модуль аудіозапису працює у зв’язці з іншими частинами системи:

– *StreamingGrpcProcessManager* підписується на подію *AudioCaptured* і передає аудіофрейми на сервер;

– *VoiceRecognitionService* отримує значення гучності та оновлює UI;

– *SphereAnimator* використовує RMS для гучності та анімації сфери.

Окремим компонентом є *AudioPlayer*, який реалізує інтерфейс *IAudioPlayer* та відповідає за відтворення звукових файлів у відповідь на команди користувача або системні події. Модуль підтримує: відтворення конкретного файлу (*Play()*) та відтворення випадкового файлу зі списку (*PlayRandom()*).

Модуль аудіозапису забезпечує стабільне захоплення аудіосигналу, передачу аудіоданих у реальному часі, повну ізоляцію логіки запису від інших компонентів. Завдяки цьому клієнтська частина працює плавно, без затримок і забезпечує коректну взаємодію з сервером та інтерфейсом користувача.

3.4.3 Мережева взаємодія та передавання даних через gRPC

Мережева взаємодія між клієнтською частиною та сервером реалізована за допомогою технології gRPC, яка забезпечує двосторонній потоковий обмін даними (bidirectional streaming). Такий підхід дозволяє передавати аудіодані у режимі реального часу та отримувати часткові й фінальні результати розпізнавання без затримок, що є критично важливим для голосових систем.

Взаємодія визначена у файлі *speech.proto*, який описує: сервіс *SpeechRecognizer*, метод *StreamRecognize*, структуру вхідних та вихідних повідомлень. Метод *StreamRecognize* працює у режимі двосторонньої передачі даних, тобто клієнт надсилає потік аудіоданих, а сервер повертає потік відповідей.

Лістинг 3.4 gRPC-контракт у файлі .proto:

```
service SpeechRecognizer {rpc StreamRecognize(stream AudioChunk) returns
(stream RecognizeResponse);}
message AudioChunk { bytes audio_data = 1;}
message RecognizeResponse {
```

string text = 1; string answer = 2;

string intent = 3; float confidence = 4;}

Клас *SpeechRecognizerGrpcClient* реалізує інтерфейс *IProcessManager* та інкапсулює логіку взаємодії з сервером. Він відповідає за встановлення з'єднання через *GrpcChannel* та порт 50051, створення двостороннього передачи даних через метод *StartContinuousStreamingAsync*, передавання аудіофреймів, асинхронне читання відповідей сервера та обробку часткових та фінальних результатів.

Клас *GrpcProcessCoordinator* виконує роль посередника між мережевим менеджером та сервісом розпізнавання, що дозволяє відокремити мережеву логіку від бізнес-логіки розпізнавання. Його завдання: запуск процесу (*Start()*), обробка подій *OutputReceived*, *ErrorReceived*, *ProcessExited,* класифікація типу повідомлення (*Log, Error, Result*), передавання результатів у *VoiceRecognitionService*.

Для наочності послідовність взаємодії між компонентами клієнтської частини під час потокового передавання аудіоданих зображено на рисунку 3.4.

Рисунок 3.4 − Діаграма взаємодії між компонентами клієнтської частини під час потокового передавання аудіоданих

Через бінарний протокол, який використовується в gRPC надає низьке навантаження на мережу, клієнт може одночасно надсилати аудіо та отримувати відповіді, а сувора типізація через .proto збергіє безпеку та коректність даних.

3.4.4 Відображення результатів та логіка UI

Дизайн застосунку описується за допомогою XAML. Основним вікном взаємодії є клас *MainWindow*, який відповідає за ініціалізацію сервісів, підписку на події розпізнавання та оновлення елементів інтерфейсу користувача. Під час створення головного вікна відбувається ініціалізація всіх ключових компонентів та *MainWindow* підписується на події:

– *OnSpeechRecognized* − оновлює основний текстовий блок *myTextBlock* (рис. 3.5);

– *OnVolumeChanged* − оновлення *volumeTextBlock* (зміна гучності);

– *OnIntentRecognized* − оновлює *intentTextBlock* та *confidenceTextBlock*, відображаючи визначений намір та рівень впевненості (рис. 3.6);

– *OnAnswerReceived* − оновлює *answerTextBlock*, показуючи отриману відповідь від NLU-сервісу (рис. 3.7);

– *OnCommandResult* − повернення результату виконання команди;

– *OnRecognitionStopped* − завершення прослуховування.

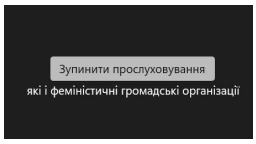


Рисунок 3.5 − Вивід розпізнаного тексту у реальному часі

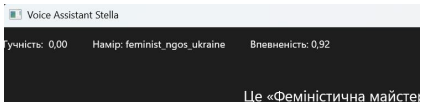


Рисунок 3.6 − Відображення наміру, впевненості та гучності

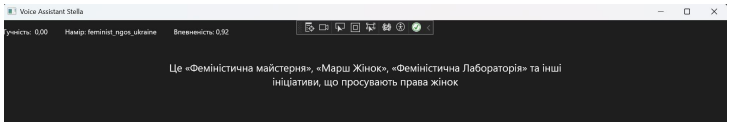
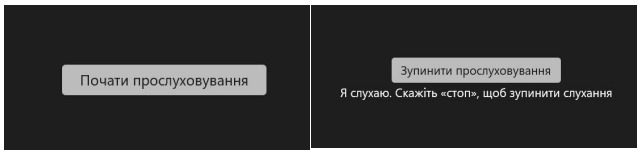


Рисунок 3.7 − Відображення відповіді від NLU-сервісу

Кнопка *myButton* змінює свій текст залежно від стану *isListening*. Якщо голосова помічниця слухає запитання, то відображається текст «Зупинити прослуховування», якщо ні − «Почати прослуховування» (рис. 3.8). При натисканні: запускається або зупиняється розпізнавання, оновлюється текст підказки, змінюється стан *isListening*.



(а) (б)

Рисунок 3.8 − Варіанти відображення кнопки *myButton*:
(а) стан *isListening = False*, система не почала слухати запити; (б) стан *isListening = True*, система слухає користувача

Таким чином, логіка UI побудована за реактивним принципом, де усі зміни інтерфейсу ініціюються подіями, що надходять від етапів розпізнавання.

3.4.5 Анімована сфера як індикатор голосової активності

Окрім текстових елементів, важливою частиною інтерфейсу є анімаційна сфера, яка візуально відображає рівень голосової активності користувача, що створює ефект «живого» індикатора, подібного до голосових помічників Siri або Google Assistant. Вона реалізована у вигляді окремого компонента *SphereControl*, який містить:

– *Canvas SphereCanvas* − контейнер для рендерингу графічних елементів;
– *EllipseTemplate* − шаблон елемента, з якого створюються кільця;
– *SphereAnimator* − клас анімаційного рушія.

Клас *SphereAnimator* відповідає за створення візуальних елементів (кілець сфери) через *Initialize(count)*, застосування кольорових ефектів

(*ColorMatrixEffect)*, запуск хвильової анімації через метод *Animate()*, зміну масштабу та швидкості залежно від гучності за допомогою expression-анімацій в методі *UpdateScaleAndSpeed(volume)*, згладжування значень RMS-гучності через метод *GetSmoothedVolume()*.

Лістинг 3.5 Фрагмент expression-анімації, що визначає коливання елементів сфери у відповідь на голосову активність:

```
var exprX = _compositor.CreateExpressionAnimation("1f +
source.waveAmplitude * sin((source.time * source.waveSpeedMultiplier +
delay) * 6.28f)");
```

Чим голосніший звук − тим більша сфера, а саме масштабування: амплітуда (*waveAmplitude*) та швидкості хвиль (*waveSpeedMultiplier*) зростає.

У *MainWindow* сфера оновлюється через подію *OnVolumeChanged*, сфера реагує на голос, відображаючи інтенсивність мовлення (рис. 3.9).

Рисунок 3.9 − Відображення анімованої сфери в Voice Assistant

3.4.6 Обробка помилок

Основний підхід полягає у фіксації помилок у логах, без виведення критичних повідомлень у графічний інтерфейс, що дозволяє уникнути переривання взаємодії з користувачем. Для централізованої обробки помилок використовується інтерфейс *IErrorHandler* та його реалізація *RecognitionErrorHandler*, яка записує всі помилки у лог за допомогою *Debug.WriteLine()*. У разі помилки викликається подія *OnProcessError*, яка

передається у *VoiceRecognitionService*. Компонент *GrpcProcessCoordinator* перехоплює помилки, що надходять від *StreamingGrpcProcessManager*, та класифікує повідомлення за наступними типами:

– *Log* − службові повідомлення;

– *Error* − помилки процесу;

– *Output* − коректні результати.

Лістинг 3.6 Класифікація повідомлень у *GrpcProcessCoordinator*:

```
if (text.StartsWith("[ERROR]") || text.Contains("помилк",
StringComparison.OrdinalIgnoreCase)) return MessageType.Error;
```

VoiceRecognitionService реагує на помилки процесу або завершення роботи передачі даних. У разі критичної помилки зупиняється процес розпізнавання, користувач отримує повідомлення у текстовому полі (через *OnRecognitionStopped*), система переходить у безпечний стан.

Наприклад, якщо сервер не запущено і ASR-сервіс не отримує даних, то виводиться повідомлення до логів, що виникла помилка відправки аудіо через неможливість підключення до каналу (рис. 3.10).

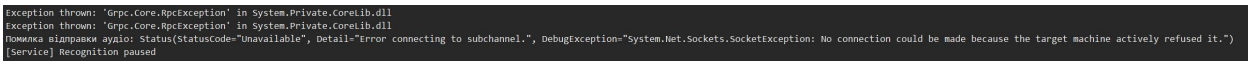


Рисунок 3.10 − Вивід про помилку відправки аудіо в логах

Якщо намір не визначено або впевненість низька, то відображається відповідне повідомлення, команда не виконується та система продовжує слухати (рис. 3.11). Це дозволяє уникнути помилкових дій.

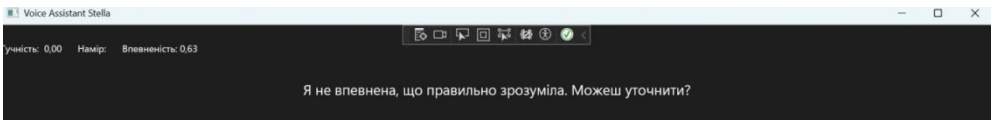


Рисунок 3.11 − Вивід про невпевненість в розпізнаваному тексті та формуванні відповіді

Механізм обробки помилок у клієнтській частині системи забезпечує стабільну роботу без преривання роботи застосунку навіть у разі виняткових ситуацій, безпечне завершення процесу розпізнавання при критичних збоях, коректну обробку порожніх або некоректних відповідей, та захист від помилок мережевої взаємодії, аудіозапису та парсингу.

3.5 Реалізація серверної частини

3.5.1 Мікросервісна архітектура

Серверна частина системи Voice Assistant реалізована на Python та побудована за принципами мікросервісної архітектури, що забезпечує модульність, масштабованість та незалежність компонентів. Кожен сервіс виконує окрему функцію − розпізнавання мовлення (ASR) або визначення наміру та формування відповіді (NLU).

Взаємодія між сервісами здійснюється через стандартизовані протоколи: gRPC для потокового передавання аудіо та HTTP/REST (FastAPI) для NLU-запитів.

Основним завданням серверної частини є обробка запитів користувача, розпізнавання мовлення та надання відповіді на запитання за допомогою алгоритмів TF-IDF та/або BERT.

Файлова структура серверної частини має наступний вигляд:

– speech_service/ − сервіс автоматичного розпізнавання мовлення (ASR);

– nlu_service/ − сервіс визначення наміру та генерації відповіді (NLU);

– proto/ − опис gRPC-контракту (speech.proto);

– *docker-compose.yml* − конфігурація для запуску обох сервісів у контейнерах.

ASR-мікросервіс реалізований на Python з використанням Vosk, gRPC на порту 50051, *ThreadPoolExecutor*. Після отримання тексту він надсилає

HTTP-запит до NLU-мікросервісу. Таким чином, ASR виконує лише свою відповідальність − розпізнавання мовлення.

NLU-мікросервіс реалізований на FastAPI та працює на порту 8000. Він має лише два основних маршрути запитів:

– GET / − повертає повідомлення про стан сервісу, чи він запущен;

– GET /predict − повертає результат роботи з BERT та TF-IDF.

Лістинг 3.7 Приклад відповіді NLU-сервісу, яку отримує ASR-сервіс:

{"intent": "define_inclusion",
"answer": "Інклюзія - це процес забезпечення рівних можливостей для всіх людей.",
"confidence": 0.92}

На діаграмі класів (рис. 3.12) показано структуру двох мікросервісів, де єдиним відношенням успадкування є клас *SpeechRecognizerServicer*, який реалізує інтерфейс, згенерований gRPC (*speech_pb2_grpc. SpeechRecognizerServicer*). Інші класи взаємодіють між собою через залежності, але не утворюють ієрархічних зв’язків.

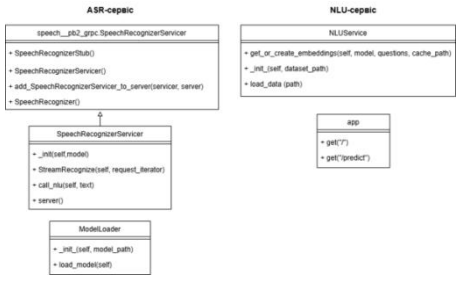


Рисунок 3.12 − Діаграма класів серверної частини Voice Assistant

Такий підхід забезпечує високу продуктивність, гнучкість та можливість подальшого масштабування. ASR-сервіс відповідає за потокове розпізнавання мовлення, тоді як NLU-сервіс виконує семантичний аналіз тексту та формує відповідь. Разом вони утворюють надійну та ефективну серверну інфраструктуру голосової помічниці.

3.5.2 Реалізація ASR-модуля

ASR-модуль відповідає за перетворення аудіопотоку, який надходить від клієнтського застосунку, у текст. Цей модуль реалізований як окремий мікросервіс *speech_service*, що працює на основі:

– Vosk − офлайн-моделі розпізнавання української мови;

– *KaldiRecognizer* − ядра розпізнавання;

– gRPC streaming − для прийому аудіоданих у режимі реального часу;

– HTTP-взаємодії з NLU-сервісом − для визначення наміру та формування відповіді.

ASR-мікросервіс використовує українську модель vosk-model-uk-v3-lgraph, яка підтримує 16 kHz PCM-аудіо, часткові та фінальні результати, розпізнавання слів (*SetWords = True*). Запуск сервера здійснюється у файлі *server.py*. Сервер працює на порту 50051 та обробляє кілька клієнтів одночасно завдяки пулу потоків (*ThreadPoolExecutor*).

Основна логіка розпізнавання реалізована у класі *SpeechRecognizerServicer* в файлі *service.py.* В методі *StreamRecognize()* створюється екземпляр *KaldiRecognizer*, де кожен аудіофрейм (*AudioChunk*) надходить у метод *AcceptWaveform()*. Якщо результат ще формується − повертається частковий текст. Для кожного фінального тексту викликається NLU-сервіс. Після цього клієнту повертається структура *RecognizeResponse* (*text, answer, intent, confidence*). Таким чином клієнт отримує миттєвий частковий текст (для відображення в UI) та фінальний текст (для NLU-аналізу).

3.5.3 Реалізація NLU-модуля

NLU-модуль відповідає за визначення наміру користувача та формування відповіді. Він реалізований як окремий мікросервіс на основі FastAPI, який працює на порту 8000 та обробляє HTTP-запити від ASR-модуля.

Основними завданнями NLU-модуля є попередня обробка тексту, класифікація наміру за допомогою TF-IDF або BERT, обчислення рівня впевненості, повернення відповідної відповіді з бази знань та кешування BERT-ембедингів для прискорення роботи.

Архітектура NLU-модуля передбачає чітке розділення етапів обробки тексту, що спрощує налагодження та розширення функціональності.

3.5.3.1 Попередня обробка тексту

Перед класифікацією текст користувача проходить базову попередню обробку. Це необхідно для підвищення точності визначення намірів та уникнення хибних спрацьовувань.

Першим кроком є приведення тексту до уніфікованого вигляду, що складається з видалення зайвих пробілів (*strip()*) та приведення до нижнього регістру (*lower()*). Це дозволяє уникнути ситуацій, коли одна й та сама фраза інтерпретується по-різному через різні регістри або пробіли.

Далі виконується перевірка, чи містить текст достатньо інформації для класифікації. Якщо фраза складається з одного слова або дуже коротка, вона не вважається інформативною. У системі визначено набір коротких відповідей, які не несуть семантичного навантаження. Це дозволяє уникнути ситуацій, коли користувач просто підтверджує дію або реагує на відповідь, а система намагається класифікувати це як запитання.

Лістинг 3.8 Перелік малозначущих слів, зазначених в NLU-сервісі:

LOW_VALUE_WORDS = {"ок", "гаразд", "ага", "так", "ні"}

Для навчання моделей TF-IDF та BERT використовується CSV-файл із трьома колонками: question, answer, intent. Функція *load_data()* виконує завантаження CSV-файлу, видалення записів із порожніми значеннями,

приведення всіх полів до рядкового типу та повернення списків запитань, відповідей та намірів.

Лістинг 3.9 Завантаження набору даних в методі *load_data()*:

```
df = pd.read_csv(path)
df = df.dropna(subset=['question', 'answer', 'intent'])
df['question'] = df['question'].astype(str)
df['answer'] = df['answer'].astype(str)
df['intent'] = df['intent'].astype(str)
return df['question'].tolist(), df['answer'].tolist(), df['intent'].tolist()
```

Попередня обробка тексту є критично важливою, оскільки зменшує кількість хибних класифікацій, запобігає обробці шумових або коротких фраз, забезпечує стабільність роботи моделей, покращує якість семантичного пошуку у BERT та дозволяє TF-IDF працювати з чистими токенами.

3.5.3.2 Реалізація TF-IDF

У системі алгоритм TF-IDF реалізований у вигляді окремого класу *TfidfModel*, який імпортується з пакету *ml_models*. TF- IDF працює як класичний статистичний підхід до аналізу тексту, що дозволяє визначити, наскільки важливими є окремі слова у запитанні користувача порівняно з усім текстом.

У конструкторі *NLUService.__init__()* дані завантажуються з CSV-файлу за допомогою функції *load_data()* та створюється екземпляр моделі *model_tfidf,* й зі списків *questions*, *answers*, *intents* відбувається навчання моделі TF-IDF. Під час навчання за допомогою функції *train()* модель аналізує всі запитання (*questions*) і формує список унікальних слів. Для кожного запитання рахується, як часто кожне слово зустрічається у цьому запитанні (частота термінів, *TF*).

Для кожного слова визначається, у скількох запитаннях воно зустрічається (зворотня частота документів, *IDF*): чим рідше слово − тим більша його вага. Кожне запитання перетворюється у вектор чисел, де кожне число − важливість певного слова. Та відбувається зв’язування векторів із намірами, модель зберігає відповідність: вектор − *intent*. Таким чином формується база для подальшої класифікації.

Коли користувач надсилає текст, у методі *predict()* виконується:

Крок 1. Попередня обробка тексту. Текст очищується, токенізується та перетворюється у TF- IDF вектор за тією ж схемою, що й під час навчання.

Крок 2. Обчислення схожості. Модель порівнює вектор користувацького тексту з усіма векторами запитань у наборі даних.

Крок 3. Вибір найближчого наміру. Модель знаходить запитання з найбільшою схожістю та повертає відповідний *intent*.

Крок 4. Впевненість. TF-IDF не обчислює реальну ймовірність, тому в системі використовується *confidence = 1,0*. Тобто модель впевнена у своєму виборі, але це не статистична впевненість, а технічне значення.

TF-IDF забезпечує високу швидкість обчислень і дозволяє ефективно працювати навіть на обмежених обчислювальних ресурсах, що є важливим для систем реального часу. Але він не враховує контекст, не розуміє синоніми, працює гірше на довгих або складних фразах.

3.5.3.3 Реалізація BERT

На відміну від TF-IDF, який працює зі статистичними частотами слів, BERT формує щільні контекстні векторні представлення, що дозволяє моделі розуміти значення фрази, а не лише окремих слів. У системі використовується Sentence-BERT − модифікація BERT, що оптимізована для обчислення семантичної подібності між реченнями.

У конструкторі NLUService створюється екземпляр моделі *model_bert*. Під час навчання за допомогою методу *train()* модель отримує список запитань (*questions*), кожне запитання перетворюється у вектор розмірністю 768 (для базової BERT-архітектури). Вектори зберігаються у пам'яті та використовуються для подальшого порівняння. Модель вчиться зіставляти текст користувача з найбільш схожим запитанням у наборі даних.

Оскільки обчислення ембедингів для всього набору даних є дорогим (особливо при великій кількості запитань), у системі реалізовано механізм кешування в методі *get_or_create_embeddings()*: якщо файл *dataset_embeddings.npy* існує − ембединги завантажуються з диска, якщо ні − модель кодує всі запитання, ембединги зберігаються у файл.

Кешування ембедингів дозволяє скоротити час запуску сервісу, уникнути повторного обчислення ембедингів, зменшити навантаження на CPU/GPU.

Класифікації намірів реалізована у методі *predict()* за схемою:

Крок 1. Кодування тексту користувача. Текст перетворюється у вектор, семантичне представлення фрази (*query_emb*) за допомогою методу *encode()*.

Крок 2. Обчислення косинусної подібності. Для кожного запитання у наборі даних обчислюється ступінь схожості з текстом користувача (*sims*) за допомогою *cosine_similarity()*.

Крок 3. Вибір найбільш схожого наміру. Обчислюється *best_idx* − індекс найбільш схожого запитання за допомогою *sims.argmax()*; *intent* − відповідний намір, визначається як найкращий *best_idx*; *confidence* − значення косинусної подібності (0-1).

Крок 4. Перевірка порогу впевненості (*CONFIDENCE_THRESHOLD*). Поріг встановлено на рівні *0,75*. Це дозволяє уникати помилкових відповідей, коли текст користувача не схожий на жодне запитання з набору даних.

Крок 5. Формування відповіді. Якщо намір визначено впевнено, система повертає відповідь із бази знань. База відповідей формується у вигляді словника.

BERT враховує контекст, тому: «Що таке інклюзія?», «Поясни інклюзію», «Що означає інклюзивність?» − будуть класифіковані як один і той самий намір.

BERT-модель забезпечує точне визначення намірів, семантичне розуміння тексту, стійкість до варіацій формулювань, можливість масштабування та розширення бази знань.

3.6 Тестування та оцінювання якості системи

Тестування системи Voice Assistant проводилося з метою оцінити якість роботи окремих модулів, а також перевірити коректність інтеграції між клієнтською та серверною частинами. Оцінювання виконувалося як на рівні окремих алгоритмів, так і на рівні всієї системи в цілому. Для тестування NLU-модуля було використано спеціальний скрипт *evaluate.py*, який порівнює TF-IDF та BERT на основі набору даних *dataset_civic_terms*.

Набір даних було розділено на: навчальну вибірку − 1174 приклади; тестову вибірку − 504 приклади.

Оцінювання проводилося за такими метриками:

– точність (*Accuracy*) − частка правильно класифікованих прикладів;
– F1-міра − збалансована метрика точності та повноти;
– час обробки одного запиту (*latency*);
– розподіл значень впевненості;
– матриця плутанини для аналізу помилок класифікації.

3.6.1 Тестування працездатності

На першому етапі було проведено тестування загальної працездатності всієї системи, що включає взаємодію клієнтської частини, ASR-модуля та NLU-модуля. Метою цього етапу було переконатися, що всі компоненти коректно запускаються, доступні за відповідними маршрутами та здатні обмінюватися даними в режимі реального часу.

Було виконано наступні перевірки:

– запуск ASR-сервера на порту 50051 (рис. 3.13);
– запуск NLU-сервера на порту 8000;
– перевірка доступності маршрутів / (рис. 3.14) та /predict (рис. 3.15);
– перевірка коректності gRPC-підключення з клієнта;
– тестування передачі аудіофреймів у режимі реального часу;
– перевірка обробки часткових та фінальних результатів (рис. 3.16).

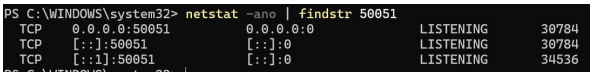


Рисунок 3.13 − Результат перевірки ASR-сервісу на порту 50051

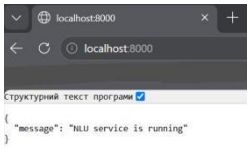


Рисунок 3.14 − Відповідь маршруту *localhost:8000/* від NLU-сервісу

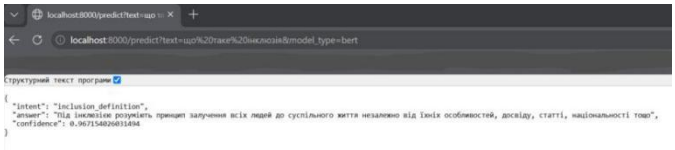


Рисунок 3.15 − Відповідь маршруту *localhost:8000/predict* із переданим текстом

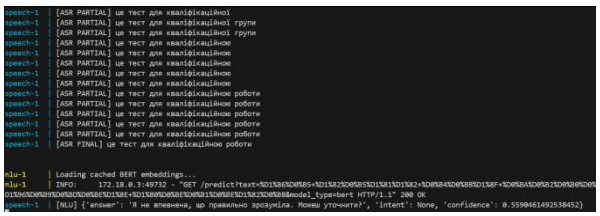


Рисунок 3.16 − Вивід часткових та фінальних результатів ASR-сервісу та відповідь NLU-модуля в консолі

За результатами тестування було підтверджено, що клієнтська частина успішно встановлює gRPC-з’єднання з ASR-модулем; аудіодані передаються стабільно та без помітних затримок; ASR-модуль коректно формує часткові та

фінальні результати розпізнавання; NLU-модуль правильно обробляє текст і повертає намір, відповідь та рівень впевненості; інтерфейс користувача оновлюється в реальному часі відповідно до отриманих подій.

Таким чином, працездатність системи стабільна, а всі основні компоненти успішно взаємодіють між собою у режимі реального часу.

3.6.2 Тестування ASR-модуля

Тестування ASR-модуля проводилося з метою оцінити якість розпізнавання української мови, стабільність роботи потокового розпізнавання та здатність системи коректно обробляти аудіодані в режимі реального часу.

Було протестовано: різні типи мікрофонів (настільний мікрофон Fifine та вбудований мікрофон ноутбука), різні рівні шуму (шепіт, напівшепіт та нормальна розмовна гучність), різні темпи мовлення (короткі та довгі фрази, швидке та повільне мовлення).

Отримані результати показали, що:

– часткові результати з’являються практично миттєво − із затримкою менше ніж 300 мс, що забезпечує плавне оновлення UI (рис. 3.17);

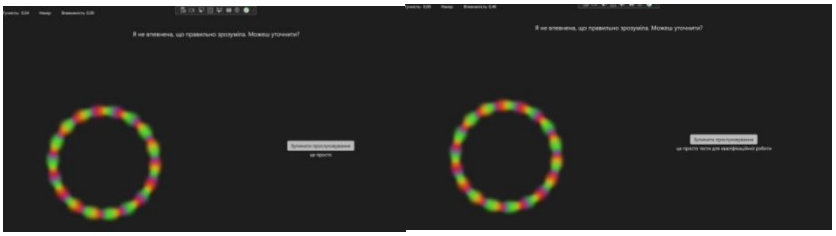
– розпізнавання шепоту та напівшепоту є нестабільним − модель потребує чіткішого та гучнішого мовлення;

– обидва протестовані мікрофони забезпечили достатню якість сигналу, хоча настільний мікрофон дав кращу чіткість;

– фінальний результат формується після короткої паузи, що відповідає стандартній поведінці Vosk;

– модель Vosk загалом добре розпізнає українську мову, але інколи допускає помилки у схожих словах, наприклад: «просто» − «пошта»; «розпізнаєш» − «спізнаєш» або «впізнаєш»;

– ASR-модуль стабільно працює при тривалій передачі даних (10+ хвилин) без витоків пам’яті чи зависань.



(а) (б)

Рисунок 3.17 − Результати виведення тексту в реальному часі:
(а) виведення часткового результату; (б) виведення фінального результату

Таким чином, проведене тестування підтвердило, що ASR-сервіс працює стабільно, забезпечує низьку затримку, коректно обробляє потокові аудіодані та може бути надійною основою для подальшої семантичної обробки тексту в NLU-модулі.

3.6.3 Порівняння TF-IDF та BERT

Для порівняння ефективності моделей TF-IDF та BERT було використано тестову вибірку обсягом 504 приклади, сформовану шляхом випадкового поділу набору даних *dataset_civic_terms* у пропорції 70/30. Таке порівняння дозволяє оцінити не лише точність моделей, а й їхню придатність для роботи в реальному часі та ефективність у контексті голосових систем.

Оцінювання проводилося за допомогою спеціального скрипту *evaluate.py*, який автоматизує:

– навчання моделей на тренувальній вибірці;

– класифікацію тестових прикладів;

– обчислення метрик якості (*Accuracy, F1, Confidence, latency*);

– побудову графіків порівняння щодо точності, часу обробки, впевненості та матриць плутанини.

У таблицях 3.7 та 3.8 узагальнено ключові відмінності між моделями TF-IDF і BERT та наведено результати їх експериментального оцінювання.

Таблиця 3.7 − Порівняння характеристик TF-IDF та BERT

Характеристика	**TF-IDF**	**BERT**
Тип моделі	Статистична	Нейромережна
Розуміння контексту	Ні	Так
Стійкість до перефразувань	Низька	Висока
Швидкість	Дуже висока	Нижча
Точність	Середня	Висока
Використання	Резервний режим	Основний режим

Таблиця 3.8 − Результати порівняння моделей TF-IDF та BERT

Модель	**Accuracy**	**F1-score**	**Latency (c)**
TF-IDF	0,936507	0,925288	0,454
BERT	0,998015	0,997354	1,656
BERT (з кешуванням)	1,0	1,0	1,561

3.6.3.1 Матриця плутанини

Для детального аналізу помилок класифікації було побудовано матриці плутанини для 10 найчастіших намірів у тестовій вибірці.

Матриця плутанини (рис. 3.18) дозволяє оцінити, які саме класи моделі плутають між собою, та наскільки чітко кожен алгоритм розмежовує семантично близькі запити.

В матриці плутанини TF-IDF присутні помилки між схожими намірами, модель інколи плутає терміни, що мають близьку лексику та є поодинокі перехресні класифікації.

Матриця плутанини BERT містить лише діагональні значення та не зробила жодної помилки, усі наміри класифіковано правильно. Це підтверджує, що BERT краще розрізняє семантично близькі фрази.

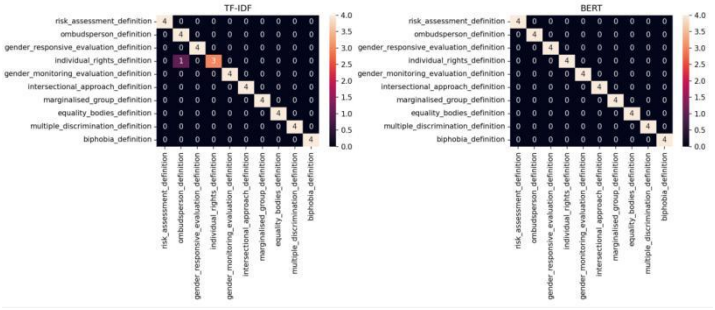


Рисунок 3.18 − Матриця плутанини TF-IDF та BERT

Таким чином, BERT є значно надійнішим інструментом для класифікації намірів у системі голосової помічниці, особливо в умовах реальних користувацьких запитів, де формулювання можуть бути різними.

3.6.3.2 Оцінка часу обробки

Окрім точності класифікації, важливим критерієм оцінювання моделей є час обробки запиту (*latency*) для якого було побудовано графік порівняння часу обробки моделей TF-IDF та BERT (рис. 3.19).

У контексті голосової помічниці цей показник має особливе значення, оскільки впливає на швидкість реакції системи та комфорт користувача.

У процесі тестування час обробки вимірювався як загальний час класифікації всіх тестових прикладів та середній час, необхідний моделі для обробки одного запиту.

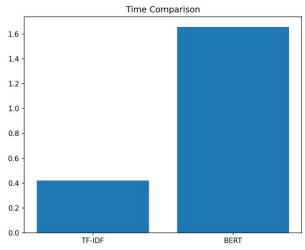


Рисунок 3.19 − Графік порівняння часу обробки моделей TF-IDF та BERT

TF-IDF працює значно швидше (~0,45 секунди), оскільки використовує просту векторизацію та обчислення ймовірностей на основі частот слів. BERT демонструє більшу затримку (~1,65 секунди), що пов’язано з обчисленням ембедингів та порівнянням їх із векторами всього набору даних.

Проте після застосування механізму кешування ембедингів для BERT, який дозволяє уникнути повторного обчислення векторів для всього набору даних, час обробки BERT зменшився до ~1,56 секунди (рис. 3.20).

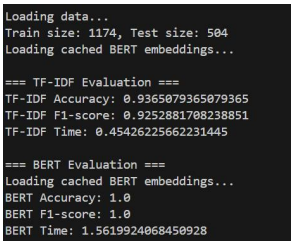


Рисунок 3.20 − Результати порівняння TF-IDF та BERT з кешуванням

3.6.3.3 Оцінка точності та впевненості

Для більш детального аналізу якості класифікації намірів було проведено порівняння моделей за точністю (рис. 3.21), та впевненістю (рис. 3.22).

Обчислення впевненості для TF-IDF виконувалось за максимальною ймовірністі серед усіх класів, а для BERT обчислювалось як косинусна подібність між ембедингом запиту та найближчим ембедингом у наборі даних.

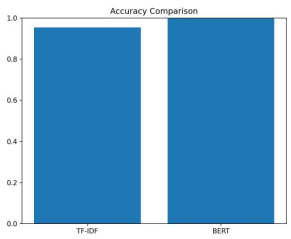


Рисунок 3.21 − Результати порівняння точності моделей TF-IDF та BERT

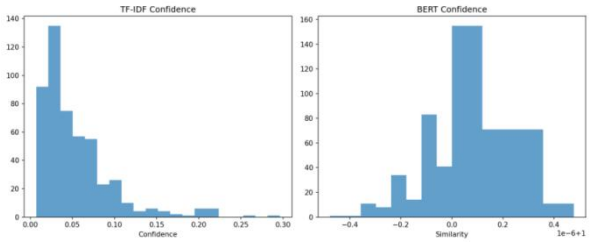


Рисунок 3.22 − Результати порівняння впевненості для TF-IDF та BERT

У результатах BERT значно перевершує TF-IDF за точністю, досягаючи 99,8-100%. TF-IDF має нижчу впевненість (0,0-0,05) так як працює лише з статистичними частотами слів. Графіки точності підтверджують, що TF-IDF менш надійний у задачах із семантично близькими формулюваннями. У реальних умовах використання BERT є оптимальним вибором, тоді як TF-IDF може застосовуватися як резервний або легковаговий режим.

Таким чином, TF-IDF може використовуватися як резервний або легковаговий режим, тоді як BERT залишається основним інструментом для високоточної семантичної обробки запитів.

ВИСНОВКИ

У кваліфікаційній роботі було виконано розроблення та експериментальне оцінювання програмної системи голосової помічниці (Voice Assistant), орієнтованої на сферу громадської діяльності. Метою роботи було створення моделі обробки запитів, здатного розпізнавати мовлення, визначати наміри користувача та надавати релевантні відповіді на основі сучасних методів машинного навчання для класифікації намірів.

У процесі виконання роботи були досягнуті такі результати:

– проведено аналіз сучасних підходів до побудови голосових помічників, включно з rule-based підходами, технологій розпізнавання мовлення (ASR), таких як: HMM-DNN, RNN-based моделі та трансформерні системи (Whisper, DeepSpeech), а також методів семантичної обробки тексту (NLU), включно з TF-IDF, Word Embeddings та BERT-подібними моделями;

– розглянуто можливості застосування TF-IDF та BERT у задачах класифікації намірів, визначено їх сильні та слабкі сторони: TF-IDF показав високу швидкість, але низьку контекстну чутливість, тоді як BERT продемонстрував значно вищу точність та здатність розрізняти семантично близькі запити;

– розглянуто існуючі цифрові рішення (Polis, Decidim, Дія, єВорог) та чат-боти (Resistbot, GovBot), що застосовуються у громадському секторі, зокрема у напрямках гендерної рівності, інклюзії та правозахисної діяльності;

– спроєктовано архітектуру програмної системи, що включає ASR-модуль на основі Vosk, NLU-модуль з реалізованими алгоритмами TF-IDF та BERT, клієнтський застосунок WinUI 3 та серверну частину на Python з FastAPI, інтегровані через gRPC, які забезпечують потокове розпізнавання української мови й запитів в реальному часі;

– підготовлено спеціалізований набір даних dataset_civic_terms із термінологіями, необхідними в громадській діяльності, для гендерної рівності та інклюзії, що став основою для навчання моделей;

– розроблено NLU-сервіс, який виконує класифікацію намірів користувача за допомогою моделей TF-IDF та BERT, а також формує відповідь на основі бази знань;

– забезпечено взаємодію між компонентами системи через gRPC, що дозволило реалізувати потокову передачу аудіо та швидкий обмін даними між сервісами;

– створено клієнтський інтерфейс з використанням WinUI 3 та XAML, який забезпечує зручну взаємодію користувача з голосовою помічницею, включно з візуалізацією голосової активності та відображенням відповідей;

– проведено експериментальне оцінювання ефективності моделей, включно з обчисленням Accuracy, F1-міри, часу обробки та аналізом матриць плутанини, де модель BERT значно перевершує TF-IDF за точністю та здатністю розрізняти семантично близькі запити;

– виконано порівняльний аналіз TF-IDF та BERT, на основі якого сформульовано рекомендації щодо вибору моделі: TF-IDF доцільно використовувати у легковагових системах або для швидкого прототипування, тоді як BERT є оптимальним вибором для реальних застосунків, де критично важлива точність та семантичне розуміння тексту.

Отримані результати підтверджують доцільність використання сучасних методів обробки природної мови у сфері громадської діяльності та відкривають можливості для подальшого розширення функціональності голосової помічниці. Результати дослідження та реалізації можуть бути використані як основа для подальших наукових і практичних розробок у галузі інтелектуальних голосових систем.

Результати роботи апробовано у вигляді тез доповідей під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ» [25].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mashtalir S. V., Nikolenko O. V. (2025). From Classification to Taxonomy: Automated Structuring of Vehicle Repair Names in Multilingual Corpora. Herald of Advanced Information Technology 2025, 8 (2), 151-163.

2. Suprun A., Tvoroshenko I., Gorokhovatskyi V., and Yakovleva O. (2025) Development and research of a method for the combined use of large language models for text generation, *International Journal of Academic and Applied Research*, 9(10), pp. 249-263.

3. Жуков А. І. (2022). Підсистема для оптимізації взаємодії між державними органами та людьми з обмеженими можливостями, Автоматизація та Приладобудування, 2, С. 167-171.

4. Bohdan N., Tvoroshenko I., Gorokhovatskyi V., and Kobylin O. (2025) Development of a hybrid method to enhance context memory for a chatbot application based on large language models, *International Journal of Academic Information Systems Research*, 9(10), pp. 7-18.

5. Творошенко, І. С., & Табашник, В. А. (2018). Розробка просторової моделі геоінформаційної підтримки людей з обмеженими можливостями, що пересуваються на інвалідних колясках, у місті Харків. *Збірник наукових праць Харківського національного університету Повітряних Сил*, (1), 122-128.

6. Творошенко, І. С. (2018). Особливості застосування сучасних принципів штучного інтелекту до розробки ефективних механізмів моделювання складних систем. *Science and Technology of the Present Time: Priority Development Directions of Ukraine and Poland*, 118-121.

7. Polis Educational Solutions. URL: https://www.polis.ai/ (дата звернення 23.04.2026).

8. Resistbot Action Fund. Resistbot. URL: https://resist.bot/ (дата звернення 23.04.2026).

9. Kovtunenko, A. R., & Mashtalir, S. V. (2026). Metrical loss functions for image segmentation based on convolutional neural networks. Applied Aspects of Information Technology, 9(2), 172-184.

10. Pomazan V., Tvoroshenko I., and Gorokhovatskyi V. (2023) Development of an application for recognizing emotions using convolutional neural networks, *International Journal of Academic Information Systems Research*, 7(7), pp. 25-36.

11. Decidim Free Software Association. Decidim. URL: https://decidim.org/ (дата звернення 23.04.2026).

12. Міністерство цифрової трансформації України. Дія. URL: https://diia.gov.ua/ (дата звернення 23.04.2026).

13. Державна служба спеціального зв'язку та захисту інформації України. єВорог. URL: https://evorog.gov.ua/ (дата звернення 23.04.2026).

14. Com Bot AI Limited. GovBot. URL: https://gov.bot/ (дата звернення 23.04.2026).

15. Jurafsky D., Martin J. H. (2023). Speech and language processing, An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 3, pp. 286. Pearson.

16. Аксак, Н. Г. (2015). Системи штучного інтелекту для студентів усіх форм навчання напряму: навч. посібник. Харків: ХНУРЕ.

17. Кавецький М. С., Руженцев В. І. (2024). Виявлення веб-атак по HTTP запитам з використання технік NLP. Радіотехніка, (218), С. 64-75.

18. Згуровський М.З. (2025). Системи і методи штучного інтелекту: підручник. Київ: Академперіодика.

19. Скрипник, І. А., & Безверхий, А. І. (2025). Векторизація формальних граматик для їх кластеризації засобами ML.NET. Праці таврійського державного агротехнологічного університету, 25(3), 94-99.

20. Normatova, T. V., Mashtalir, S. V. (2025). Road Traffic Accident Classification Using a Sparse Videotransformer and Adaptive Fragmentation. Herald of Advanced Information Technology 2025, 8(4), 464-475.

21. Tvoroshenko I., and Tkachenko D. (2020) Mechanisms of image classification based on descriptors of local features, *Abstracts of IV International Scientific and Practical Conference «Integration of scientific bases into practice» (October 12-16, 2020). Stockholm, Sweden*, pp. 443-448.

22. Mashtalir S.V., Nikolenko O. V. (2024). Hierarchical classification of ukrainian technical texts using tree-based models: applications in the automotive industry, Advances in information-control systems and technologies, 288 (2), pp. 288-312.

23. Mashtalir, S. V., Nikolenko, O. V. (2024). Optimizing hierarchical classifiers with parameter tuning and confidence scoring, Herald of Advanced Information Technology 2024, 7(3), pp. 231-242.

24. Гороховатський В., Творошенко І. (2026) Продуктивні моделі аналізу даних у методах розпізнавання зображень: монографія. Харків: ХНУРЕ. 153 с.

25. Єфімова В. В. (2026) Машинне навчання у спеціалізованих голосових системах. *Радіоелектроніка і молодь у XXI столітті (МРФ-2026). Тези тридцятого міжнародного молодіжного форуму (22-24 квітня 2026 року). Харків: ХНУРЕ, С. 53-55.*

26. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2024) Improving the effectiveness of image classification structural methods by compressing the description according to the information content criterion, *Computers, Materials & Continua*, vol. 80, no. 2, pp. 3085-3106.

27. Tvoroshenko, I., & Temchur, K. (2021). Features of software application development for food recognition using deep machine learning methods.

28. Gorokhovatskyi V., Chmutov Y., Tvoroshenko I., and Kobylin O. (2025) Reducing computational costs by compressing the structural description in image classification methods, *Advanced Information Systems*, vol. 9, no. 1, pp. 5–12.

29. M. Ayaz Ahmad, Irina Tvoroshenko, Jalal Hasan Baker, and Vyacheslav Lyashenko (2019) Modeling the Structure of Intellectual Means of Decision-Making Using a System-Oriented NFO Approach, *International Journal of Emerging Trends in Engineering Research*, 7(11), pp. 460-465.

30. М. Бабак, О. Давліканова, М. Дмитрієва, М. Козир, Л. Компанцева, К. Левченко, М. Скорик, О. Суслова. (2021). Глосарій і тезаурус Європейського інституту з ґендерної рівності. Вістка.

31. Асоціація жінок-юристок України “ЮрФем”. Глосарій з гендерно чутливого відновлення. URL: https://genderplatform.org.ua/wp content/uploads/2025/11/glosarij-d.pdf (дата звернення 19.05.2026).