Кафедра *програмного забезпечення*

ПОЯСНЮВАЛЬНА ЗАПИСКА

до магістерської кваліфікаційної роботи на тему:

Побудова специфікації діаграми класів засобами штучного інтелекту для розпізнавання англійської мови (Constructing a class diagram specification using artificial intelligence for English language recognition).

Студент групи ПЗМ-21, Качан М. В.

Керівник проекту _____ (*Левус Є. В.*)
(підпис)

Консультанти _____ (_____)
(підпис)

_____ (_____)
(підпис)

_____ (_____)

_____ (_____)

_____

Завідувач кафедри *Федасюк Д. В.*
(підпис)

“ ____ ” _____ 20 *25* р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення
Спеціальність 121 "Інженерія програмного забезпечення"

"ЗАТВЕРДЖУЮ"
Завідувач
кафедри Федасюк Д. В.

« ____ » _____ 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу (проект) студента групи ПЗМ-21 ОКР магістр
Качана Михайла Вікторовича
(прізвище, ім'я, по-батькові)

1. Тема роботи (проекту) Побудова специфікації діаграми класів засобами штучного інтелекту для розпізнавання англійської мови
(у разі виконання комплексної роботи в дужках вказується "комплексна робота(проект)")

затверджена наказом по університету від « 04 » квітня 2025 р. № 1264-4-06

2. Термін подання студентом закінченої роботи (проекту) 16 травня 2025 р

3. Вхідні дані для роботи (проекту) документація побудови UML діаграми класів, документація Whisper, документація ffmpeg

4. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити) аналіз наявних рішень і постановка завдання, вибір алгоритмічного забезпечення, специфікація вимог, реалізація, тестування, дослідження системи та оцінка її ефективності, висновки та перспективи розвитку

5. Перелік графічного матеріалу діаграма класів, семантичні мережі, графічне представлення архітектури Whisper

6. Перелік програмних продуктів, які належить використати в процесі розроблення роботи (проекту) модель перетворення голосу в текст Whisper, Visual Studio, ffmpeg, PyTorch, Python, Chocolatey, C#, github

7. Консультування роботи (проекту), із зазначенням розділів роботи

Розділ	Консультант	Завдання видав		Завдання прийняв	
		підпис	дата	підпис	дата

8. Дата, коли видано завдання 04.04.2025

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи (проекту)	Термін виконання етапів роботи (проекту)	Примітка
1	Ознайомлення з предметною областю	04.04.2025 – 06.04.2025	ВИКОНАНО
2	Аналіз наявних рішень	07.04.2025 - 15.04.2025	ВИКОНАНО
3	Проектування системи	16.04.2025 - 23.04.2025	ВИКОНАНО
4	Розробка системи	24.04.2025 - 08.05.2025	ВИКОНАНО
5	Тестування системи	09.05.2025 - 13.05.2025	ВИКОНАНО
6	Оформлення пояснювальної записки	14.05.2025 - 18.05.2025	ВИКОНАНО

Студент _____
(підпис)

Керівник _____
(підпис)

Видавництво Львівської політехніки. Наклад 5000. Зам. 180288. 2018

АНОТАЦІЯ

У магістерській кваліфікаційній роботі досліджено і розроблено систему автоматизації побудови UML-діаграм класів із застосуванням штучного інтелекту для розпізнавання природної мови. Проведений аналіз сучасних підходів та інструментів створення діаграм класів підтвердив актуальність використання голосових технологій для автоматизації та оптимізації проектування програмного забезпечення.

Розроблена система використовує модель Whisper від OpenAI, яка ефективно перетворює голосові команди англійською мовою у текстові специфікації UML-діаграм класів. Для програмної реалізації обрано мову програмування Python із застосуванням фреймворку PyTorch, що забезпечило високу продуктивність та точність роботи.

Експериментально досліджено вплив якості запису голосу, рівня фонового шуму та параметрів використовуваної моделі Whisper на точність та швидкість розпізнавання. Дослідження показали, що модель Whisper Medium забезпечує найвищу точність, однак вимагає значних обчислювальних ресурсів. Модель Whisper Small визначена оптимальною за співвідношенням швидкості та точності.

Статистичний аналіз результатів експериментів підтвердив достовірність отриманих даних та їхню практичну значущість. Розроблена програмна система демонструє значний потенціал для застосування у реальних умовах проектування ПЗ, суттєво скорочуючи витрати часу і зменшуючи кількість помилок, пов'язаних із людським фактором.

Ключові слова: UML-діаграма класів, штучний інтелект, розпізнавання голосу, Whisper, PyTorch, Python, автоматизація проектування.

ABSTRACT

The master's thesis investigates and develops a system for automating the construction of UML class diagrams using artificial intelligence for natural language recognition. An analysis of modern approaches and tools for creating class diagrams confirmed the relevance of using voice technologies to automate and optimize software design processes.

The developed system employs OpenAI's Whisper model, effectively transforming voice commands in English into textual specifications of UML class diagrams. Python was chosen as the programming language along with the PyTorch framework, ensuring high performance and accuracy.

Experimental research examined the impact of voice recording quality, background noise levels, and parameters of the Whisper model on recognition accuracy and speed. The studies showed that the Whisper Medium model provides the highest accuracy but requires significant computational resources. The Whisper Small model was identified as optimal in terms of the balance between speed and accuracy.

Statistical analysis of the experimental results confirmed the reliability and practical significance of the obtained data. The developed software system demonstrates substantial potential for real-world software design applications, significantly reducing time costs and human-related errors.

Keywords: UML class diagram, artificial intelligence, voice recognition, Whisper, PyTorch, Python, design automation.

ЗМІСТ

ВСТУП

Поточне століття відрзіняється від інших особливою швидкістю розвитку комп'ютерних технологій. Те, що колись було неможливо уявити, сьогодні вже стало реальністю. Наприклад, машинне навчання дозволило комп'ютеру "розуміти" мову людину, що знаменувало нову еру в людинно-машинній взаємодії. Тепер людині не обов'язково знати мову програмування, щоб "спілкуватись" з комп'ютером. Chat GPT, до прикладу, розуміє звичайну мову і є дуже потужним засобом пошуку і обробки інформації.

В умовах, коли технології розвинулись настільки добре, постає задача спрощення рутинної роботи людини, щоб дати їй можливість фокусуватись на її головній задачі – творчості.

Для створення ПЗ необхідним етапом є проектування майбутньої системи. Правильно розроблена архітектура дозволяє відносно легко вносити зміни у вимоги проекту, а також полегшує процес супроводу розробленої системи. Однією з найкращих і найкорисніших моделей є діаграма класів, що розбиває систему на взаємодіючі між собою класи. Проте процес побудови діаграми класів вимагає ручного введення кожного класу і їхніх компонент, що, в цілому, є довгою і нудною задачею. Як один з методів пришвидшення роботи архітектора можна надати йому можливість диктувати діаграму машині, при цьому вона сама буде розміщувати елементи. Так архітектор максимально зосереджується на творчій частині – процесі побудови діаграми і її компонент – уникаючи потреби вводити кожен атрибут, метод і клас вручну, самому будувати зв'язки, що само по собі не є інтелектуальною задачею. Таким чином, архітектор природно продумуватиме діаграму і надиктовуватиме думки машині, яка, в свою чергу, реалізовуватиме ці думки у вигляді конкретних компонент. Це дозволяє зменшити час, необхідний на створення діаграми, що призведе до економії грошей і менших ризиків невчасного здавання готового ПЗ.

Проаналізувавши джерела, стало зрозуміло, що подібних проєктів на даний момент не існує. Проте зменшення рутинної частини роботи архітектора ПЗ є важливою задачею. Архітектор – це звичайна людина, а отже, може допускати

невеликі помилки, які згодом ведуть до витраченого часу програмістів, щоб зрозуміти, що саме мав на увазі архітектор, а в гіршому випадку помилки лишаються як спадок, і ті люди, яким потім доведеться розширювати або змінювати систему, витрачатимуть час на розуміння помилкових частин діаграми. Комп'ютер, за умови доброго вибору архітектури моделі і її параметрів, матиме менший шанс помилки. Таким чином, це не тільки пришвидшує процес розробки діаграми класів, але і знижує ймовірність внесення в неї помилки.

РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ПОБУДОВИ СПЕЦИФІКАЦІЇ ДІАГРАМИ КЛАСІВ ЗАСОБАМИ ШІ ДЛЯ РОЗПІЗНАВАННЯ ПРИРОДНОЇ МОВИ

1.1. Актуальність, проблематика і практичне значення кваліфікаційної роботи

Розробка ПЗ відбувається за певним життєвим циклом (ЖЦ) і методологією. ЖЦ – це набір етапів, через які проходить проект від початку до впровадження і супроводу готового програмного рішення. Методологія – це предмет, який вказує, які методи використовуються при розробці ПЗ і за яким ЖЦ відбувається розробка. Кожна методологія передбачає власний порядок ЖЦ ПЗ, проте константою лишається одне – проектування завжди відбувається до процесу розробки.

Проектування є важливим етапом, оскільки правильно розроблена архітектура дозволяє відносно легко вносити зміни у вимоги проекту, а також полегшує процес супроводу розробленої системи. Одним з найпоширеніших методів проектування є побудова UML діаграм. Серед них найпопулярнішою є діаграма класів, яка розбиває систему на взаємодіючі між собою класи. Варто зазначити, що UML діаграми загалом, і діаграма класів зокрема, використовуються також для документації системи – з її допомогою новій людині легше розібратись у тому, як працює система.

На даному етапі, зручність конструювання діаграми класів – це питання, яке потребує особливої уваги, так як користувачі орієнтуються не лише на функціонал, але і на зручний інтерфейс користувача (UX). Наявні програмні рішення або надають дивний і незручний для користувача дизайн, або в принципі є непрофесійними системами по прикладу програми для малювання draw.io.

Іншим важливим чинником для покращення і пришвидшення процесу проектування є голосовий ввід елементів діаграми. Це дозволяє покращити досвід користувача (UX), дозволяючи йому зосередитись на самому процесі

продумування діаграми, витрачаючи мінімальну кількість зусиль на ввід самих елементів у систему.

1.2. Огляд предметної області, її специфіки та проблеми

Предметною областю є побудова специфікації UML діаграм класів з використанням штучного інтелекту для голосового вводу.

Процес розробки ПЗ включає різноманітні етапи: від проектування до впровадження та супроводу. Критичним аспектом є проектування, яке передбачає розробку архітектури системи. Важливим інструментом на цьому етапі є UML діаграми, що дозволяють візуалізувати структуру та взаємозв'язки елементів системи. Побудова ефективних UML діаграм є ключовою для успішної реалізації проекту.

Однією з основних проблем у сфері проектування ПЗ є складність та витрати часу на побудову деталізованих UML діаграм. Традиційно цей процес здійснюється вручну, що може бути часомістким і схильним до помилок. Необхідність постійних змін та уточнень у проектах ставить під вимогу більш гнучкі та ефективні інструменти для проектування.

Штучний інтелект пропонує нові можливості для оптимізації процесу проектування. Використання ШІ для розпізнавання голосу та автоматичного створення UML діаграм може значно спростити процес проектування, зробити його швидшим і точнішим. Це також дозволяє розробникам зосередитися на більш творчих аспектах проектування, замість рутинної роботи.

Однак інтеграція ШІ в процес проектування ПЗ стикається з кількома викликами. По-перше, це необхідність забезпечення точності розпізнавання голосових команд та їх адекватної інтерпретації в контексті UML діаграм. По-друге, існує потреба в розробці інтуїтивно зрозумілого інтерфейсу, який би враховував специфіку роботи розробників.

1.2.1. Перспективи розвитку:

Розвиток технологій ШІ і їх застосування в проектуванні ПЗ відкриває нові перспективи для поліпшення процесів розробки. Інноваційні підходи, такі як

використання ШІ для голосового вводу в проектуванні, можуть стати важливим кроком на шляху до створення більш ефективних та адаптивних інструментів для розробників.

Загалом, цей проект сприяє розвитку сфери розробки програмного забезпечення, пропонуючи рішення для підвищення ефективності та якості проектування. Впровадження ШІ у цей процес відкриває шлях до більш інноваційного та ефективного майбутнього в галузі розробки ПЗ.

1.3. Огляд та аналіз існуючих підходів до вирішення проблеми та визначення їхніх недоліків

Для початку варто зрозуміти, як загалом працює ШІ для розпізнавання мови. Для цього необхідно розібратись в термінології мови і як з цим працює ШІ[1]:

- Фонологія – наука про звук. Звук передає семантичне значення людської мови і посідає провідне місце в даній роботі.
- Морфологія – описує складові слів. Це є корінь, суфікс, префікс, закінчення.
- Лексикологія – інтерпретація значення слів. На першому етапі аналізу мови відбувається видобуток первинної інформації за допомогою таких технік:
    - Видалення стоп-слів – стоп-слова – це слова, які не вносять смислового навантаження (прийменники, "і", "або" і т. д.). Вони збільшують час обробки інформації, при тому їхня частота є високою.
    - Stemming – техніка, коли зі слів видаляються суфікси для отримання кореневої форми. Наприклад: consulting -> consult, using -> us. Іншими словами, не відбувається перевірки, чи таке слово існує. Лише видаляється суфікс.
    - Lemmatization – техніка, при якій замість видалення суфіксів використовується словник для отримання початкової форми

слова. Таким чином, беручи слово using як приклад, stemming видає us, a lemmatization – use.

- Синтаксис – після аналізу окремих слів, слова групуються у фрази, а фрази – у речення. На цьому рівні аналізується граматична структура речення, а також залежності між словами, так як групи слів можуть розкривати набагато більший зміст, ніж кожне слово окремо. На цьому рівні аналізується порядок слів, стоп-слова, морфологія. На цьому рівні неможливо використати lemmatization і stemming, так як зміна слів до їхньої базової форми міняє граматику речення.
- Семантика – на цьому рівні треба зрозуміти смисл речення. Для цього аналізується логічна структура для пошуку зв'язаних слів. Наприклад, ШІ розумієш, що мова іде про фільми, навіть якщо речення не має слова "Фільм", але має "Актор", "Діалог", "Сценарій". Також на цьому рівні відбувається пошук того значення слова, яке підходить найбільше. Також на цьому рівні для слів використовуються словники для правильної інтерпретаціїречення. Таким чином можна знайти сенс навіть незнайомих слів, які не внесені в словник. Напирклад, речення "Krishna is good and noble", слово "Krishna" позначає або людину, або лорда (тому що використовується слово"noble"). Тому для правильного розуміння речення необхідно аналізувати ціле речення.
- Дискурс – синтаксичний і семантичний рівні працюють в межах речення. Рівень дискурсу вже покриває набір речень. Він знаходить зв'язок між ними. Напирклад: "Рам був на вершині рейтингу. Він був дуже розумним." Ці два речення формують дискурс і ШІ має розуміти, що слово "Він" відноситься до "Рам", яке зустрічається раніше в тексті. Якщо не провести цей зв'язок, то не зрозуміло, хто був розумним і чому Рам був на вершині рейтингу.
- Прагматичність – цей рівень бере до уваги контекст. Одне і те ж речення може мати різне значення в залежності від контексту. Наприклад, речення "Чи ти знаєш котра година?" може означати

"Питання про поточний час" або "Занепокоєння про людину, яка просиділа за комп'ютером до п'ятої ранку". Тому семантичний аналіз – це зрозуміти, що означає речення. А прагматичний – підставити контекст для правильної інтепретації змісту. Це допомагає зрозуміти правильний смисл за допомогою контексту.

У [2] детально розглядається використання технік обробки природної мови для автоматизації створення діаграм системної інженерії, зокрема діаграм мови моделювання систем (SysML). Основна увага приділяється генерації діаграм SysML із неструктурованих текстів природної мови, з акцентом на структурні та вимогові діаграми. Представлено підхід, що включає кілька етапів: від вибору текстів, екстракції ключових іменників та відносин, до створення та візуалізації SysML-діаграм. Описано також алгоритми та інструменти, використані для реалізації цього процесу.

У останні роки розроблено багато методів і інструментів для генерації діаграм класів мови моделювання (UML) із вимог програмного забезпечення, викладених природною мовою (NL) [3]. Ці методи та інструменти займаються перетворенням текстових вимог NL на діаграми UML. Процес перетворення включає аналіз вимог NL і вилучення відповідної інформації з тексту для генерації моделей класів UML. Метою цієї роботи є огляд існуючих робіт з перетворення текстових вимог на моделі класів UML, щоб вказати їхні сильні та слабкі сторони. Робота надає всебічний опис і оцінку існуючих підходів та інструментів. Аналізуються та досліджуються ступінь автоматизації, ефективність, повнота, а також використані техніки. Дослідження показало необхідність автоматизації процесу, а також поєднання штучного інтелекту з інженерними вимогами і використання технік обробки природної мови (NLP) для вилучення діаграм класів із вимог NL.

У UML моделі створюються графічно за допомогою діаграм. Проте маніпулювати діаграмами за допомогою сучасних CASE-інструментів не завжди легко. Типово функціональність та інформація приховані за складними ієрархіями меню або діалогів, що зменшує зручність інструментів. Розпізнавання мовлення

може бути використане для підвищення зручності існуючих інструментів як доповнюючий користувацький інтерфейс, який дозволяє одночасно використовувати інші інтерфейси. У роботі [4] представлено підхід до розробки мовних інтерфейсів для інструментів UML. Також показується, що UML є сприятливою ціллю для розпізнавання мовлення і що розпізнавання мовлення є достатньо розвиненим для використання з метою покращення використання інструментів UML.

Завдяки останнім досягненням у сфері глибокого навчання, продуктивність додатків NLP значно покращилася. У роботі [5] представлено огляд застосування методів глибокого навчання в NLP з акцентом на різних завданнях, де глибоке навчання має найбільший вплив. Також досліджуються основні ресурси в дослідженнях NLP, включаючи програмне забезпечення, апаратне забезпечення та популярні корпуси. Нарешті, висвітлюються основні обмеження глибокого навчання в NLP та сучасні напрямки досліджень.

У дослідженні [6] розглядаються можливості систем обробки мовлення, які навчені лише на основі великої кількості транскриптів аудіо з Інтернету. При масштабуванні до 680 000 годин багатомовного і багатозадачного навчання, отримані моделі добре узагальнюються на стандартних бенчмарках і часто конкурують з попередніми результатами, повністю отриманими за допомогою навчання, без необхідності специфічного до даних тонкого налаштування. Порівняно з людьми, моделі наближаються до їх точності та стійкості.

Некероване розпізнавання мовлення демонструє великий потенціал у тому. щоб зробити системи автоматичного розпізнавання мовлення (ASR) доступними для кожної мови. Однак існуючі методи все ще значною мірою залежать від ручної попередньої обробки. Слідуючи тренду перетворення керованого розпізнавання мовлення в кінцевий продукт, робота [7] представляє метод, який відмовляється від усієї аудіо-попередньої обробки і покращує точність за допомогою кращої архітектури. Крім того, вводиться допоміжна самокерована ціль, яка пов'язує передбачення моделі з входом. Експерименти показують, що

цей метод покращує результати некерованого розпізнавання у різних мовах, будучи концептуально простішим.

Мовлення є однією з найбільш фундаментальних і необхідних форм людського спілкування. Люди та комп'ютери взаємодіють через так званий інтерфейс людина-комп'ютер. Мовлення може використовуватися для спілкування з комп'ютером. Розпізнавання мовлення використовується не тільки в мобільних пристроях, але й у вбудованих системах, сучасних настільних та портативних комп'ютерах, операційних системах та браузерах. Це корисно для дітей, літніх людей, а також людей, які не бачать або мають обмежені можливості зору. Це особливо важливо для фізично обмежених осіб, які покладаються тільки на цей спосіб взаємодії з комп'ютерними системами. Дослідження у галузі розпізнавання голосу стають більш винахідливими. Дослідники намагаються розширити способи, якими комп'ютери можуть використовувати людське мовлення. Оглядова стаття [8] має на меті класифікувати методи перетворення людського мовлення на формат, зрозумілий для комп'ютерів. Аналізуються виклики найпопулярніших на сьогодні систем розпізнавання мовлення, і представляються рішення. Ця оглядова робота призначена для надання підсумків дослідникам, які працюють у сфері розпізнавання мовлення. Як критичні компоненти системи розпізнавання мовлення, вирішальними є як виділення ознак, так і класифікація. Фокус цього дослідження полягає у представленні огляду літератури щодо стратегій виділення ознак і класифікації для систем розпізнавання мовлення.

Стаття [9] представляє порівняльне дослідження систем розпізнавання емоцій мовлення (SER). Подано теоретичне визначення, категоризацію афективного стану та модальності вираження емоцій. Для досягнення цього дослідження розроблено систему SER на основі різних класифікаторів і методів вилучення ознак. Використані коефіцієнти Mel-частотного кепстру (MFCC) та модуляційні спектральні характеристики для навчання різних класифікаторів. Застосовано вибір ознак для пошуку найрелевантнішого набору ознак. Використані різні парадигми машинного навчання для класифікації емоцій.

Рекурентні нейронні мережі (RNN) порівнювали з методами багатовимірної лінійної регресії (MLR) та машин опорних векторів (SVM), які широко застосовуються у розпізнаванні емоцій у мовленні. Використані Берлінська та Іспанська бази даних. Для Берлінської бази всі класифікатори досягли точності 83% з нормалізацією мовця та вибором ознак. Для Іспанської бази найкращу точність (94%) показав класифікатор RNN без нормалізації мовця та з вибором ознак.

Стаття [10] містить класифікацію алгоритмів, за допомогою яких вимовлене слово може бути перетворене на форму, зрозумілу комп'ютеру. У статті перелічуються та аналізуються виклики в області розпізнавання мовлення для найпопулярніших сучасних технік розпізнавання. Аналіз завершується коротким описом деяких застосувань розпізнавання мовлення.

З розвитком технології неперервного розпізнавання мовлення користувачі висувають вищі вимоги до точності розпізнавання мовлення. Розпізнавання мовлення з обмеженими ресурсами, як типова технологія розпізнавання мовлення в умовах обмежень, стало сучасним дослідницьким напрямком через низький рівень розпізнавання та велику практичну цінність. Виходячи з основ розпізнавання мовлення з обмеженими ресурсами, робота [11] оглядає стан дослідження вилучення ознак та акустичних моделей, проводить дослідження розширення ресурсів. Особливо розглядаються технічні виклики, які стоять перед цією технологією, пропонуються рішення та прогнозуються майбутні напрямки досліджень.

Робота [12] описує покращені дослідження систем розпізнавання емоцій у мовленні (SER). Теоретично вводяться визначення, класифікація стану емоцій та їх вираження. Розроблено систему SER на основі класифікатора CNN і вилучення ознак MFCC. Застосовані коефіцієнти мел-частотного кепстру (MFCC) для навчання різних класифікаторів. Використані бази даних SAVEE, RAVDESS, TESS, CREMA-D для експериментів. Дослідження показує, що всі чотири бази даних використовують класифікатор CNN. Загальна точність розпізнавання емоцій з 1D-CNN - 43%, точність розпізнавання статі - 81%, нейтральна до статі

точність розпізнавання емоцій - 48%. З 2D-CNN загальна точність - 67,58%, точність розпізнавання статі - 98%, точність розпізнавання емоцій без врахування статі - 65%.

У роботі [13] пропонується використання традиційної технології ASR для розпізнавання мовлення в рамках глибокої моделі мовлення. Модель ASR повинна виявляти мовців індійської англійської та перетворювати їх мовлення на текст. Акцент визначається основними акустичними характеристиками та вимовою, що є значним джерелом варіацій у мовленні. Для покращення точності розпізнавання мовлення може бути використаний акцентозалежний словник або модель. Експериментальний підхід полягає у передачі навчань попередньої моделі на різні акценти, дозволяючи моделі самостійно навчатися різним акцентам.

Протягом останніх десятиліть проведено значну кількість досліджень використання машинного навчання для обробки мовлення, особливо для розпізнавання мовлення. Однак останніми роками увага дослідників зосередилася на використанні глибокого навчання у застосуваннях, пов'язаних з мовленням. Ця нова галузь машинного навчання принесла значно кращі результати у різноманітних застосуваннях, включаючи мовлення, та стала привабливою для досліджень. Робота [14] надає детальний аналіз різних досліджень з 2006 року, коли глибоке навчання з'явилося як нова галузь машинного навчання, для застосувань мовлення.

У дослідженні [15] розглядається розпізнавання емоцій у мовленні, що привертає значну увагу останніми роками. Пропонується модель на основі глибокого навчання для розпізнавання емоцій в мовленні з використанням синтетичного кістково-провідного мовлення. Модель перетворює повітряно-провідне мовлення в кістково-провідне за допомогою фільтра IIR. Використовуються техніки збільшення даних та модифіковані параметри CNN для підвищення точності моделі. Результати моделювання показують, що пропонована модель перевершує існуючі за точністю розпізнавання кістково-провідного мовлення.

Головні результати роботи [16] полягають у розробці методу автоматичного створення UML-діаграм класів із текстових специфікацій, написаних природною мовою. Автори використали поєднання NLP-технік і набору евристичних правил для виявлення об’єктно-орієнтованих концептів, таких як класи, атрибути, методи та зв’язки. Метод показав прийнятну точність при оцінюванні на прикладі відомого кейсу, і результати були зіставлені з іншими інструментами. Ці результати можуть бути використані для створення модуля автоматичного перетворення голосових або текстових описів у UML-структури, що дозволить підвищити точність та ефективність побудови діаграм класів із вимог користувача. Також доцільно впровадити власні правила або моделі для покращення розпізнавання структур у текстах англійською мовою.

Результати роботи [17] показують ефективність використання Graph Neural Networks (GNN) для автоматичного генерування UML-діаграм класів на основі вихідного коду в контексті зворотної інженерії та Model Driven Architecture. Запропонований підхід значно зменшує час і зусилля, необхідні для побудови діаграм вручну, забезпечує високу точність у відображенні структури, зв’язків та асоціацій у програмному забезпеченні. Також підкреслюється користь такої автоматизації для поліпшення налагодження, технічного обслуговування та передачі знань між членами команди.

У роботі [18] описується створення автоматизованого фреймворку для генерації діаграм варіантів використання та класів з текстових вимог, написаних природною мовою. Запропонований підхід поєднує методи обробки природної мови (таких як токенізація та POS-аналіз) із евристичними правилами та алгоритмом k-ближчих сусідів (k-NN) для підвищення точності виділення ключових елементів. Експерименти показали високі показники якості — середню повноту 96% і точність 92%, що свідчить про ефективність системи. Крім того, фреймворк здатен враховувати семантичні зв’язки в тексті, що підвищує його здатність до логічного аналізу вимог.

Робота [19] описує розробку системи READ, яка автоматично генерує UML-діаграми класів із текстових вимог, написаних англійською мовою,

використовуючи методи обробки природної мови (NLP) та доменну онтологію. Система успішно витягує назви класів, атрибути, методи та зв’язки між класами, що значно скорочує ручні зусилля та час, необхідний для створення моделей у процесі розробки ПЗ. READ реалізовано на Python і протестовано на загальнодоступних стандартних наборах даних. Результати експериментів показали, що запропонований підхід перевершує наявні методи об’єктно-орієнтованого проєктування за точністю та ефективністю.

Результати роботи [20] полягають у створенні інструменту Class Diagram Critic, який автоматично аналізує UML-діаграми класів і надає рекомендації щодо покращення їхньої структури відповідно до заданих правил і принципів об'єктно-орієнтованого проєктування. Інструмент допомагає виявити типові помилки, такі як неправильні зв’язки між класами, надмірна дублюваність та слабка модульність, що особливо корисно для початківців.

1.4. Мета і завдання кваліфікаційної роботи

Метою кваліфікаційної роботи є вивчення наявних моделей штучного інтелекту для розпізнавання природної мови, розроблення ПЗ, яке реалізовує створену модель і дозволяє користувачам будувати специфікацію діаграми класів за допомогою голосового вводу англійською мовою, а також враховує помилки моделі під час розпізнавання ключових слів діаграми класів, яких не існує в англійській мові, наприклад int.

На основі визначеної мети сформовано перелік завдань:

- Провести огляд наукових джерел за темою;
- Здійснити аналіз наявних підходів до розпізнавання природної мови;
- Реалізувати і протестувати модель;
- Розробити ПЗ для будування специфікації діаграми класів; інтегрувати модель з розробленим ПЗ.

Об'єктом дослідження є процес розпізнавання природної мови, враховуючи додаткові семантичні обмеження.

Предметом дослідження є побудова специфікації діаграми класів засобами штучного інтелекту для розпізнавання природної мови.

Новизна полягає у застосуванні штучного інтелекту для розпізнавання людської мови для побудови специфікації діаграми класів.

1.5. Висновки

Аналіз існуючих методів та підходів вказує на значний потенціал для вдосконалення за допомогою новітніх технологій, особливо штучного інтелекту. Розвиток більш ефективних, інтуїтивно зрозумілих та гнучких інструментів для проектування може значно поліпшити процеси розробки ПЗ, зменшуючи часові та фінансові витрати.

Можна констатувати, що сфера проектування програмного забезпечення знаходиться на порозі значних змін з появою нових технологій, таких як штучний інтелект. Поточні недоліки в існуючих методах, а саме занадто велика частина мануальних рутинних задач в процесі, який в цілому є творчий, небезпека допустити незначні і не помітні одразу помилки, а також значні витрати часу, відкривають широкі можливості для інновацій та покращення, що можуть бути реалізовані в рамках цього дипломоного проекту. Зокрема, замість використання ШІ для автоматичного створення діаграми класів, де ШІ може легко помилитись у зв'язках або невірно зрозуміти предметну область, дана робота зосереджується на тому, щоб дати користувачу можливість самостійно продумати і голосом ввести діаграму. Результати роботи можна поєднати з іншими проектами, зокрема можна доповнити автоматичною побудовою діаграми класів з подальшою можливістю вносити швидкі правки голосовим вводом.

РОЗДІЛ 2. РОЗРОБКА РОЗВ'ЯЗКУ ЗАДАЧІ ПОБУДОВИ СПЕЦИФІКАЦІЇ ДІАГРАМИ КЛАСІВ ЗАСОБАМИ ШІ ДЛЯ РОЗПІЗНАВАННЯ ПРИРОДНОЇ МОВИ

2.1. Семантичний аналіз задачі

2.1.1. Сематична мережа

У результаті аналізу предметної області “Побудова діаграми класів засобами ШІ” отримано такий перелік інформаційних об'єктів (рис. 2.1):

- Природня мова;
- Англійська мова;
- Текст англійською мовою;
- Слово;
- Морфема;
- ПЗ;
- ШІ;
- ПЗ побудови діаграми класів;
- Людина;
- Клавіатура;
- Мишка;
- Діаграма класів;
- Клас;
- Атрибут;
- Метод.

Отримані інформаційні об’єкти можна поділити за характеристиками на узагальнені, конкретні та агрегатні:

Узагальнені: Природня мова, Текст, Слово, Морфема, ПЗ, Людина, Клавіатура, Мишка, Діаграма класів, Клас, Атрибут, Метод, Видимість атрибутів і методів, Зв’язки між класами - на мою думку ці об’єкти є узагальненими, так як вони представляються цілий клас об’єктів даного проблемного середовища.

Конкретні: Англійська мова, ШІ, ПЗ побудови діаграми класів - ці об’єкти є конкретними в межах класу наших об’єктів, так як вони конкретизують їхні характеристики.

Агрегатні: Клас (складається з атрибутів і методів).

Приклад відношень “є родом”, “є видом”, “є представником”, “є частиною":

- "Є родом":
    - Англійська мова є родом Природньої мови;
    - ШІ і ПЗ побудови діаграми класів є родом ПЗ.
- "Є видом" (обернений до родового):
    - Природня мова є видом Англійської мови;
    - ПЗ є видом ШІ і ПЗ побудови діаграми класів.
- "Є представником":
    - Текст є представником Англійської мови.
- "Є частиною":
    - Морфема є частиною Слова;
    - Слово є частиною Текста;
    - Атрибут і Метод є частинами Класу.

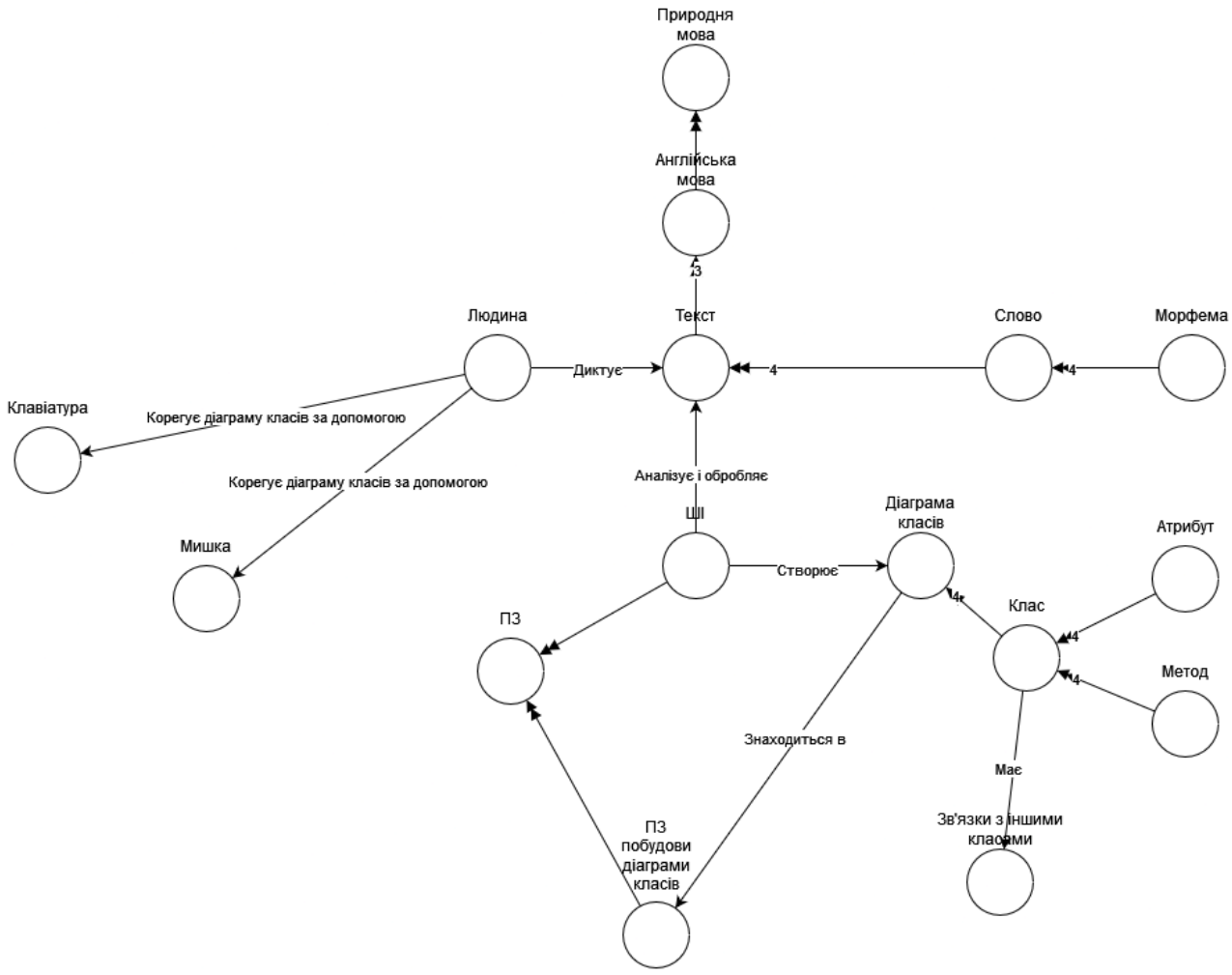

Рис. 2.1. Семантична мережа ПрО "Побудова специфікації діаграми класів засобами ШІ"

2.1.2. Предикатна семантична мережа

З даної семантичної мережі можна виділити такі предикати:

P1: Людина / редагує діаграму за допомогою / миші (рис. 2.2)

P2: Людина / редагує діаграму за допомогою / клавіатури

P10(∧(P1, P2)): Людина / редагує діаграму за допомогою / миші AND клавіатури

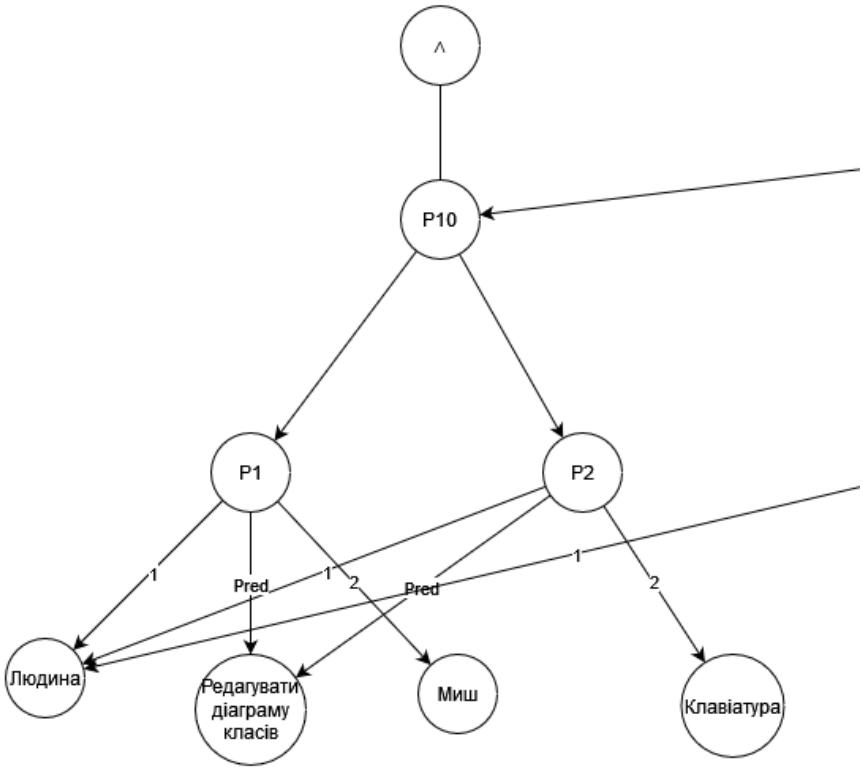

Рис. 2.2. Предикатна семантична мережа ПрО "Побудова специфікації діаграми класів засобами ШІ". Частина 1

P3: Людина / диктує / англомовний текст (рис. 2.3)

P4: ШІ /аналізує і обробляє / англомовний текст

P5: ШІ / створює / діаграму класів

P11(∨(P3, P10)): (Людина / редагує діаграму за допомогою / миші AND клавіатури) OR Людина / диктує / англомовний текст

P12(∧(P11, P4, P5, P6, P9)): ((Людина / редагує діаграму за допомогою / миші AND клавіатури) OR Людина / диктує / англомовний текст) AND (ШІ /аналізує і обробляє / англомовний текст) AND (ШІ / створює / діаграму класів) AND (Діаграма класів / складається з / класів) AND (Клас / складається з / атрибутів AND методів)

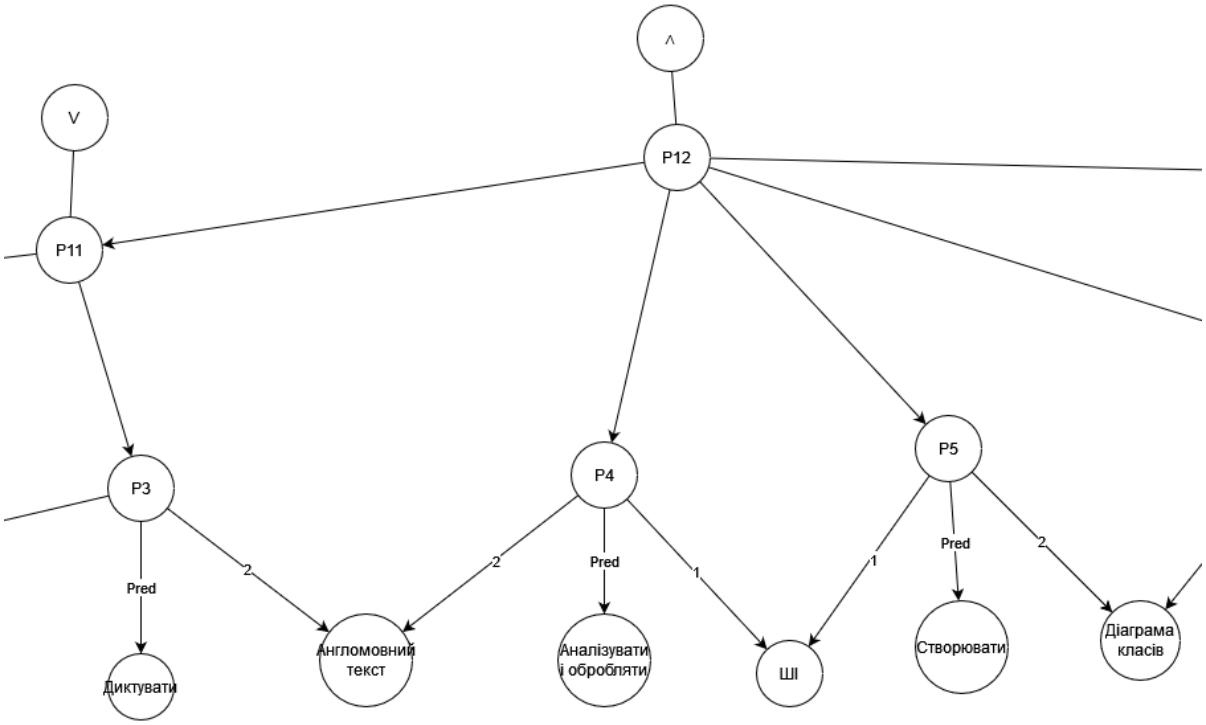

Рис. 2.3. Предикатна семантична мережа ПрО "Побудова діаграми класів засобами ШІ". Частина 2

P6: Діаграма класів / складається з / класів (рис. 2.4)

P7: Клас / складається з / атрибутів

P8: Клас / складається з / методів

P9(∧(P7, P8)): Клас / складається з / атрибутів AND методів

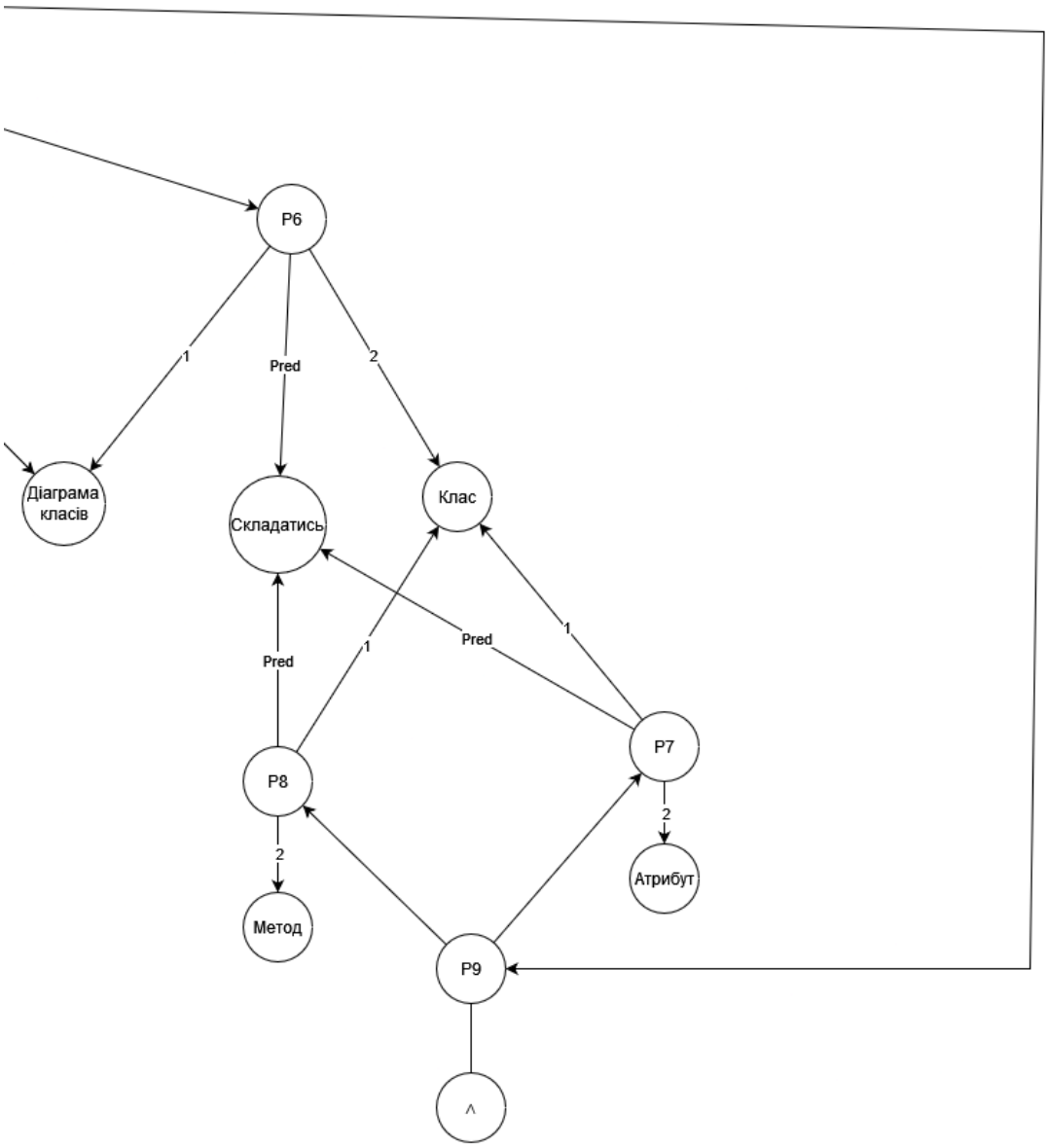

Рис. 2.4. Предикатна семантична мережа ПрО "Побудова специфікації діаграми класів засобами ШІ". Частина 3

2.1.3. Атрибутивна семантична мережа

Об'єкт визначається за допомогою атрибутів і їхніх доменів. Атрибути складають постійну частину, а домени – змінну. Таким чином об'єкт задається і статично, і динамічно. При представленні конкретного об'єкту фіксується місце кожного атрибуту в загальному описі і це дає змогу зробити згортання унарних відношень в n-арне відношення. Ця згортка необхідна по тій причині, що вона дуже економить пам'ять і значно підвищує швидкодію. Ім'я атрибута при цьому розуміють за замовчуванням, а на місцях відповідних позицій стоять конкретні значення атрибутів. Таким чином, в атрибутивних семантичних мережах кожна

вершина визначається як деяке відношення з набором атрибутів і їх доменами: R = (<$A_1, domain(A_1)$>, <$A_2, domain(A_2)$>, ..., <$A_n, domain(A_n)$>).

На основі цього узагальненого представлення можна побудувати множину конкретних описів, які називаються фактами.

Визначемо об'єкти через їхні атрибути і домени (рис. 2.5 і рис. 2.6):

- Людина (Чіткість вимови, Швидкість вимови)
    - Домен Чіткість вимови: чітка, нечітка.
    - Домен Швидкість вимови: швидка, середня, повільна.
- ПЗ (Модель розповсюдження)
    - Домен Модель розповсюдження: open-source, subscription, turnkey.
- Морфема (Тип)
    - Домен Тип: префікс, суфікс.
- ШІ (Тип)
    - Домен Тип: Машинне навчання, робототехніка, обробка природньої мови, експертні системи.
- Клавіатура / Миш (Модель, Рік випуску, Тип, Тип підключення)
    - Домен Модель: Bloody R30, HP S450.
    - Домен Рік випуску: 2001, 2007.
    - Домен Тип: ігрова, офісна.
    - Домен Тип підключення: PS/2, USB.

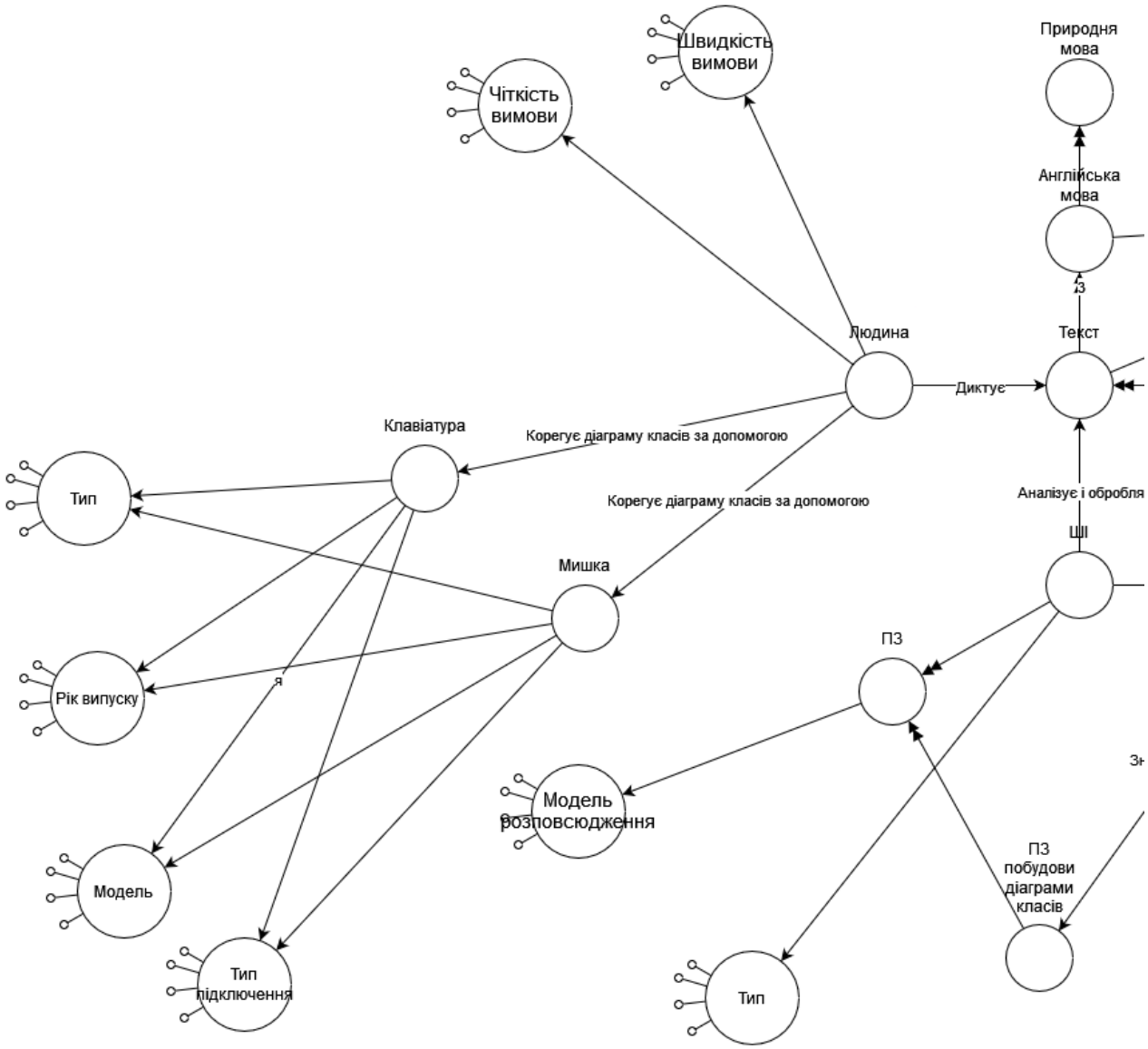

Рис. 2.5. Атрибутивна семантична мережа ПрО "Побудова специфікації діаграми класів засобами ШІ". Частина 1

- Англійська мова (Діалект)
    - Домен Діалект: ланкашир, дорсет.
- Текст (Величина)
    - Домен Величина: велика, середня,мала.
- Слово (Мова)
    - Домен Мова: англійська, українська.
- Клас / Атрибут / Метод (Видимість, Назва)
    - Домен Видимість: private, public, protected.
    - Домен Назва: EnctyptionOperations, AESImplementation.

- Атрибут (Тип)
    - Домен Тип: int, double, string.
- Метод (Тип повернення, Кількість параметрів)
    - Домен Тип повернення: int, double, string.
    - Домен Кількість параметрів: 2, 4.

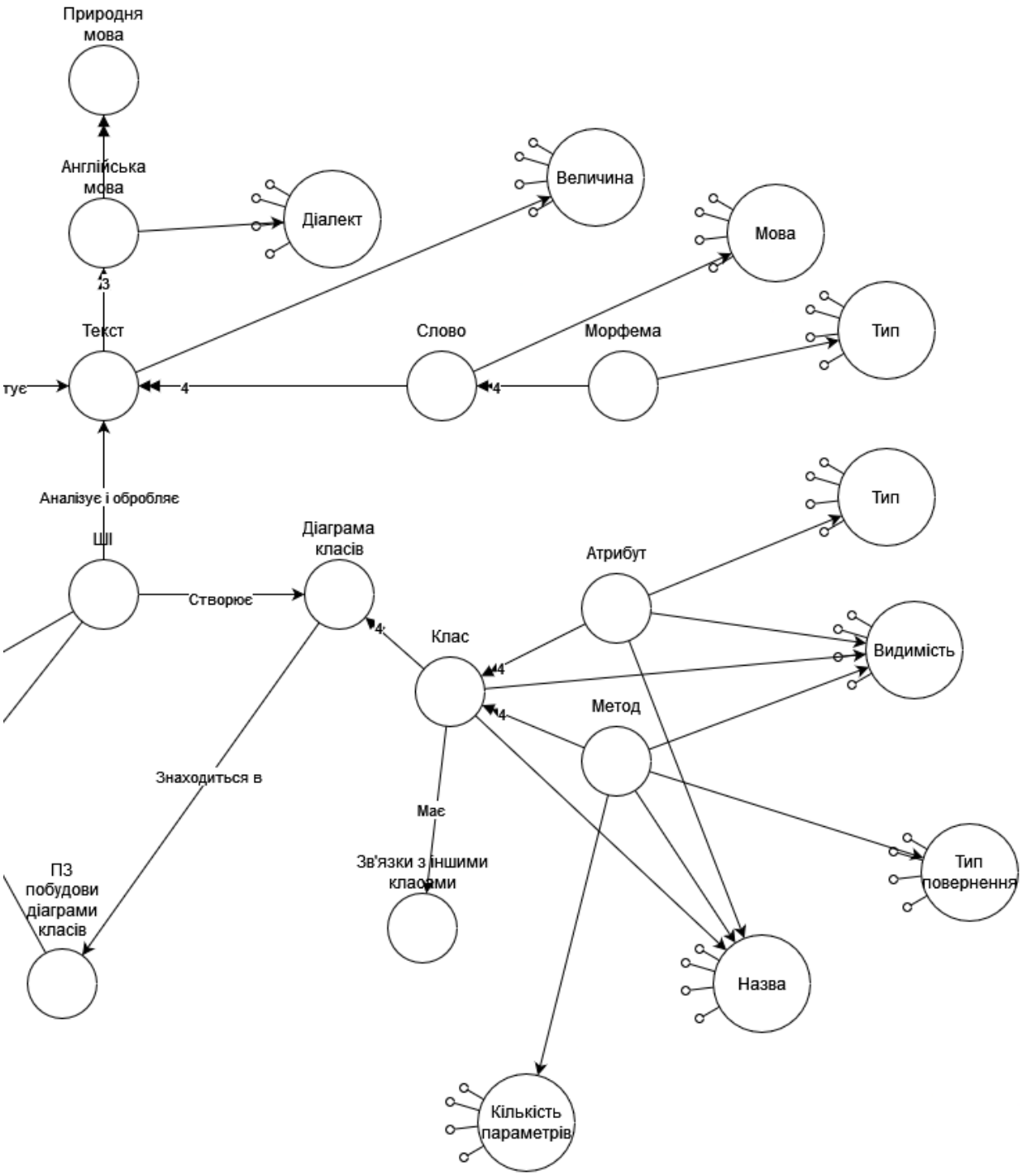

Рис. 2.6. Атрибутивна семантична мережа ПрО "Побудова специфікації діаграми класів засобами ШІ". Частина 2

При роботі в семантичних мережах велике значення мають принципи наслідування властивостей від класу об'єктів до конкретних представників класу: клас-підклас, клас-екземпляр. Можна сказати, що наслідування – це спосіб, у який переходить значення властивостей від одного об’єкту до іншого. Основним представником наслідування є ієрархія понять, яка будується за родо-видовим відношенням, а також відношенням "є частиною", які представлені на рис 2.1 подвійними, потрійними і четверними стрілками.

Клас є узагальненням, він зберігає інформацію, яку мають всі об'єкти даного класу, тому об'єкти класу можуть наслідувати, коли потрібно, від класу цю інформацію, тобто її не треба виписувати для кожного об'єкту окремо. Це значно зменшує розмір атрибутивних семантичних мереж.

Наслідування властивостей означає, що властивості, приписані класу більш високого рівня, автоматично присвоюються об'єктам більш низького рівня за замовчуванням, але треба зважати на існування деяких особливостей.

- Відношення клас-підклас (відношення спеціалізації). Тут властивості та значення властивостей наслідуються за замовчуванням за зв'язком спеціалізації IS A. Зворотне відношення підклас-клас – це відношення узагальнення. Таким чином, підклас спеціалізує, а клас узагальнює.
- Відношення клас-об'єкт. Це також відношення типу IS A і дозволяє наслідувати властивості за замовчуванням. Це відношення називається відношенням класифікації. Зворотне відношення об'єкт-клас – це відношення інтеграції.
- Відношення об'єкт-підоб'єкт або екземпляр-властивості. Це відношення декомпозиції, яке задає розбиття об'єкта на складові частини. Властивості класу не наслідуються їхніми частинами. Зворотне відношення називається агрегуванням. Для наслідування властивостей частинами у відношенні агрегування повинні бути присутні або всі частини, або деякий обов'язковий мінімум цих частин.

2.1.3.1. Приклади наслідування властивостей в атрибутивній семантичній мережі ПрО "Побудова діаграми класів засобами ШІ"

Принцип наслідування властивостей полягає в тому, що всі властивості родового поняття, як правило, присутні і у видовому. У даній предметній області “Побудова діаграми класів засобами ШІ” можна навести такі приклади наслідування властивостей. Оскільки "ПЗ" є родом для "ШІ" та "ПЗ побудови діаграми класів", всі властивості, притаманні об’єкту "ПЗ", притаманні як "ШІ", так і "ПЗ побудови діаграми класів", а саме "Модель розповсюдження".

Але оскільки видове поняття є багатшим за змістом, і родове поняття не охоплює повністю всіх його властивостей, то наприклад, "ШІ" має ще й метод машинного навчання, якого може не мати "ПЗ".

Аналогічно, "Текст" наслідує властивості поняття "Англійська мова" як родовою для них, тобто він має атрибут "Діалект". Водночас "Англійська мова" має і власні атрибути, наприклад "Наявність суфіксів", "Наявність префіксів".

2.1.4. Виведення на семантичних мережах

Для виведення логічного висновку в семантичних мережах виконується порівняння графових структур. Для цього вузли двох структур порівнюють один за одним поки не буде знайдено співпадіння. Якщо знайдено неспівпадання, то здійснюється повернення до того місця, де було останнє співпадання, і відбувається порівняння з іншими елементами.

2.1.4.1. Приклади виведення на семантичних мережах

1. Чи має атрибут класу видимість (рис. 2.7)?

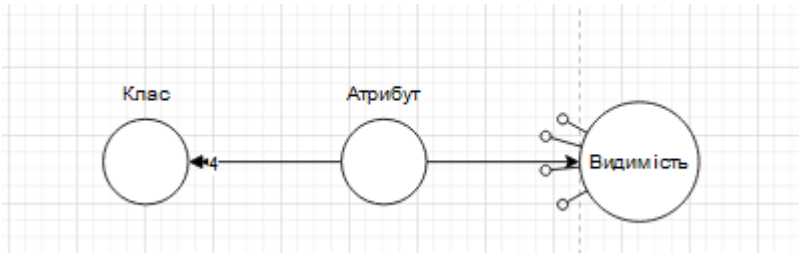

Рис. 2.7. Пошук "Чи має атрибут класу видимість"

Так, має (рис. 2.8):

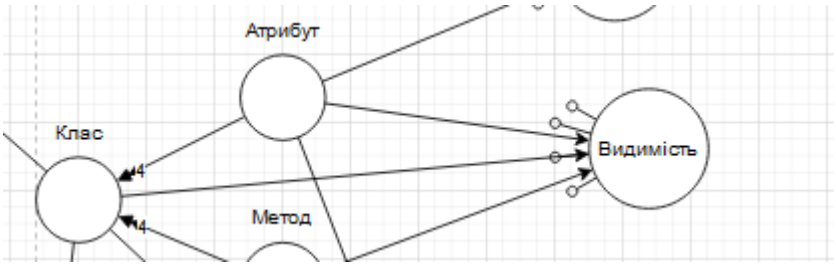

Рис. 2.8. Результат пошуку "Чи має атрибут класу видимість"

2. **Чи має текст англійською мовою діалект (рис. 2.9)?**

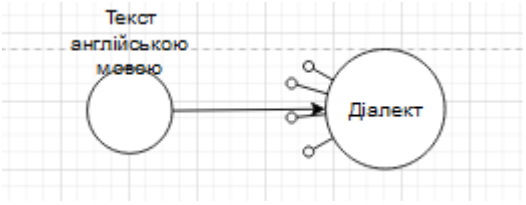

Рис. 2.9. Пошук "Чи має текст англійською мовою діалект"

Так, має (рис. 2.10):

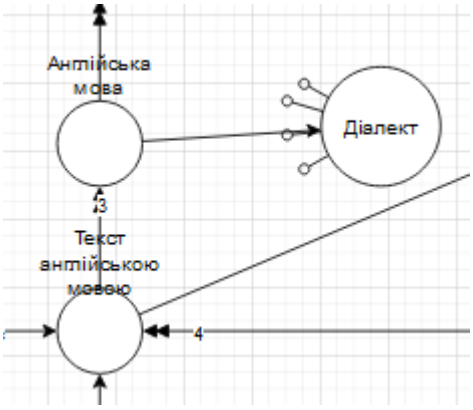

Рис. 2.10. Результат пошуку "Чи має текст англійською мовою діалект"

3. **Чи має природня мова діалект (рис. 2.11)?**

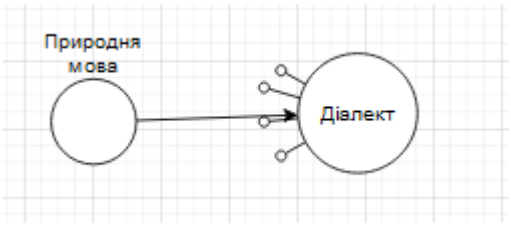

Рис. 2.11. Пошук "Чи має природня мова діалект"

Ні, сама по собі не має (лише підклас Англійська мова має) (рис. 2.12):

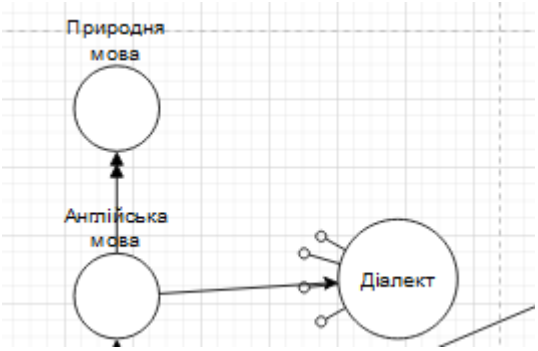

Рис. 2.12. Результат пошуку "Чи має природня мова діалект"

Таким чином знаходяться відповіді на питання в семантичних мережах.

2.2. Побудова моделі перетворення голосу в текст

2.2.1. Навчання моделі

Для побудови діаграми класів голосом першим ділом потрібно перетворити голос в текст (speech to text). Для цього існує спеціальний клас методів машинного навчання. Загалом процес створення моделі виглядає наступним чином:

1. Зібрати дані для навчання і тестування
   1.1. Треба зібрати великий набір аудіозаписів з мовленням людини. Для покращення узагальнення набір даних має включати різноманітних мовців, акценти та середовища;
   1.2. До зібраних даних необхідно прописати транскрипції. Іншими словами, використовується навчання з вчителем.
2. Попередня обробка даних
   2.1. Аудіофайли конвертуються в один формат (наприклад, WAV 16 кГц). За потреби нормалізується звук і виконується шумозаглушення. Важливо відзначити, що попередня обробка даних також застосовується до аудіофайлів перед тим, як вони потраплять в натреновану модель.
3. Тренування моделі
   3.1. Потрібно вибрати модель навчання. Для speech to text моделей в основному застосовуються моделі Long Short-Term Memory (LSTM) та Gated Recurrent Units (GRUs);

3.2. Як функція втрат (Loss function) добрі результати показує Connectionist Temporal Classification (CTC);

3.3. Вибірку даних необхідно поділити на навчальну і тестову. На навчальній вибірці модель тренується. На тестовій вибірці перевіряється точність моделі;

3.4. Проводиться тренування моделі.

4. Оцінювання результатів і підгонка

4.1. На тестовій вибірці перевіряються два важливі аспекти:

4.1.1. Відсутність недонавчання (underfitting): модель має показувати високу точність розпізнавання мови (більше 80%);

4.1.2. Відсутність перенавчання (overfitting): коли модель перенавчається, вона "запам'ятовує" навчальну вибірку і не володіє властивістю узагальненості, що означає, що вона показує високу точність на навчальній вибірці, але низьку на тестовій;

4.1.3. Використовуючи точність розпізнавання як метрику, інженер експериментує з різними моделями навчання і параметрами вибраних моделей, серед яких найбільшої уваги вартує приділити трьом:

4.1.3.1. Learning rate - наскільки сильно міняються параметри моделі;

4.1.3.2. Epochs (скільки разів модель має обійти всі дані під час навчання;

4.1.3.3. Batch size - на скільки частин поділити дані для навчання – на цих частинах модель навчається окремо. Загальне правило: що менше batch size, то більшою мірою навчена модель володіє властивістю узагальнення.

4.1.4. Також вартує аналізувати, які помилки робить модель при розпізнаванні – це показує ті області, на які вартує приділити увагу.

2.2.2. Модель розпізнавання тексту

На основі інформації, отриманої під час дослідження предметної області, а також оцінки наявних апаратних ресурсів для магістерської кваліфікаційної роботи, стало зрозуміло, що є потреба використовувати вже натреновану модель

по причині значних обмежень ресурсів в ситуації, де висуваються значні вимоги до апаратного забезпечення для навчання моделі. Більше того, немає сенсу робити те, що вже існує і добре працює.

Провівши дослідження різних моделей перетворення голосу в текст, обрано Whisper [25]. Whisper — це модель розпізнавання мовлення (ASR) загального призначення, розроблена OpenAI, яка перетворює усне мовлення в текст. Вона натренована на великому наборі різноманітних аудіо, а також є багатозадачною моделлю, яка може виконувати розпізнавання мовлення на багатьох мовах, перекладати мовлення на різні мови, а також розпізнавати, якою мовою говориться у аудіо. Модель працює з аудіо і відео в багатьох форматах і частотою дискретизації завдяки ffmpeg.

Whisper є загальнодоступним (open-source) і за ліцензією його можна використовувати, модифікувати і використовувати комерційно за умови включення у продукт ліцензії [26].

2.2.2.1. Архітектура Whisper

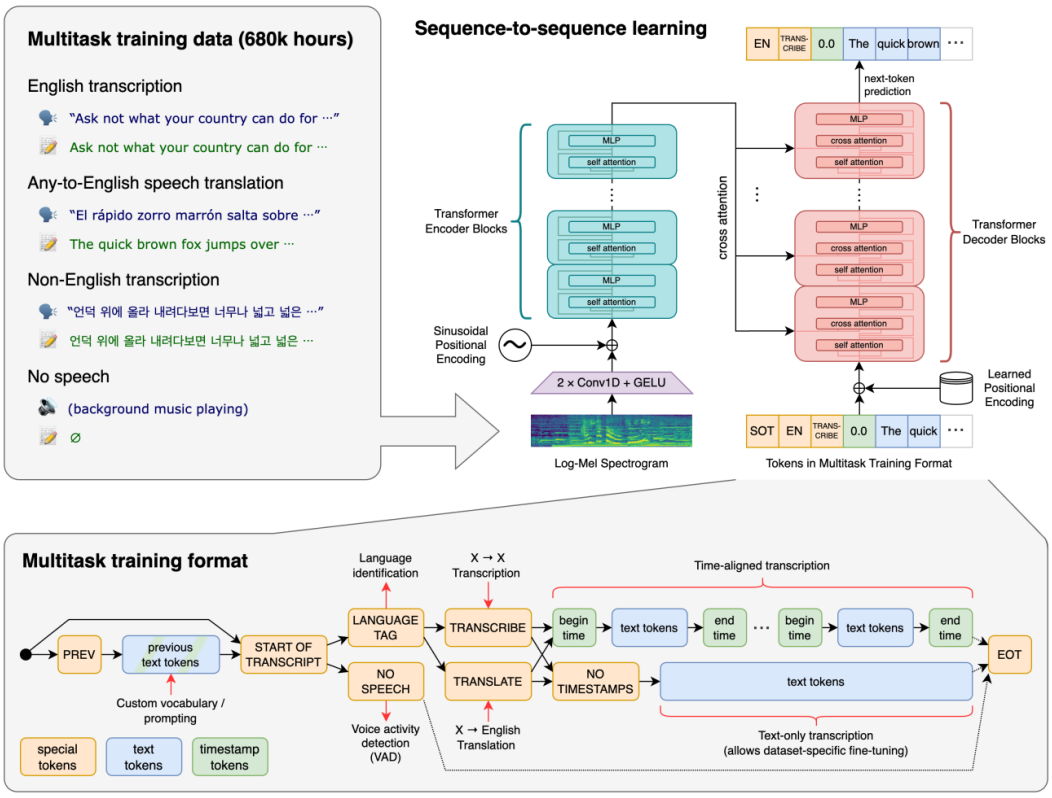

Рис. 2.13. Архітектура Whisper [25]

- Модель трансформера

Whisper базується на архітектурі трансформера, яка складається з енкодера (encoder) та декодера (decoder). Він має такі основні компоненти (рис. 2.13):

- Механізм багатоголової самоуваги (Multi-head Self-Attention Mechanism) - дозволяє моделі фокусуватися на різних частинах аудіовведення (вхідної послідовності), тим самим захоплюючи різні контекстуальні взаємозв'язки в цих даних. Це покращує розпізнавання складних шаблонів і довготривалих залежностей.
- Мережі прямого розповсюдження - кожен шар трансформера включає повнозв'язні мережі прямого розповсюдження, які додатково обробляють вхідні дані.

Архітектура енкодер-декодер є універсальною і може бути адаптована до різних завдань ASR, включаючи розпізнавання різних мов, шумні середовища та роботу з термінами, специфічними для конкретного домену.

- Налаштування енкодера-декодера
  - Енкодер - обробляє вхідні мел-спектрограми і створює послідовність контекстно-насичених представлень. Він складається з декількох шарів, кожен з яких включає механізм самоуваги та мережі прямого розповсюдження. Кожен шар енкодера покращує представлення вхідних даних, враховуючи контекст інших частин послідовності.
  - Декодер - генерує вихідну транскрипцію з представлень енкодера. Він також використовує самоувагу та увагу до виходів енкодера.
- Представлення входу

Whisper використовує мел-спектрограми як вхідні дані. Мел-спектрограми є представленням аудіосигналу, що відображає його частотний вміст у часі.

2.2.2.2. Принцип роботи Whisper

1. Попередня обробка - вхідний аудіо сигнал конвертується у мел-спектрограму. Цей процес включає кілька етапів: розбиття аудіосигналу на

частини – так звані фрейми (frame), – застосування перетворення Фур'є для переведення його у частоти, і відображення частот на мел-шкалу.

2. Видобування фіччерів - мел-спектрограма подається на вхід енкодеру трансформера, який обробляє її через багато шарів для видобутку високорівневих фіччерів. Кожен шар застосовує самоувагу для фокусування на різних частинах послідовності та використовує мережі прямого розповсюдження для подальшого удосконалення фіччерів.
3. Кодування - енкодер генерує послідовність прихованих станів, які містять контекст аудіомовлення. Ці приховані стани передаються декодеру.
4. Декодування - декодер генерує транскрипцію токен за токеном. На кожному кроці він використовує самоувагу для врахування раніше згенерованих токенів і увагу до виходів енкодера для фокусування на істотних частинах вхідних даних.
5. Постобробка - вихідні токени складаються у слова та речення (тобто вихідний текст). Також на цьому етапі можливо застосувати мовні моделі для покращення тексту.

2.2.2.3. Швидкість обробки тексту

В офіційному репозиторії Whisper [25] міститься перелік моделей з їхніми вимогами до віртуальної пам'яті. Що більше у моделей параметрів, то більші вимоги до апаратного забезпечення і менша швидкість роботи, натомість точність розпізнавання збільшується (рис. 2.14).

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	`tiny.en`	`tiny`	~1 GB	~32x
base	74 M	`base.en`	`base`	~1 GB	~16x
small	244 M	`small.en`	`small`	~2 GB	~6x
medium	769 M	`medium.en`	`medium`	~5 GB	~2x
large	1550 M	N/A	`large`	~10 GB	1x

Рис. 2.14. Варіації моделі Whisper [25]

Для знаходження компромісу між швидкістю і точністю при наявних ресурсах, записано голосове повідомлення, яке передано на вхід моделям tiny, base і small.

Вхідний текст: I need class A with attributes string name and string surname.

Tiny model: I need class A with attributes string name and string cell name.

Base model: I need class A with attribute's string name and string cell name.

Small model: I need class A with attributes string name and string surname.

Вхідні дані звучали нечітко і достатньо тихо. В той же час, враховуючи це, модель показала вражаючі результати. Проте видно, що зі словом "surname" у модель виникли труднощі і лише Small версія змогла розпізнати слово правильно. При цьому затрачено 8 секунд часу на обробку. Доцільно зробити Base модель базовою для програми, але дати користувачу можливість динамічно міняти варіант моделі Whisper.

2.3. Перетворення тексту в компоненти діаграми класів і побудова її специфікації

Після того, як Whisper перетворить мовлення людини в текст, цей текст подається на вхід іншому програмному модулю, який на виході міститиме специфікацію класів і їхнього наповнення (методів і атрибутів). Побудова діаграми класів включає в себе специфічні терміни, які необхідно розпізнати. Наприклад, "Class", "Method", "Attribute", "int", "Return(s)". На цьому етапі програма знаходить у тексті ці специфічні слова і відповідно до цього формує специфікацію вихідних класів і наповнення. Так як користувачу не обов'язково описувати одразу всю діаграму, а можна її будувати частинами, то на цьому етапі на вході також подається специфікація вже існуючої діаграми класів, а специфікація поточної частини об'єднується з вже існуючою перед тим, як її передавати наступному модулю.

З використанням сформованої специфікації класів і їхнього наповнення, а також наперед визначених правил побудови діаграми класів, програма конструює діаграму і виводить на екран.

2.4. Повний процес побудови специфікації діаграми класів засобами штучного інтелекту

Процес роботи користувача з програмним забезпеченням виглядає наступним чином: користувач натискає на кнопку вводу, словесно голосом описує частину діаграми класів, ще раз натискає кнопку вводу. Програма зберігає введений голосовий опис в пам'яті комп'ютера і передає Whisper. Останній конвертує мовлення в текст і створює файл з розширенням "txt", який містить опис користувача у вигляді тексту. Програма зчитує цей текст і складає специфікацію побудови діаграми класів наперед визначеним інтерфейсом, знаходячи необхідні компоненти за специфічними словами. Далі ПЗ будує діаграму класів і виводить на екран. Після цього користувач знову натискає кнопку вводу і диктує наступну частину діаграми і, за потреби, зміни, які треба внести до поточної версії.

2.5. Висновки

В цьому розділі побудовано семантичну мережу предметної області для поглиблення її розуміння і розглянуто методи роботи нейронних мереж, процес їхнього навчання разом з методами попердньої обробки даних. Вибрано технології реалізації моделі, такі як Python, PyTorch, ffmpeg, розглянуто потребу в апаратному забезпеченні. На основі цього прийнято рішення використовувати натреновану модель Whisper з архітектурою трансформера, що складається з енкодера і декодера, що дозволяє ефективно розпізнавати мовлення навіть в шумних середовищах.

Вирішивши задачу перетворення мовлення в текст, розглянуто спосіб пошуку компонентів діаграми класів за допомогою виділення ключових, специфічних для даної предметної області, слів. З цих компонентів формується специфікація діаграми класів за наперед визначеним інтерфейсом. Модуль виведення, використовуючи специфікацію, будує і показує діаграму класів користувачу, який може або редагувати, або додати нові компоненти до діаграми своїм голосом.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ПОБУДОВИ СПЕЦИФІКАЦІЇ ДІАГРАМИ КЛАСІВ ЗАСОБАМИ ШІ ДЛЯ РОЗПІЗНАВАННЯ ПРИРОДНОЇ МОВИ

3.1. Призначення та мета ПЗ

Програмна система для побудови діаграм класів засобами штучного інтелекту призначена для автоматизації процесу створення UML-діаграм класів на основі природної мови, розпізнаної голосом. Основною метою розробленої системи є спрощення та пришвидшення роботи архітекторів програмного забезпечення шляхом усунення значної частини рутинних, ручних операцій з введення інформації про класи, атрибути, методи та зв'язки між ними. Завдяки використанню технологій розпізнавання природної мови та автоматичної інтерпретації тексту, система дозволяє архітектору сфокусуватись на творчих та концептуальних аспектах проектування програмного продукту.

Ключовою перевагою програмної системи є можливість голосового введення англійською мовою, що суттєво підвищує ефективність проектувальних робіт. В результаті цього користувач має змогу описати необхідні класи та їх складові усно, без потреби ручного набору тексту, що знижує ймовірність механічних помилок та сприяє більш точній реалізації задумів архітектора.

Цільовою аудиторією розробленої програмної системи є архітектори програмного забезпечення, системні аналітики, розробники та інші спеціалісти у сфері програмної інженерії, які активно використовують UML-діаграми у своїй професійній діяльності. Також система може стати у нагоді студентам для вивчення семантики побудови діаграми класів. Система забезпечує зручний та інтуїтивно зрозумілий інтерфейс, що не потребує спеціалізованих знань у сфері штучного інтелекту або голосових технологій.

Основні завдання, які виконує дана програмна система, включають:

- Перетворення голосового вводу користувача в текстову форму за допомогою нейромережевих моделей;

- Виділення ключових елементів діаграми класів з отриманого тексту, таких як класи, атрибути, методи, їхні типи та параметри;
- Формування структурованої специфікації UML-діаграми класів;
- Можливість подальшого редагування та доповнення специфікації діаграми класів через повторні голосові команди.

Використання системи дозволяє значно скоротити час на створення діаграм класів, мінімізувати людський фактор у вигляді механічних помилок, а також підвищити продуктивність роботи архітекторів програмного забезпечення. Загальна мета полягає у покращенні процесу проектування програмних систем завдяки інтеграції сучасних технологій штучного інтелекту та розпізнавання природної мови.

3.2. Класи користувачів та їхні потреби

Основними користувачами програмної системи є:

Архітектори програмного забезпечення – це спеціалісти, які відповідають за проектування загальної структури програмних систем. Вони використовують систему для швидкого створення первинних діаграм класів, які надалі деталізуються в процесі розробки проекту.

Розробники програмного забезпечення - використовують систему для зручного отримання та уточнення інформації про архітектуру майбутнього ПЗ, що дозволяє їм краще розуміти вимоги до реалізації та інтеграції.

Студенти – використовують систему для вивчення правил побудови діаграми класів, особливо при складних зв'язках.

Бізнес-аналітики та менеджери проектів - можуть використовувати систему для швидкого формування і перегляду початкових вимог і моделей майбутньої системи без глибокого занурення у технічні деталі.

3.3. Середовище функціонування

Розроблена програмна система призначена для функціонування на персональних комп'ютерах під управлінням операційної системи Windows 10 і

вище. Для забезпечення її роботи використовуються сучасні апаратні компоненти, які відповідають наступним вимогам:

- Процесор: Intel Core i5 або еквівалентний AMD;
- Оперативна пам'ять: 8 Гб або більше;
- Наявність графічного процесора (GPU), сумісного з CUDA, для оптимальної роботи нейромережевої моделі розпізнавання мовлення;
- Об'єм вільного місця на диску – від 1 ГБ для встановлення програмного забезпечення та зберігання проміжних результатів роботи.

Для роботи з голосовим введенням система використовує зовнішні пристрої – рекомендується мікрофон із стабільною частотою дискретизації 16 кГц для забезпечення точності розпізнавання мовлення.

Програмне забезпечення використовує Python як основну мову розробки через її гнучкість, продуктивність та широку підтримку інструментів машинного навчання, таких як PyTorch та Whisper. Система також використовує утиліту ffmpeg для роботи з аудіофайлами та стандартний набір бібліотек для роботи з графічним інтерфейсом користувача.

3.4. План тестування програмної системи

Тестування розробленої програмної системи спрямоване на підтвердження її відповідності заявленим вимогам, правильності функціонування та виявлення можливих помилок. Для цього розроблено чіткий план тестування, який включає в себе детальний опис кожної вимоги, її пріоритет та кроки перевірки.

3.4.1. Перетворення голосового вводу на текст.

Пріоритет: Високий

Кроки тестування:

1. Запустити програму.
2. Ввести голосову команду англійською мовою через мікрофон.
3. Перевірити, чи правильно програма перетворює мовлення у текст.

3.4.2. Виділення класів із тексту

Пріоритет: Високий

Кроки тестування:

1. Ввести текстову команду, що містить інформацію про класи.
2. Перевірити консольний вивід на відповідність введеним класам.

3.4.3. Виділення атрибутів та методів класів

Пріоритет: Високий

Кроки тестування:

1. Ввести голосову команду з описом атрибутів та методів класів.
2. Перевірити правильність і повноту списку атрибутів і методів у консольному виводі.

3.4.4. Визначення типів і імен складних атрибутів та методів

Пріоритет: Високий

Кроки тестування:

1. Оголосити атрибути та методи класів з типами класів (наприклад, Parent, Person).
2. Перевірити, чи правильно визначено типи і імена у консольному виводі.

3.4.5. Редагування вже створеної діаграми класів

Пріоритет: Середній

Кроки тестування:

1. Побудувати початкову версію діаграми класів.
2. Додати нову інформацію голосовою командою.
3. Перевірити, чи коректно оновилась діаграма в консольному виводі.

3.4.6. Стабільність роботи системи

Пріоритет: Високий

Кроки тестування:

1. Виконати багаторазові послідовні голосові команди.
2. Перевірити, чи програма залишається стабільною, не виникає винятків або помилок.

3.4.7. Реакція на некоректні команди

Пріоритет: Середній

Кроки тестування:

1. Ввести невалідну команду голосом.
2. Перевірити, чи система обробляє помилку коректно, залишаючись працездатною.

3.4.8. Продуктивність системи

Пріоритет: Середній

Кроки тестування:

1. Виміряти час від моменту завершення голосового вводу до появи текстового результату в консольному виводі.
2. Перевірити відповідність часу виконання встановленим вимогам.

3.5. Вимоги зовнішніх інтерфейсів та нефункційні вимоги

3.5.1. Вимоги зовнішніх інтерфейсів:

1. Інтерфейс користувача на першому етапі реалізовано у вигляді консольного додатку. Основними елементами взаємодії є:

- Командний рядок для введення команд.
- Вивід результатів розпізнавання голосу та побудови діаграми класів у консоль.

2. Інтерфейс апаратного забезпечення:

- Мікрофон з рекомендованою частотою дискретизації 16 кГц, для коректного запису та передачі голосових команд.

3. Інтерфейс з програмним забезпеченням сторонніх розробників:

- Використання ffmpeg для попередньої обробки аудіоданих.
- Використання бібліотек Whisper і PyTorch для розпізнавання та обробки природної мови.

3.5.2. Нефункційні вимоги:

1. Продуктивність:

- Час обробки одного типового голосового запиту має бути не більше 10 секунд.
- Відгук на запит користувача повинен бути достатньо швидким для комфортної взаємодії (не більше секунди затримки між натисканням кнопки і початком запису голосу).

2. Надійність:

- Система повинна стабільно функціонувати при багаторазовому повторенні операцій введення, обробки голосових команд та побудови діаграм без критичних помилок або аварійних завершень.

3. Супроводжуваність:

- Код програми повинен бути структурованим, легко підтримуваним та розширюваним. Використовуються зрозумілі імена змінних, класів і методів, а також належна коментованість коду англійською мовою.

4. Зручність використання:

- Інтерфейс повинен бути інтуїтивно зрозумілим для користувачів без спеціалізованої підготовки у сфері ШІ чи машинного навчання.
- Система має інформувати користувача про стан процесу (запису, обробки, побудови) через зрозумілі текстові повідомлення.

5. Масштабованість:

- Система має потенціал до масштабування у випадку збільшення складності оброблюваних діаграм та обсягів голосових команд. Технологічна база дозволяє інтеграцію нових, більш потужних моделей розпізнавання мовлення.

6. Безпека:

- Система забезпечує збереження конфіденційності голосових команд та діаграм класів, не дозволяючи стороннього доступу до даних користувача.

3.6. Обґрунтування вибору технологій для реалізації програмного забезпечення

Для задач машинного навчання в основному прийнято використовувати мову програмування python [21], так як вона має широкі можливості роботи з масивами, велику спільноту інженерів машинного навчання і створені спеціалізовані фреймворки для попередньої обробки даних і тренування моделей.

Одним з фреймворків для навчання моделі є PyTorch [22] — це платформа для глибокого навчання, розроблена лабораторією AI Research від Facebook, яка забезпечує зручний і гнучкий інтерфейс для створення та тренування нейронних мереж. Вона підтримує динамічні обчислювальні графи, що спрощує налагодження та дослідження моделей, а також пропонує потужні інструменти для прискорення обчислень на GPU.

Для можливості обробки великого різноманіття аудіофайлів – з різними форматами, частотою дискретизації – необхідо мати інструмент попередньої обробки даних, який конвертує аудіофайли в один стандартний формат, на якому і буде навчатись і працювати модель. Для цієї задачі обрано ffmpeg [23] - потужний інструмент, який надає широкі можливості редагування аудіо і відео. В даному випадку, ffmpeg застосовується для перетворення аудіо і відео в аудіо з частотою дискретизації 16 кГц. Основна частина інформації людської мови знаходиться в діапазоні від 0 до 8 кГц. За теоремою Найквіста, частота дискретизації має бути як мінімум вдвічі більшою за найвищу частоту в сигналі. Тому 16 кГц є достатнім для задачі розпізнавання мови. З іншої сторони, якщо брати більшу частоту дискретизації, то кількість даних значно зростає, що потребує більших ресурсів що для навчання моделі, що для її використання. Тому 16 кГц є, де-факто, стандартом для моделей розпізнавання мови.

Система ffmpeg може бути встановлена з більшості пакетних менеджерів. В даній роботі вибрано Chocolatey [24]. Він дозволяє встановлювати, оновлювати і видаляти застосунки з командного рядка, що спрощує задачу керування встановленням застосунків порівняно з традиційним ручним методом.

Для розпізнавання голосового вводу була обрана модель Whisper, розроблена компанією OpenAI. Вибір саме цієї моделі зумовлений її високою точністю, широкими можливостями багатомовності, а також стабільністю роботи навіть в

умовах наявності шуму або нечіткої вимови користувача. Whisper має відкритий вихідний код та дозволяє гнучко налаштовувати параметри моделі відповідно до апаратних ресурсів та вимог до швидкості роботи.

Для реалізації графічного інтерфейсу користувача обрано мову програмування C#, яка надає кросплатформенні можливості розробки багатих і складних інтерфейсів.

Для написання коду використовується Visual Studio по причині широких можливостей налаштування, підтримці всіх необхідних мов програмування, доповнення коду і інших технологій, які пришвидшують процес розроблення програмного забезпечення.

Таким чином, вибрані технології дозволяють забезпечити високу продуктивність, точність роботи, легкість підтримки та подальшого розвитку програмної системи, що відповідає початково визначеним вимогам та критеріям якості.

3.7. Архітектура програмного забезпечення

Проект DiagramBuilder – відповідає за перетворення тексту в специфікацію діаграми класів.

- Інтерфейс IDiagramClassElement (рис. 3.1)
    - Інтерфейс, що містить єдину властивість Name, призначений для узагальнення спільних характеристик елементів діаграми класів.
- Class
    - Представляє окремий клас в діаграмі.
    - Має властивості: Name (назва класу), Links (зв’язки між класами), ToDelete (індикатор видалення).
    - Включає колекції атрибутів (ClassAttribute) та методів (ClassMethod).
- ClassAttribute
    - Описує атрибути класів.

- Має властивості: Name (назва атрибуту), Type (тип даних), Visibility (видимість атрибуту), IsClassObject (вказує, чи атрибут є об'єктом іншого класу).

- ClassMethod
  - Представляє методи класів.
  - Включає властивості: Name (назва методу), ReturnType (тип повернення), Visibility (видимість методу), Args (аргументи методу у вигляді колекції ClassAttribute).
- ClassDiagramSpecification
  - Зберігає специфікацію усієї діаграми класів.
  - Містить колекцію класів (Class).

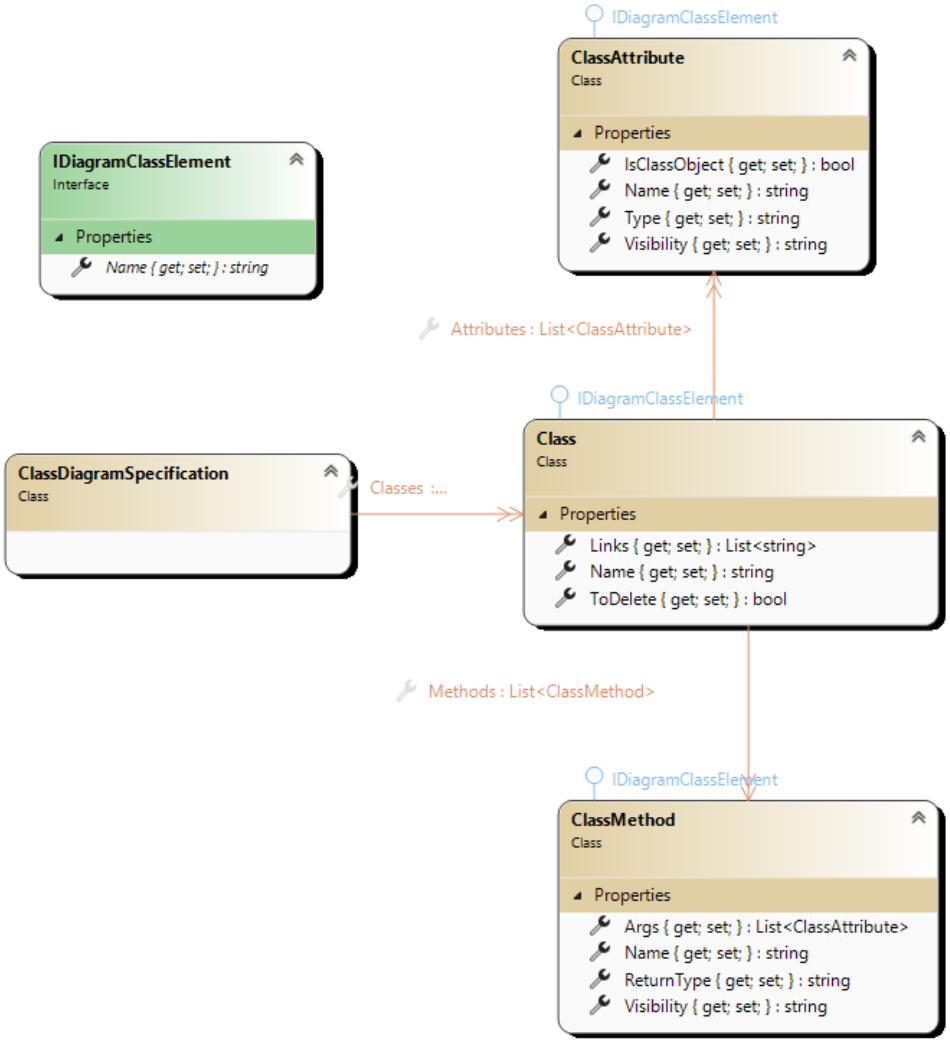

Рис. 3.1. Діаграма класів проекту DiagramBuilder. Частина 1

- ClassDiagramTransformer (рис. 3.2)
  - Відповідає за перетворення тексту у специфікацію класів.
  - Також методи класу реалізують: визначення видимості атрибутів та методів, перетворення JSON-специфікації у об’єкт ClassDiagramSpecification і навпаки.
- ClassDiagramOperations
  - Виконує операції над специфікаціями класів, включаючи злиття, вивід у консоль, перетворення назви відповідно до обраного стилю іменування.
  - Містить поле namingStyle типу NamingStyles.

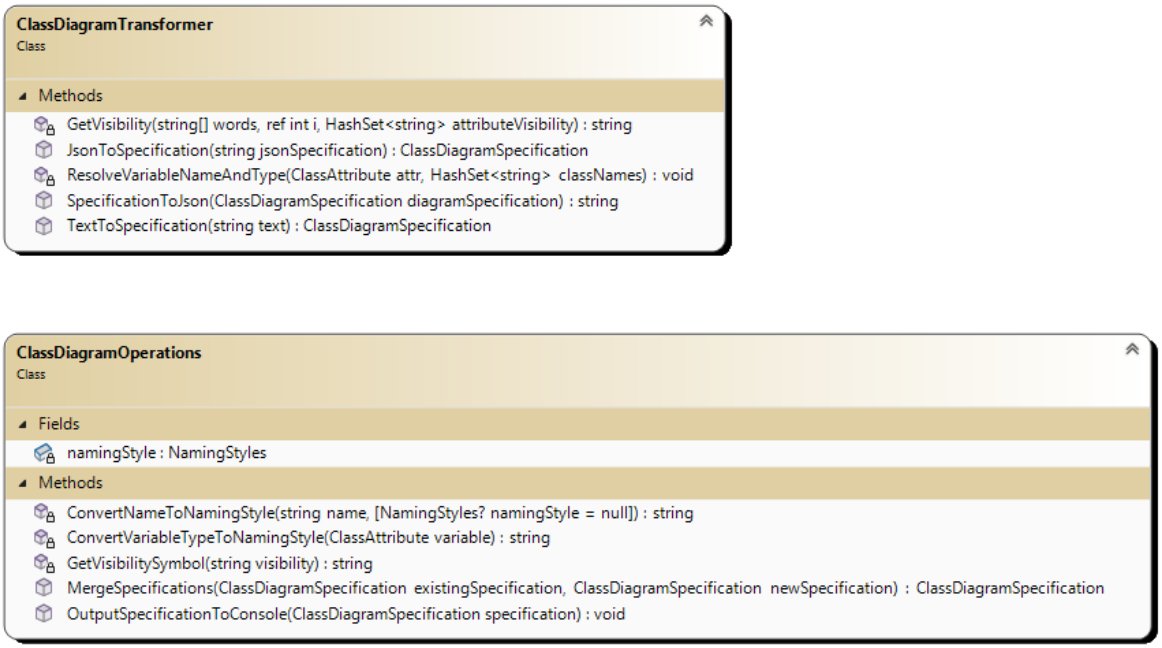

Рис. 3.2. Діаграма класів проекту DiagramBuilder. Частина 2

- NamingStyles (перелік) (рис. 3.3)
  - Перелік стилів іменування елементів: CamelCase, PascalCase, SnakeCase.
- StringExtensions (статичний клас)
  - Має єдиний метод розширення для типу string для перетворення першого символу рядка у верхній регістр.
- WordToNumberConverter

- Включає метод WordsToNumber, що перетворює словесне представлення чисел у цілі числа.

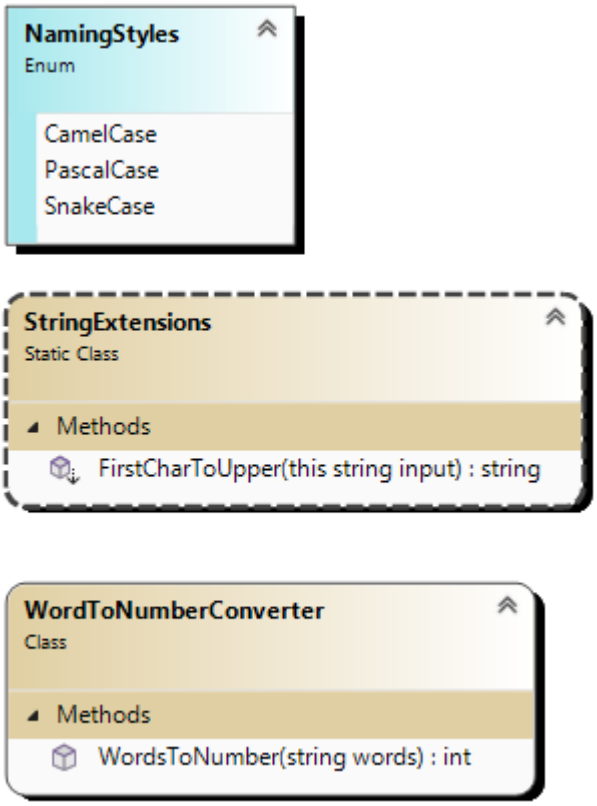

Рис. 3.3. Діаграма класів проекту DiagramBuilder. Частина 3

Проект SpeechRecognition – відповідає за перетворення аудіо в текст.

1. Клас WhisperOperations (рис. 3.4):
   - Виконує операції, пов'язані з перетворенням аудіозаписів у текст за допомогою моделі Whisper.
   - Методи: ConvertAudioToText (приймає шлях до аудіофайлу і шлях до директорії з результатами транскрипції, повертає розпізнаний текст), RunAudioToTextConversion (запускає процес перетворення аудіофайлу в текст і зберігає результат у задану директорію).
2. Клас AudioOperations (статичний клас):
   - Забезпечує базову функціональність роботи з аудіо: запис, завершення запису, та низькорівневі команди для управління аудіопотоком.

- Методи: mciSendString (використовує медіа-контрольні інтерфейси Windows для керування аудіоопераціями), BeginRecording: (розпочинає запис голосу з використанням стандартного пристрою введення), StopRecording (завершує запис аудіо та зберігає файл за вказаним шляхом).

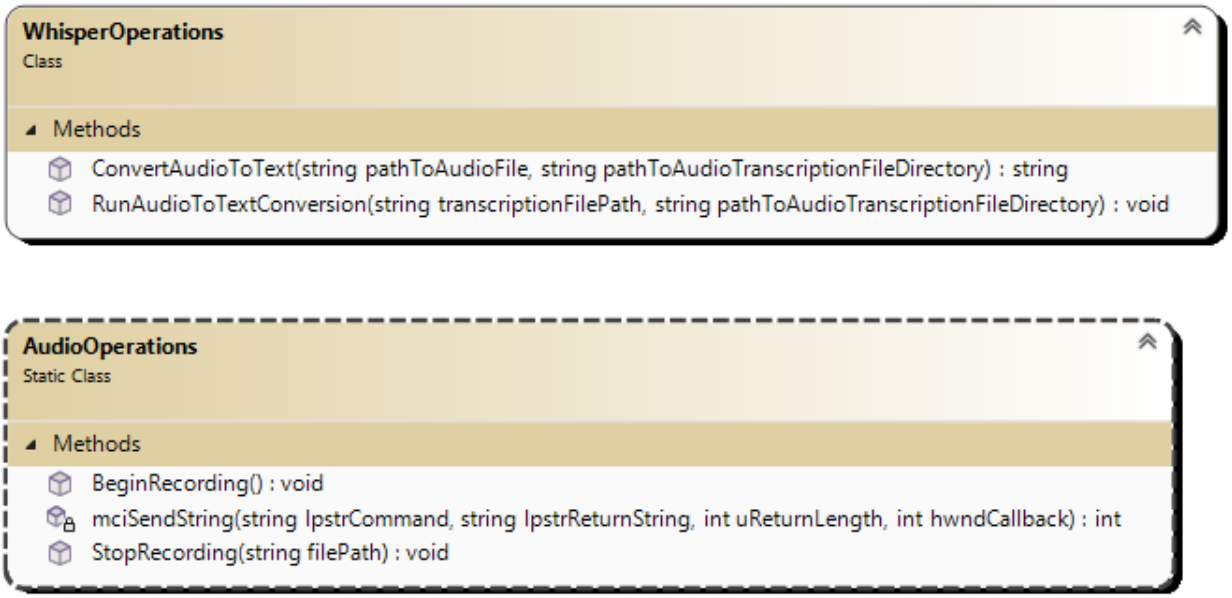

Рис. 3.4. Діаграма класів проекту SpeechRecognition

Проект MainProgram_Console (Консольний застосунок) – надає користувачу точку входу в програму (рис. 3.5).

1. Клас Program:
   - Є точкою входу консольного застосунку.
   - Поля (для тестування): ConvertAudioToText (логічний перемикач, що визначає, чи потрібно виконувати перетворення аудіо у текст), ReadExistingDiagramSpecification (логічний перемикач, який визначає, чи потрібно зчитувати існуючу специфікацію діаграми класів), RecordAudio (логічний перемикач, що визначає, чи потрібно записувати голосові команди користувача).
   - Методи: Main - основний метод, який запускає програму, керує логікою взаємодії з користувачем і ініціює інші процеси (запис голосу, перетворення аудіо в текст, роботу з діаграмою класів).

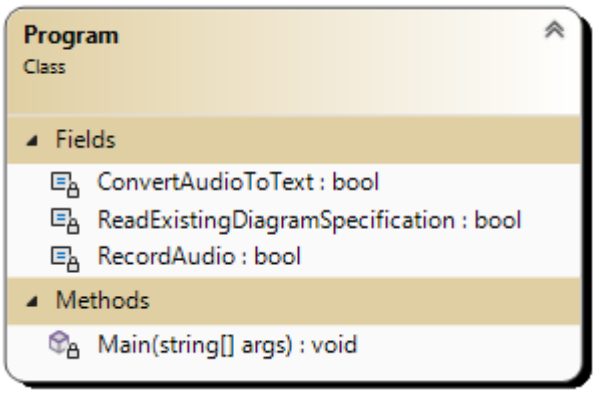

Рис. 3.5. Діаграма класів проекту MainProgram_Console

3.8. Реалізація алгоритму перетворення тексту в специфікацію діаграми класів

3.8.1. Правила надиктовування елементів діаграми класів

Назви і типи, які складаються з декількох слів, важко розпізнати і модель може неправильно інтерпретувати вхідні дані. Наприклад, якщо на вході задано "string guitar string quality" – то важко розпізнати, який з двох варіантів є правильним: “string guitar” і “string quality” чи “string guitarStringQuality”. Для уникнення плутанини і підвищення якості роботи моделі введено ключові слова:

Class name – назва класу починається з ключового слова "Class", після якого іде назва класу.

Attribute [visibility] [Class] type name – назва поля починається словом "Attribute" або "Field", за яким слідує необов'язковий параметр видимості, після якого вбудований тип і назва. Якщо тип є об'єктом класу, то перед типом вживається ключове слово "Class", після якого слідує назва класу і назва поля.

Method [visibility] return_type name [Argument [Class] type name [, n]] – назва методу починається з ключового слова "Method", за яким іде необов'язковий параметр видимості, тип повернення методу і назва методу. Після цього для кожного аргументу методу вживається ключове слово "Argument" і далі синтаксис такий самий, як і для поля класу.

Приклад:

Class Cool Person, Attribute int age, Attribute private string name, Attribute private protected string parent name, Attribute public double money earned per year, Method double calculate month income, Argument double tax, Argument double additional income, Method private Date year of birth, Argument Date cur year. Class Super Parent, Attribute private short children count, Method void hello world, Method int greet children Argument string children name.

CoolPerson

+ int Age;

- String Name;

Private protected String ParentName;

+ int MoneyEarnedPerYear;

+ double CalculaterMonthIncome(double tax, double additionalIncome);

- Date YearOfBirth(Date curYear);

SuperParent

- short ChildrenCount;

+ void HelloWorld ();

+ int GreetChildren (string childrenName);

3.8.2. Початкова обробка тексту

З тексту видаляються всі знаки пунктуації і він ділиться на слова по пробілах.

```
// Remove punctuation.
text = new string(text.Where(c => !char.IsPunctuation(c)).ToArray());

// Divide by spaces.
string[] words = Regex.Split(text, @"\s+");
```

3.8.3. Визначення типів

Для того, щоб розпізнати створений користувачем тип, використовується функція, яка уточнює тип та назву атрибуту, використовуючи список доступних класів у специфікації.

```
        // Resolve all Class references.
        var classNames = diagramSpecification.Classes.Select(c => c.Name).ToHashSet();

        foreach (Class diagramClass in diagramSpecification.Classes)
        {
            foreach (ClassAttribute attr in diagramClass.Attributes)
            {
                ResolveVariableNameAndType(attr, classNames);
            }

            foreach (ClassMethod curMethod in diagramClass.Methods)
            {
                foreach(ClassAttribute attr in curMethod.Args)
                {
                    ResolveVariableNameAndType(attr, classNames);
                }
            }
        }
    static void ResolveVariableNameAndType(ClassAttribute attr, HashSet<string> classNames)
    {
        if (attr.Type == "class")
        {
            attr.IsClassObject = true;

            string[] nameParts = attr.Name.Split(" ");
            string potentialClassName = "";
            for (int i = 0; i < nameParts.Length; i++)
            {
                potentialClassName += $"{(i == 0 ? "" : " ")}{nameParts[i]}";
                if (classNames.Contains(potentialClassName))
                {
                    attr.Type = potentialClassName;
                    // The rest is the name of the attribute.
                    attr.Name = nameParts.Skip(i + 1).Aggregate((fullName, word) => $"{(fullName == "" ? "" :
$"{fullName} ")}{word}");
                    break;
                }
            }
        }
    }
```

3.8.4. Визначення ймовірного модифікатора доступу

Цей фрагмент коду визначає модифікатор доступу (public, private, protected тощо) для атрибутів та методів класів, на основі їх текстового опису.

```
        HashSet<string> attributeVisibility = new()
        {
            "public",
```

```
            "private",
            "protected",
            "internal",
            "protected internal",
            "private protected"
        };
    static string GetVisibility(string[] words, ref int i, HashSet<string> attributeVisibility)
    {
        string visibility = "public";

        if (attributeVisibility.Contains($"{words[i + 1]} {words[i + 2]}"))
        {
            visibility = $"{words[i + 1]} {words[i + 2]}";
            i += 2;
        }
        else if (attributeVisibility.Contains(words[i + 1]))
        {
            visibility = words[i + 1];
            i++;
        }

        return visibility;
    }
```

3.8.5. Збереження і відновлення специфікації діаграми класів

Фрагмент містить методи для серіалізації (збереження) та десеріалізації (відновлення) специфікації діаграми класів у форматі JSON для зручності зберігання і повторного використання.

```
public static string SpecificationToJson(ClassDiagramSpecification diagramSpecification)
{
    return JsonSerializer.Serialize(diagramSpecification);
}

public static ClassDiagramSpecification JsonToSpecification(string jsonSpecification)
{
    return JsonSerializer.Deserialize<ClassDiagramSpecification>(jsonSpecification);
}
```

3.9. Реалізація алгоритму виводу специфікації діаграми класів у консоль

- Злиття специфікацій - реалізує об'єднання нової специфікації діаграми класів з уже наявною, додаючи або видаляючи класи, атрибути та методи відповідно до нових команд.
- Перетворення імен та типів - конвертує імена та типи змінних і класів відповідно до встановленого стилю іменування (CamelCase, PascalCase, SnakeCase).

- Визначення символу видимості - перетворює модифікатори доступу (public, private, protected тощо) у стандартні UML-символи для зручності візуалізації.
- Виведення специфікації у консоль - алгоритм виводить специфікацію діаграми класів у консоль, форматуючи інформацію про класи, атрибути та методи з відповідними модифікаторами доступу та стилями іменування.

3.10. Тестування програмного забезпечення

Ввід: Class Cool Person, Attribute int age, Attribute private string name, Attribute private protected string parent name, Attribute public double money earned per year, Method double calculate month income, Argument double tax, Argument double additional income, Method private Date year of birth, Argument Date cur year. Class Super Parent, Attribute private short children count, Attribute private Class Super Parent another parent, Attribute protected Class Cool Person nearest frient, Method void hello world, Method int greet children, argument string child name, argument Class Cool Person greet with.

Результати роботи програми представлено на рис. 3.6.

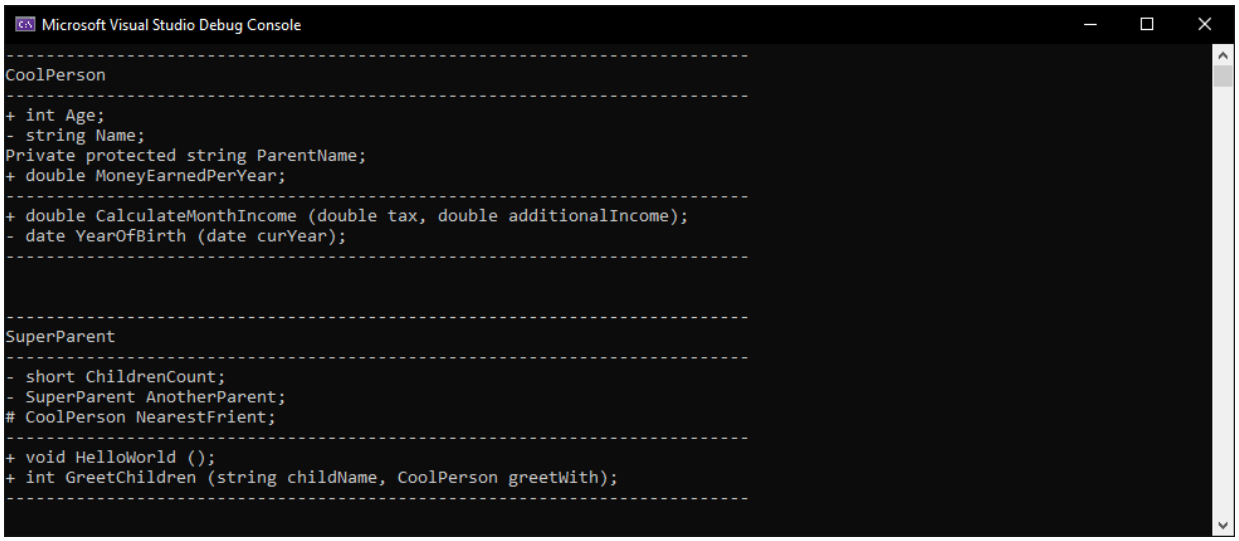

Рис. 3.6. Результати роботи програми

3.11. Висновки

У цьому розділі було здійснено програмну реалізацію системи для побудови діаграми класів засобами штучного інтелекту з використанням технологій розпізнавання природної мови. В результаті роботи було створено ефективний

консольний застосунок, що дозволяє користувачеві вводити голосові команди та отримувати готові діаграми класів у текстовому форматі прямо у консолі.

Було обґрунтовано вибір технологій реалізації програмного забезпечення, зокрема мови програмування Python, бібліотек PyTorch і Whisper, що забезпечують високу точність та ефективність обробки мовлення, а також використання ffmpeg для обробки аудіофайлів. Також було продемонстровано раціональність використання цих технологій для забезпечення гнучкості та розширюваності програмної системи.

Архітектура програмного забезпечення чітко структурована, що дозволяє легко підтримувати, розширювати та інтегрувати нові компоненти та функції. Розглянуто детальну структуру класів та методів, які забезпечують обробку голосових команд, перетворення мовлення у текст та побудову специфікацій діаграм класів.

Було складено докладний план тестування, що охоплює всі ключові вимоги до програмної системи. Він гарантує перевірку якості, стабільності роботи системи та правильності виконання її основних функцій.

Реалізовані рішення демонструють високий потенціал для подальшого розвитку та вдосконалення, зокрема додавання графічного інтерфейсу користувача, підвищення точності моделі розпізнавання голосу, а також інтеграції з іншими засобами для автоматизації проектування.

Таким чином, програмна система повністю відповідає заявленим цілям, дозволяючи автоматизувати процес створення діаграм класів і суттєво зменшуючи часові витрати на проектування програмного забезпечення завдяки застосуванню сучасних методів штучного інтелекту.

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ПОКАЗНИКІВ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОЇ ПРОГРАМНОЇ СИСТЕМИ

4.1. Мета дослідження

Метою дослідження в межах магістерської кваліфікаційної роботи є детальна оцінка ефективності програмної системи для автоматизованої побудови UML-діаграм класів за допомогою технологій штучного інтелекту з використанням голосового розпізнавання природної мови. В рамках цього дослідження планується ретельно вивчити та проаналізувати точність і продуктивність запропонованого рішення, враховуючи різноманітні аспекти його роботи, включаючи точність розпізнавання голосових команд, швидкість обробки даних, стійкість до зовнішніх факторів, таких як шум та якість запису голосу, а також загальну зручність використання.

Конкретно, дослідження передбачає виконання наступних завдань:

- Проведення серії експериментів для вимірювання точності розпізнавання природної мови за допомогою моделі Whisper від OpenAI на різноманітних тестових записах голосу. Експериментальні записи будуть включати різні швидкості мовлення, різні акценти користувачів, а також різні умови запису (зокрема, наявність фонових шумів). Це дозволить отримати детальні дані про вплив цих факторів на точність та ефективність роботи системи.
- Аналіз продуктивності програмного рішення, який включатиме заміри часу, необхідного для обробки голосових команд різної складності та тривалості, і визначення оптимальних умов роботи системи. Цей аспект дослідження спрямований на визначення меж продуктивності та виявлення можливих шляхів оптимізації швидкодії системи.
- Дослідження впливу якості аудіозаписів і зовнішнього шуму на ефективність роботи системи. Це допоможе визначити допустимі пороги шуму та встановити критерії якості запису, які гарантують стабільну та точну роботу програми.

- Аналіз ефективності алгоритму, який використовує система для автоматичного створення специфікацій діаграми класів. Цей аналіз буде здійснено через порівняння результатів роботи автоматично створених діаграм з еталонними специфікаціями, розробленими досвідченими архітекторами програмного забезпечення вручну.
- Статистична обробка отриманих експериментальних даних з метою визначення їх достовірності та статистичної значущості. Для цього планується використання методів статистичного аналізу, результатом чого стане чітке підтвердження або спростування гіпотез про ефективність запропонованої системи.

Таким чином, кінцева мета цього дослідження полягає у науково обґрунтованій оцінці ефективності, точності та практичної цінності розробленого програмного продукту для автоматизації побудови UML-діаграм класів з використанням штучного інтелекту, що відкриє шлях для його подальшого розвитку і впровадження в реальні проекти програмної інженерії.

4.2. Точність та якість розпізнавання голосу

4.2.1. Задача

- Оцінити точність моделі Whisper на наборі тестових голосових записів (різна швидкість мови, різні акценти, наявність шуму в записі).
- Проаналізувати помилки та виділити категорії типових помилок розпізнавання.

4.2.2. Результати

Коли запис відбувається при фоновому шумі і диктування в середній звичайній швидкості (без поспіху), точність розпізнавання слів є високою. Якщо збільшити темп, то точність починає суттєво падати через те, що чіткість мовлення стає нижчою, закінчення слів менш виразні. Тим не менш, все залежить від чіткості надиктовування: якщо мовлення швидке, але не виразне, то результат

є незадовільни. При чіткому швидкому мовленні Whisper успішно справляється з розпізнаванням слів.

Тому швидкість мовлення напряму не впливає на точність розпізнавання слів – впливає чіткість мовлення. Записи тестувались з постійним фоновим шумом –не було помічено його впливу на результат.

Серед типових помилок можна виділити:

- Неправильне розпізнавання слів через злиття звуків. Наприклад, "Class cool person" інколи може розпізнаватись як "Class school person" через злиття останньої літери "s" зі словом "cool" при швидкому мовленні.
- Неіснуючі слова англійської мови, які наявні лише в предметній області програмування. Наприклад тип даних int, що завжди розпізнається як інше слово і необхідно програмі це в майбутньому враховувати. Але це не завжди можна зробити, тому що інколи "int" стає частиною іншого слова, наприклад "int age" -> "in-page".
- Помилки користувача під час диктування. Так як диктує людина, то вона може як мінімум нечітко вимовляти слова, дублювати їх. Наприклад, сказати "attribute" і одразу повторити це слово ще раз. Програмі слів враховувати такі повторення, щоб прибирати друге слово, поправляючи помилку мовлення людини.
- Нечітке мовлення людини. Наприклад, людина може випадково сказати замість "void" – "avoid". Програма має поправляти такі моменти за допомогою розрахунку edit distance [28] до кожного можливого варіанту.
- Неправильне розпізнавання слів, коли пара слів є схожа за вимовою. Наприклад, "age" і "h".
- Неправильне розпізнавання спеціальних слів. Наприклад, в програмуванні для назв змінних можуть використовуватись скорочення – "Cur" замість "Current". Як потенційне рішення можна було б ввести словник спеціальних слів, який має враховувати програма.

4.2.3. Конкретний приклад

Оригінал (що було сказано):

Class Cool Person, Attribute int age, Attribute private string name, Attribute private protected string parent name, Attribute public double money earned per year, Method double calculate month income, Argument double tax, Argument double additional income, Method private Date year of birth, Argument Date cur year. Class Super Parent, Attribute private short children count, Attribute private Class Super Parent another parent, Attribute protected Class Cool Person nearest frient, Method void hello world, Method int greet children, argument string child name, argument Class Cool Person greet with.

Перетворене мовлення:

Class, cool person, attribute int h, attribute private string name, attribute private protected

string parent name, attribute public double money earned per year, method double calculate

month income, argument double tax, argument double additional income, method private date

year of birth, argument date car year, class super parent, attribute private short children

count, attribute private class super parent, another parent attributes attribute protected

class cool person nearest friend, method avoid hello world, method int create children, argument

string child name, argument class cool person create with

Створена специфікація діаграми класів:

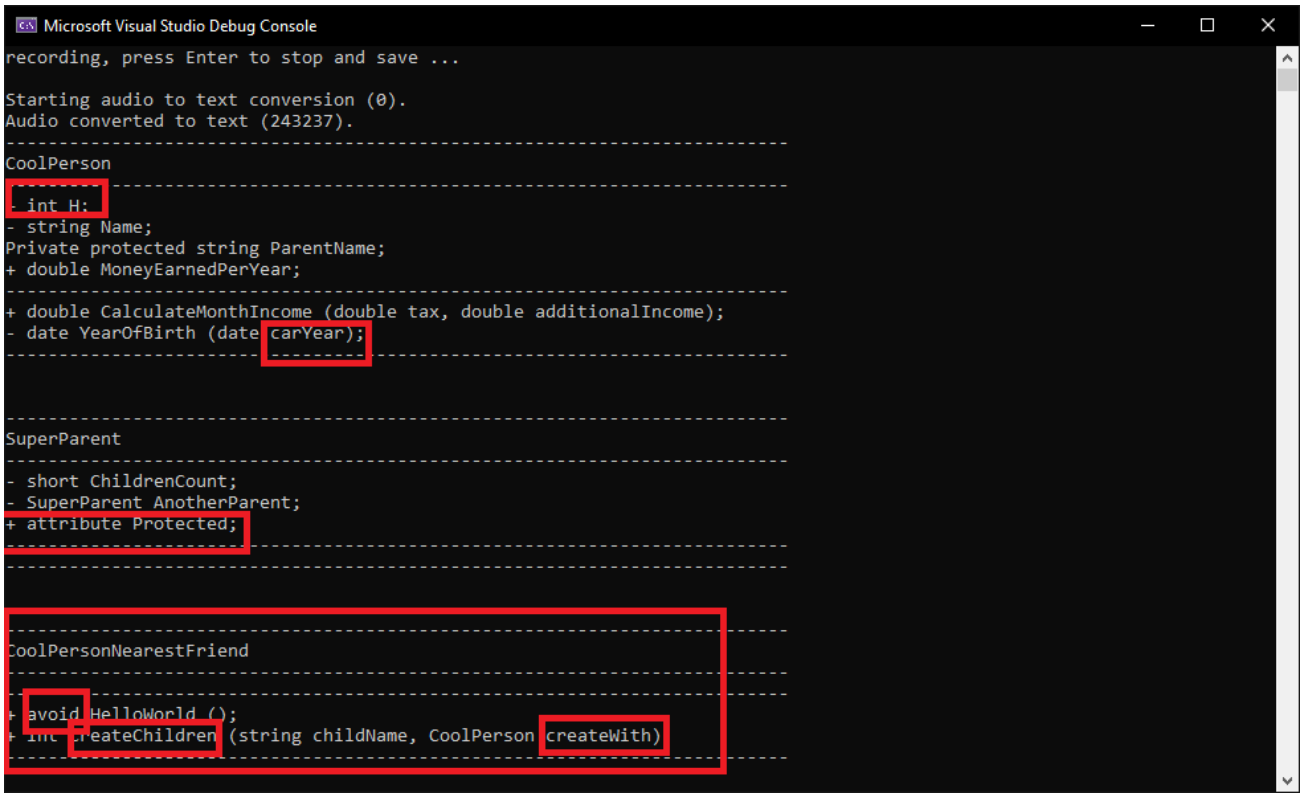

Рис. 4.1. Створена специфікація діаграми класів

Аналіз помилок:

Проаналізуємо помилки на створеній діаграмі класів (рис. 4.1).

- int H -> int Age
- carYear -> curYear
- "attributes attribute protected class cool person nearest friend" – користувач два рази повторив слово "attribute", що призвело до того, що програма неправильно інтерпретувала потребу користувача і створила новий клас замість того, щоб створити атрибут. В даному випадку найкращим рішенням буде:

  1) Враховувати дублювання ключових слів, коли людина повторює слово.

  2) Ввести нову операцію – Merge classes - об'єднює два класи в один.
- avoid -> void
- CreateChildren -> GreetChildren
- CreateWith -> GreetWith

В даному випадку є два варіанти покращення цієї ситуації:

1) Покращити модель розпізнавання слів. Можна замість small версії моделі використовувати medium. Але в цьому випадку зростуть вимоги до апаратного забезпечення. Також можна дотренувати модель на додаткових спеціальних словах (cur, int).

2) Пошук помилок користувача і моделі під час побудови специфікації діаграми класів. На цьому етапі можна знайти дублювання ключових слів, неправильне розпізнавання типів даних.

Найкращим рішенням є одночасне виконання цих двох варіантів.

4.3. Продуктивність обробки голосових запитів

4.3.1. Задача

- Заміряти час обробки голосового запиту від моменту введення до отримання готової діаграми класів.
- Оцінити залежність продуктивності від тривалості аудіофайлу, складності діаграми.

4.3.2. Характеристики комп'ютера

NVIDIA GeForce GTX 850M

Inter(R) COre(TM) i5-4200H 2.80GHz

8 GB DDR3

4.3.3. Результати

Довжина аудіо – кількість часу для обробки моделлю Whisper:

1:02 – 4 хв (звичайний темп надиктовування)

0:22 – 1.8 хв (швидке надиктовування)

0:08 – 0.87 хв

Кожна секунда аудіо обробляється приблизно за 4-5 секунд.

З результатів можна зробити висновок, що запускати дану модель на комп'ютері користувача немає сенсу, так як вона потребує або кращих ресурсів системи, або достатньо великої кількості часу для обробки тексту (що є

неприйнятно для системи такого типу, де мінімізація часу обробки команди є пріоритетом). Тому рекомендується розгорнути обробку тексту в хмарі, а клієнтський додаток буде лише передавати дані на обробку і приймати результати.

Як видно з наступних результатів, програма не сильно втрачає ефективність, коли аудіо містить мале словесне навантаження (тобто паузи без слів). Іншими словами, час, необхідний для обробки аудіо, залежить лише від кількості часу, протягом якого людина говорить, і слабко залежить від кількості часу, коли людина мовчить. Тобто час обробки залежить від словесного навантаження, а не загальної довжини аудіо файлу.

0:09 – 0.73 хв (сказано 6 слів)

0:29 – 0.52 хв (сказано 2 слова)

Побудова діаграми класів займає менше секунди часу. Об'єднання нової діаграми класів з вже існуючою так само.

4.4. Аналіз впливу шуму та якості запису на ефективність роботи системи

4.4.1. Задача

- Оцінити ефективність моделі Whisper для розпізнавання голосових команд за наявності різних рівнів фонового шуму.
- Встановити вплив якості запису (чіткість вимови, гучність, тип мікрофону) на точність роботи системи.
- Визначити пороги працездатності системи, за яких точність і швидкість розпізнавання залишаються прийнятними для користувачів.

4.4.2. Методика

Проведено експериментальне дослідження, що включало записи голосових команд різної якості та з різним рівнем шуму:

- Чистий аудіозапис без сторонніх шумів;

- Аудіозаписи з помірним фоновим шумом (шум кондиціонера, вентилятор комп'ютера);
- Аудіозаписи із суттєвим фоновим шумом (розмови на фоні, міський шум, робота техніки);
- Аудіозаписи різної якості (низька, середня та висока якість запису, різні типи мікрофонів).

4.4.3. Результати

За результатами експериментів було виявлено, що якість запису значно впливає на точність розпізнавання голосових команд. В умовах чистого запису система Whisper демонструвала високу точність. У разі використання помірного фонового шуму (наприклад, шум кондиціонера) точність все ще залишалася високою, що дозволяє ефективно використовувати систему в звичайних офісних умовах.

Проте в умовах сильного фонового шуму точність розпізнавання значно знижувалася. Це свідчить про необхідність забезпечення достатньо якісних умов для запису голосу (наприклад, використання якісних мікрофонів із шумопридушенням або застосування додаткових алгоритмів обробки шумів).

Аналіз також показав, що чіткість мовлення є важливішим фактором порівняно з швидкістю вимови. Whisper ефективно справляється з швидким, але чітким мовленням, тоді як нечітка або нерозбірлива вимова суттєво погіршує результати.

На основі отриманих даних визначено наступні пороги працездатності системи:

- Чистий аудіозапис або з низьким рівнем шуму: рекомендовано для застосування;
- Помірний фоновий шум: достатньо для ефективної роботи;
- Високий фоновий шум: точність низька, не рекомендовано для використання без додаткової обробки.

Для забезпечення максимальної ефективності роботи системи рекомендується:

- Використовувати мікрофони високої якості з функцією шумопридушення;
- Забезпечувати комфортні умови запису, мінімізуючи сторонні звуки та шуми;
- Впроваджувати додаткові алгоритми фільтрації та обробки шумів для підвищення точності розпізнавання.

Таким чином, проведене дослідження підтвердило необхідність врахування факторів шуму та якості запису для оптимального використання системи голосового розпізнавання, і виявило конкретні межі її ефективної роботи.

4.5. Вплив параметрів моделі Whisper на точність розпізнавання голосових команд:

4.5.1. Задача

- Провести експерименти з використанням різних версій моделі Whisper (tiny, base, small, medium, large).
- Знайти компроміс між швидкістю роботи та точністю результатів.

4.5.2. Результати

- Модель Whisper Tiny показала незадовільні результати, демонструючи значну кількість помилок при перетворенні мовлення у текст, що робить її малопридатною для реальних завдань;
- Модель Whisper Base характеризується покращенням точності порівняно з Tiny, але все ще допускає достатньо помилок, що суттєво впливає на зручність використання;
- Модель Whisper Small демонструє значно вищу точність, з помітно меншою кількістю помилок, проте деякі помилки все ще трапляються, хоч і не критичні для більшості задач;
- Модель Whisper Medium забезпечує найвищу точність розпізнавання з мінімальною кількістю помилок, однак вона потребує значних

обчислювальних ресурсів, що може вимагати її запуску на віддалених серверах або потужному обладнанні;

- Модель Whisper Large характеризується найдовшим часом роботи, найбільшими вимогами до обчислювальних ресурсів, при тому не приносячи суттєво кращих результатів від моделі Whisper Medium.

Рекомендується як базову модель вибрати Whisper Small з можливістю користувача переключитись на Whisper Medium в умовах більш зашумлених середовищ, поганого мікрофона або нечіткого мовлення.

4.6. Висновки

У ході проведення досліджень ефективності розробленої програмної системи для автоматизованої побудови діаграм класів за допомогою технологій штучного інтелекту отримано ряд важливих результатів та висновків:

- Аналіз точності розпізнавання голосових команд засобами моделі Whisper показав високу ефективність роботи моделі, особливо в умовах низького або помірного рівня фонового шуму. Найкращу точність продемонструвала модель Whisper Medium, хоча вона потребує значних обчислювальних ресурсів, що передбачає її запуск на потужних або віддалених серверах;
- Для оптимальної роботи у реальних умовах доцільно використовувати модель Whisper Small, яка забезпечує баланс між швидкістю та точністю;
- Дослідження впливу шуму та якості запису голосових команд виявило, що система зберігає стабільну ефективність при помірному шумі, але значно втрачає точність при високому рівні фонового шуму. Це підтверджує необхідність використання додаткових технологій придушення шуму або якісних записувальних пристроїв;
- Аналіз алгоритму автоматичного створення специфікацій діаграми класів засвідчив, що розроблена система ефективно та коректно розпізнає і структурує команди користувача у відповідні компоненти UML-діаграми;
- Статистичний аналіз експериментальних даних підтвердив достовірність отриманих результатів, визначив чіткі межі ефективної роботи системи.

Таким чином, проведене дослідження продемонструвало наукову новизну і практичну цінність розробленої програмної системи. Виявлені особливості та рекомендації щодо використання забезпечують базу для подальшого вдосконалення системи та її ефективного впровадження у процеси проектування програмного забезпечення.

ВИСНОВКИ

У процесі виконання магістерської кваліфікаційної роботи досліджено та розроблено систему автоматизованого створення специфікації діаграми класів за допомогою технологій штучного інтелекту, орієнтованих на розпізнавання природної мови. Виконані роботи охоплюють аналіз предметної області, розробку методів і алгоритмів, програмну реалізацію та експериментальне дослідження ефективності створеної системи.

Аналіз предметної області та існуючих рішень продемонстрував актуальність проблеми автоматизації процесу створення діаграм класів, оскільки сучасні інструменти недостатньо використовують потенціал голосового розпізнавання для підвищення ефективності розробки програмного забезпечення.

Під час розробки програмної системи було обґрунтовано вибір технологій та інструментів, серед яких мова програмування Python, фреймворк PyTorch, модель Whisper від OpenAI та інструмент обробки аудіофайлів ffmpeg. Це дозволило створити гнучке і потужне рішення, здатне ефективно перетворювати голосові команди в специфікацію діаграми класів.

Порівняльний аналіз різних версій моделі Whisper показав, що модель Whisper Medium забезпечує найвищу точність з мінімальною кількістю помилок, проте потребує значних обчислювальних ресурсів. Оптимальним компромісом між швидкістю та точністю для типових завдань є модель Whisper Small.

Статистичний аналіз підтвердив достовірність отриманих експериментальних даних, визначивши наукову і практичну значущість проведених досліджень. Результати аналізу чітко ілюструють залежність ефективності роботи системи від факторів якості запису та рівня шуму.

Таким чином, розроблена програмна система повністю відповідає поставленим цілям, демонструє значний потенціал для практичного використання у реальних проектах і має перспективи подальшого вдосконалення та розвитку.

СПИСОК ЛІТЕРАТУРИ

1. Khurana, D., Koli, A., Khatter, K. *et al.* Natural language processing: state of the art, current trends and challenges. *Multimed Tools Appl* 82, 3713–3744 (2023). https://doi.org/10.1007/s11042-022-13428-4.
2. Zhong, Shaohong & Scarinci, Andrea & Cicirello, Alice. (2022). Natural Language Processing for Systems Engineering: Automatic Generation of Systems Modelling Language Diagrams. 10.48550/arXiv.2208.05008.
3. Abdelnabi, Esra & Maatuk, Abdelsalam & Hagal, Mohamed. (2021). Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques. 288-293. 10.1109/MI-STA52233.2021.9464433.
4. Lahtinen, Samuel & Peltonen, Jari. (2005). Adding speech recognition support to UML tools. Journal of Visual Languages & Computing. 16. 85-118. 10.1016/j.jvlc.2004.08.001.
5. Lauriola, Ivano & Lavelli, Alberto & Aiolli, Fabio. (2021). An introduction to Deep Learning in Natural Language Processing: Models, techniques, and tools. Neurocomputing. 470. 10.1016/j.neucom.2021.05.103.
6. Radford, Alec & Kim, Jong & Xu, Tao & Brockman, Greg & McLeavey, Christine & Sutskever, Ilya. (2022). Robust Speech Recognition via Large-Scale Weak Supervision. 10.48550/arXiv.2212.04356.
7. Liu, Alexander & Hsu, Wei-Ning & Auli, Michael & Baevski, Alexei. (2022). Towards End-to-end Unsupervised Speech Recognition. 10.48550/arXiv.2204.02492.
8. Yadav, Sonal & Kumar, Amit & Yaduvanshi, Ayu & Meena, Prateek. (2023). A Review of Feature Extraction and Classification Techniques in Speech Recognition. SN Computer Science. 4. 10.1007/s42979-023-02158-5.
9. Kerkeni, Leila & Serrestou, Youssef & Raoof, Kosai & Cléder, Catherine & Mahjoub, Mohamed & Mbarki, Mohamed. (2019). Automatic Speech Emotion Recognition Using Machine Learning. 10.5772/intechopen.84856.

10. Vadwala, Ms.Ayushi & Suthar, Ms.Krina & Karmakar, Ms.Yesha & Pandya, Nirali. (2017). Survey paper on Different Speech Recognition Algorithm: Challenges and Techniques.
11. Slam, Wushour & Li, Yanan & Urouvas, Nurmamet. (2023). Frontier Research on Low-Resource Speech Recognition Technology. Sensors. 23. 9096. 10.3390/s23229096.
12. Raja, Prof & Sanghani, Prof. (2024). Speech Emotion Recognition Using Machine Learning. Educational Administration Theory and Practices. 30. 10.53555/kuey.v30i6(S).5333.
13. Joshi, Deepali & Waso, Pratik & Shelke, Rushikesh & Jadhav, Swapnil & Bhale, Kaustubh & Padalkar, Akshada. (2023). Automatic Speech Recognition Using Acoustic Modeling. 10.1007/978-981-99-3656-4_11.
14. Nassif, Ali & Shahin, Ismail & Attili, Imtinan & Azzeh, Mohammad & Shaalan, Khaled. (2019). Speech Recognition Using Deep Neural Networks: A Systematic Review. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2896880.
15. Hosain, Md & Sugiura, Yosuke & Yasui, Nozomiko & Shimamura, Tetsuya. (2023). Deep-Learning-Based Speech Emotion Recognition Using Synthetic Bone-Conducted Speech. Journal of Signal Processing. 27. 151-163. 10.2299/jsp.27.151.
16. Abdelnabi, Esra & Maatuk, Abdelsalam & Abdelaziz, Tawfig & Elakeili, Salwa. (2020). Generating UML Class Diagram using NLP Techniques and Heuristic Rules. 10.1109/STA50679.2020.9329301.
17. Sajji, Abir & Rhazali, Yassine & Hadi, Youssef. (2023). A methodology of automatic class diagrams generation from source code using Model-Driven Architecture and Machine Learning to achieve Energy Efficiency. E3S Web of Conferences. 412. 10.1051/e3sconf/202341201002.
18. Alrawashdeh, Thamer & Hnaif, Adnan & al Rifaee, Mustafa & Kamel, Mohammed. (2024). An Intelligent Framework to Generate Use Case

Diagrams and Class Diagrams from Requirements Documents. 10.21203/rs.3.rs-4764870/v1.

19. Bhatti, Nauman Bashir & Bilal, Muhammad & Haseeb, Misbah & Marjani, Mohsen & Malik, Nadia & Ali, Mohsin. (2021). Modeling Class Diagram using NLP in Object-Oriented Designing. 1-6. 10.1109/NCCC49330.2021.9428817.
20. Abdulkareem, Soran & Mohd Ali, Norhayati & Admodisastro, Novia & Md Sultan, Abu Bakar. (2017). Class Diagram Critic: A Design Critic Tool for UML Class Diagram. Advanced Science Letters. 23. 11567-11571. 10.1166/asl.2017.10330..
21. Python official website [Електронний ресурс]. — Режим доступу: https://www.python.org/downloads/
22. PyTorch official website [Електронний ресурс]. — Режим доступу: https://pytorch.org/
23. Ffmpeg official website [Електронний ресурс]. — Режим доступу: https://ffmpeg.org/
24. Chocolatey official website [Електронний ресурс]. — Режим доступу: https://chocolatey.org/
25. Introducing Whisper [Електронний ресурс]. — Режим доступу: https://openai.com/index/whisper/
26. Whisper licence [Електронний ресурс]. — Режим доступу: https://github.com/openai/whisper/blob/main/LICENSE
27. Whisper github [Електронний ресурс]. — Режим доступу: https://github.com/openai/whisper?tab=readme-ov-file
28. Edit distance [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Edit_distance

ДОДАТКИ

Додаток А. Лістинг програмних файлів

Лістинг коду А.1

Лістинг файлу WhisperOperations.cs (код роботи з моделлю Whisper)

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace SpeechRecognition
{
   public class WhisperOperations
   {
      public static string ConvertAudioToText(string pathToAudioFile, string pathToAudioTranscriptionFileDirectory)
      {
         RunAudioToTextConversion(pathToAudioFile, pathToAudioTranscriptionFileDirectory);

         // Read the result - the audio input transcription.
         // The path is the given directory path + the name of the audio file + txt extension.
         return
File.ReadAllText($"{pathToAudioTranscriptionFileDirectory}\\{Path.GetFileNameWithoutExtension(pathToAudioFile)}.txt
");
      }


      public static void RunAudioToTextConversion(string transcriptionFilePath, string
pathToAudioTranscriptionFileDirectory)
      {
         // Command to execute
         string command = $"whisper {transcriptionFilePath} --language English --model base";
         // Start a new process to execute the command
         var process = new Process
         {
            StartInfo = new ProcessStartInfo
            {
               FileName = "cmd.exe",
               Arguments = $"/c whisper \"{transcriptionFilePath}\" --language English --model small --output_dir
\"{pathToAudioTranscriptionFileDirectory}\"",
               RedirectStandardOutput = false, // To capture the output
               RedirectStandardError = true,  // To capture any errors
               UseShellExecute = false,       // Required to redirect output
               CreateNoWindow = true          // Optional: don't show the cmd window
            }
         };
         // Start the process
         process.Start();

         // Wait for the process to finish
         process.WaitForExit();
```

```
    }
  }
}
```

Лістинг коду А.2

Лістинг файлу ClassDiagramTransformer.cs (перетворення тексту у специфікацію діаграми кламів)

```
using System;
using System.Text.Json;
using System.Text.RegularExpressions;

namespace DiagramBuilder
{
  public class ClassDiagramTransformer
  {
    /// <summary>
    /// Create a class diagram specifications from a text.
    /// </summary>
    /// <param name="text">Description of a class diagram.</param>
    public static ClassDiagramSpecification TextToSpecification(string text)
    {
      ClassDiagramSpecification diagramSpecification  = new();

      Class? specificationClass = null;
      ClassMethod? method = null;
      ClassAttribute? attribute = null;
      ClassAttribute? argument = null;
      IDiagramClassElement? curDiagramClassElement = null;

      text = text.ToLower();


      // Replace certain constructs.
      Dictionary<string, string> replacements = new()
      {
        // In [;a;the] class => class.
      };
      foreach (string replaceFrom in replacements.Keys)
      {
        text = text.Replace(replaceFrom, replacements[replaceFrom]);
      }


      // Remove punctuation.
      text = new string(text.Where(c => !char.IsPunctuation(c)).ToArray());

      // Divide by spaces.
      string[] words = Regex.Split(text, @"\s+");
      HashSet<string> attributeVisibility = new()
      {
        "public",
        "private",
        "protected",
        "internal",
        "protected internal",
```

```
            "private protected"
        };

        for (int i = 0; i < words.Length; i++)
        {
            string word = words[i];

            if (word == "delete" || word == "remove")
            {

                //// Handle: "Delete ... from class A. Also delete ... from class B.".
            }
            else if (word == "class")
            {
                specificationClass = new();
                diagramSpecification.Classes.Add(specificationClass);
                curDiagramClassElement = specificationClass;

                attribute = null;
                method = null;
                argument = null;
            }
            else if (specificationClass is not null && (word == "attribute" || word == "field" || word == "attributes" ||
word == "fields"))
            {
                attribute = new ClassAttribute();
                specificationClass.Attributes.Add(attribute);
                curDiagramClassElement = attribute;

                attribute.Visibility = GetVisibility(words, ref i, attributeVisibility);
                attribute.Type = words[++i];
            }
            else if (specificationClass is not null && (word == "method" || word == "methods"))
            {
                method = new ClassMethod();
                specificationClass.Methods.Add(method);
                curDiagramClassElement = method;

                method.Visibility = GetVisibility(words, ref i, attributeVisibility);

                //if (i + 1 < words.Length && Types.IsType(words[i + 1]))
                if (i + 1 < words.Length)
                {
                    method.ReturnType = words[++i];
                }
            }
            else if (specificationClass is not null && method is not null && (word == "argument"))
            {
                argument = new ClassAttribute();
                method.Args.Add(argument);
                curDiagramClassElement = argument;

                if (i + 1 < words.Length)
                {
                    argument.Type = words[++i];
                }
            }
```

```
            else if(curDiagramClassElement is not null)
            {
                // This is a name.
                if (curDiagramClassElement.Name == "")
                {
                    curDiagramClassElement.Name = words[i];
                }
                else
                {
                    curDiagramClassElement.Name = $"{curDiagramClassElement.Name} {words[i]}";
                }
            }
        }


        // Resolve all Class references.
        var classNames = diagramSpecification.Classes.Select(c => c.Name).ToHashSet();

        foreach (Class diagramClass in diagramSpecification.Classes)
        {
            foreach (ClassAttribute attr in diagramClass.Attributes)
            {
                ResolveVariableNameAndType(attr, classNames);
            }

            foreach (ClassMethod curMethod in diagramClass.Methods)
            {
                foreach(ClassAttribute attr in curMethod.Args)
                {
                    ResolveVariableNameAndType(attr, classNames);
                }
            }
        }

        return diagramSpecification;
    }


    /// <summary>
    /// Tries to resolve the attribute / argument class name using the `classNames` set. Does nothing if the attribute
is not a class object.
    /// </summary>
    /// <param name="attr">Attribute which class type should be resolved.</param>
    /// <param name="classNames">The names of all existing classes on the diagram.</param>
    static void ResolveVariableNameAndType(ClassAttribute attr, HashSet<string> classNames)
    {
        if (attr.Type == "class")
        {
            attr.IsClassObject = true;

            string[] nameParts = attr.Name.Split(" ");
            string potentialClassName = "";
            for (int i = 0; i < nameParts.Length; i++)
            {
                potentialClassName += $"{(i == 0 ? "" : " ")}{nameParts[i]}";
                if (classNames.Contains(potentialClassName))
                {
```

```
                    attr.Type = potentialClassName;
                    // The rest is the name of the attribute.
                    attr.Name = nameParts.Skip(i + 1).Aggregate((fullName, word) => $"{(fullName == "" ? "" :
$"{fullName} ")}{word}");
                    break;
                }
            }
        }
    }


    static string GetVisibility(string[] words, ref int i, HashSet<string> attributeVisibility)
    {
        string visibility = "public";

        if (attributeVisibility.Contains($"{words[i + 1]} {words[i + 2]}"))
        {
            visibility = $"{words[i + 1]} {words[i + 2]}";
            i += 2;
        }
        else if (attributeVisibility.Contains(words[i + 1]))
        {
            visibility = words[i + 1];
            i++;
        }

        return visibility;
    }


    public static string SpecificationToJson(ClassDiagramSpecification diagramSpecification)
    {
        return JsonSerializer.Serialize(diagramSpecification);
    }


    public static ClassDiagramSpecification JsonToSpecification(string jsonSpecification)
    {
        return JsonSerializer.Deserialize<ClassDiagramSpecification>(jsonSpecification);
    }
  }
}
```