МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

КРИВКО Ростислав Васильович

Алгоритми синтезу програмного коду на основі розпізнававння природної мови / Software code synthesis algorithms based on natural language recognition

"спеціальність: 123 - Комп’ютерна інженерія
освітньо-професійна програма - Комп’ютерна інженерія"
Кваліфікаційна робота

Виконав студент групи КІм-21
Р. В. Кривко
_____

Науковий керівник:
к.т.н., Ю. М. Батько
_____

Кваліфікаційну роботу допущено до захисту:

"""___"" _____ 20___ р."

Завідувач кафедри
_____**Л. О. Дубчак**

ТЕРНОПІЛЬ – 2023

РЕЗЮМЕ

Кваліфікаційна робота на тему «Алгоритми синтезу програмного коду на основі розпізнавання природної мови» зі спеціальності 123 «Комп'ютерна інженерія» освітньо-професійна програма «Комп'ютерна інженерія» написана обсягом в 81 сторінки і містить 18 ілюстрацій, 2 таблиці. 3 додатки та 52 використаних джерел.

Метою кваліфікаційної роботи є розробка алгоритму програмного коду на основі розпізнавання природної мови.

Наукова новизна. Наукова новизна дослідження полягає в розробці інтегрованої системи, яка з'єднує технології розпізнавання природної мови з механізмами автоматичного генерування програмного коду. Це дозволяє користувачам перетворювати мовні запити на синтаксично правильний програмний код. відкриваючи нові можливості для інтерактивної розробки програмного забезпечення.

Результати дослідження: отримані результати в магістерській роботі відіграють значну роль у покрашенні процесу розробки програмного забезпечення, пропонуючи ефективний інструмент для автоматизації генерації коду. Ця інновація спрощує прототипізацію та розробку, зокрема для рутинних завдань, тим самим підвищуючи продуктивність розробників. Водночас, система відкриває двері для не-програмістів до світу програмування, дозволяючи їм створювані програми за допомогою простих мовних команд.

Ключові слова: ОБРОБКА ПРИРОДНОЇ МОВИ, АЛГОРИТМ РОЗПІЗНАВАННЯ ЗВУКОВИХ СИГНАЛІВ, PYTHON, ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ, РОЗПІЗНАВАННЯ ТА ОБРОБКИ ПРИРОДНОЇ МОВИ, HELPEK.

RESUME

The qualification work on the topic "Algorithms for program code synthesis based on natural language recognition", "Master" from the specialty 123 "Computer Engineerin" of the educational program "Computer Engineerin'' is written in the volume of 81 pages and contains 18 illustrations, 2 tables, 3 appendices and 52 used sources.

The primary objective of this qualification work is to devise an algorithm for program code generation using natural language recognition.

Scientific novelty. The scientific novelty of the research lies in the development of an integrated system that combines natural language recognition technologies with mechanisms for automatic code generation. This integration allows users to transform linguistic queries into syntactically correct program code, unlocking new possibilities for interactive software development.

Research results: the obtained results in this master's thesis play a significant role in enhancing the software development process by providing an efficient tool for code generation. This innovation simplifies prototyping and development, particularly for routine tasks, thereby increasing developers' productivity. Simultaneously, the system opens doors for non-programmers to enter the programming world, enabling them to create programs using simple language commands.

Keywords: NATURAL LANGUAGE PROCESSING, SOUND SIGNAL RECOGNITION ALGORITHM, PYTHON, CODE GENERATION, NATURAL LANGUAGE RECOGNITION AND PROCESSING, HELPEK.

ЗМІСТ

ВСТУП

Сучасний світ переживає технологічний ренесанс, і в центрі цього ренесансу знаходиться програмне забезпечення. Щодня мільйони розробників по всьому світу створюють нові програми, що впливають на кожен аспект нашого життя — від повсякденних додатків до складних систем управління. У такому контексті здатність швидко і ефективно перетворювати ідеї в програмний код є ключовим фактором успіху.

Сьогодні розпізнавання природної мови не обмежується тільки перекладом текстів або голосовими помічниками. Це стає ключовою технологією в медицині для аналізу клінічних даних, в юридичній сфері для аналізу документів, а також в освіті для створення персоналізованих навчальних планів.

Однією з головних проблем у програмуванні є пошук шляхів оптимізації та автоматизації процесу розробки. Уявіть, якби програмісти могли описувати задачі звичайною мовою, а система сама генерувала відповідний програмний код. Відповідь на цю потребу лежить у області обробки природної мови (NLP). Із зростанням потужності обчислення та прогресом в галузі штучного інтелекту можливість створення систем, які можуть розпізнавати та аналізувати природну мову для генерації програмного коду, стає все більш реалістичною.

Поглиблення в розпізнаванні природних мов відкриває нові можливості для гуманітарних наук, де аналіз мови виявляє непомітні тенденції та зв'язки в літературних, історичних та соціокультурних текстах. Технології, засновані на розпізнаванні природних мов, можуть служити інструментами для соціологів, психологів та лінгвістів, дозволяючи здійснювати кількісний аналіз великих наборів даних.

Також, ці технології активно інтегруються в бізнес-сферу. Компанії використовують алгоритми розпізнавання природних мов для аналізу відгуків клієнтів, що допомагає поліпшити якість обслуговування та розробити продукцію, яка відповідає потребам ринку.

У сфері економіки та фінансів системи, здатні аналізувати природну мову, можуть бути використані для прогнозування ринкових трендів на основі новинних повідомлень та фінансових звітів. З іншого боку, у сфері екології такі системи можуть допомогти аналізувати великі обсяги даних з досліджень, спрямованих на вивчення кліматичних змін, екосистем та біорізноманіття.

Природна мова має великий потенціал у сфері освіти. Сучасні студенти та викладачі можуть скористатися засобами розпізнавання природних мов для автоматизації процесів вивчення, підбору матеріалів, а також для створення інтерактивних додатків, що адаптуються до індивідуальних потреб кожного користувача.

Інтеграція розпізнавання природних мов у сфері здоров'я може революціонізувати медичне обслуговування, дозволяючи лікарям швидше та точніше аналізувати анамнез пацієнта, а пацієнтам - краще розуміти рекомендації свого лікаря.

Наукові дослідження в області розпізнавання природних мов привертають увагу урядів та громадських організацій. Вони використовують ці технології для моніторингу соціальних мереж та інших платформ з метою виявлення інформації про громадську думку, екстремістські настрої або можливі загрози безпеці.

Враховуючи широкий спектр застосувань і потенціал для подальших досліджень, вивчення алгоритмів програмного коду на основі розпізнавання природної мови стає невід'ємною частиною сучасного наукового та технологічного прогресу.

Специфіка NLP полягає у високому ступені складності та непередбачуваності людської мови. Але саме це стимулює науковців і інженерів штучного інтелекту до пошуку нових, все більш вдосконалених методів аналізу та інтерпретації природної мови, що веде до створення все більш інтелектуальних, адаптивних та інтуїтивно зрозумілих систем.

Однією з найбільш поширених технік є використання векторних представлень слів, таких як Word2Vec або GloVe. Ці техніки перетворюють слова в

вектори у багатовимірному просторі так, що слова зі схожими значеннями мають близькі векторні представлення.

Більш сучасний підхід полягає у використанні моделей глибокого навчання, таких як трансформатори. Це дозволяє вивчати контекст і відносини між словами в тексті, адже вони вміють аналізувати послідовності слів, враховуючи їхній взаємний контекст. Моделі, такі як BERT, GPT і їх варіації, стали золотим стандартом у сфері обробки природної мови завдяки їх високій ефективності та універсальності.

Крім того, дедалі актуальнішим стає застосування таких технік, як нейронні мережі та рекурентні нейронні мережі, особливо для задач, пов'язаних з обробкою послідовностей тексту, як-то переклад, аналіз настроїв чи розпізнавання голосу.

З ростом обсягів даних, які щоденно з'являються в інтернеті у вигляді відгуків, статей, блогів та інших текстових матеріалів, з'являється величезний потенціал для аналізу та використання цієї інформації. Щоб оптимізувати цей процес, дослідники та розробники звертають увагу на можливості мови програмування Python, яка вже довела свою ефективність у роботі з великими обсягами даних та машинним навчанням.

Python завдяки своїй гнучкості та великій кількості спеціалізованих бібліотек для обробки природної мови, таких як NLTK, SpaCy та інші, став золотим стандартом у галузі NLP. Він не тільки пропонує інструменти для аналізу та обробки тексту, але й підтримує інтеграцію з іншими системами машинного навчання, що дозволяє створювати все більш складні та ефективні моделі розпізнавання та обробки мови.

З використанням Python можливе розроблення алгоритмів, які не просто "розуміють" природну мову, але й здатні виконувати складні завдання, такі як автоматичний переклад, генерація тексту, розпізнавання емоцій в тексті та інше. Ця глибока інтеграція Python у світ NLP показує важливість інновацій у цій галузі для сучасного цифрового суспільства.

Актуальність розробки алгоритмів програмного коду на основі розпізнавання природної мови полягає в можливості розробників використовувати природну мову

для формулювання завдань, що робить процес програмування більш інтуїтивним. Такий підхід знижує бар'єри для входження в галузь програмування та має потенціал кардинально трансформувати IT-індустрію.

Мета і завдання дослідження. Метою кваліфікаційної роботи є розробка алгоритму програмного коду на основі розпізнавання природної мови.

Об’єктом дослідження процес розпізнавання та перетворення природної мови в програмний код.

Предметом дослідження є розробка алгоритму генерації програмного коду на основі розпізнавання природної мови.

Для досягнення цієї мети необхідно розв’язати наступні задачі:

1. Проаналізувати природні мови їх характеристики та особливості аналізу.
2. Проаналізувати мови програмування високого рівня класифікації та сфери застосування.
3. Інтегрувати середовища створення програмних кодів.
4. Розробити алгоритм розпізнавання звукових сигналів.
5. Розробити алгоритм генерації програмного коду та основи розпізнавання природної мови.
6. Провести дослідження розроблених алгоритмів.

Методи дослідження базуються на використанні інтегрованих технологій розпізнавання природної мови, алгоритмічного моделювання, машинного навчання, та програмування. Цей підхід включає в себе застосування інноваційних бібліотек Python, таких як pyautogui для роботи з графічним інтерфейсом користувача, speech_recognition для перетворення голосу в текст, і gTTS для генерації мовлення з тексту.

Наукова новизна одержаних результатів. Наукова новизна дослідження полягає в розробці інтегрованої системи, яка з'єднує технології розпізнавання природної мови з механізмами автоматичного генерування програмного коду. Це дозволяє користувачам перетворювати мовні запити на синтаксично правильний програмний код, відкриваючи нові можливості для інтерактивної розробки програмного забезпечення.

Практичне значення отриманих результатів. Отримані результати в магістерській роботі відіграють значну роль у покращенні процесу розробки програмного забезпечення, пропонуючи ефективний інструмент для автоматизації генерації коду. Ця інновація спрощує прототипізацію та розробку, зокрема для рутинних завдань, тим самим підвищуючи продуктивність розробників. Водночас, система відкриває двері для не-програмістів до світу програмування, дозволяючи їм створювати програми за допомогою простих мовних команд.

Публікації та апробація випускної кваліфікаційної роботи. Отримані результати апробовані в межах VIII науково-практичної конференції молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі» Західноукраїнського національного університету та опубліковано дві тези доповідей за темою роботи [3,4].

Кваліфікаційна робота складається із трьох розділів, висновків, списку використаної літератури та додатків.

У першому розділі систематизовано та описано алгоритми розпізнавання звукових повідомлень.

У другому розділі розроблено алгоритми розпізнавання звукових повідомлень.

У третьому розділі розроблено та реалізовано алгоритми розпізнавання звукових сигналів, інтерфейс додатку. Проведено тестування по функціонуванню розроблених алгоритмів.

1. ПРОГРАМНІ СИСТЕМИ АВТОМАТИЗОВАНОЇ ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ

Сучасний світ програмної інженерії прогресує надзвичайно швидкими темпами. Інноваційні технології та методики розробки не тільки підвищують ефективність процесів, але й роблять їх більш зручними та доступними. У цьому контексті автоматизована генерація коду виступає як один з важливих інструментів, що революціонізують підхід до створення програмних продуктів.

Автоматизована генерація програмного коду – це процес, під час якого програмний код генерується автоматично за допомогою певних інструментів і методик, замість того, щоб бути написаним вручну розробниками. Такий підхід підвищить продуктивність, зменшити кількість помилок та забезпечити консистентність коду.

За допомогою автоматизації можна відтворювати фрагменти програмного коду, базуючись на попередньо визначених параметрах і шаблонах. Такий підхід спрямований на максимальне використання готових рішень, що дозволяє зекономити час і зосередитися на унікальних аспектах конкретного проекту.

Автоматизована генерація коду також сприяє стандартизації. Коли код генерується автоматично, можна бути впевненим, що він відповідає певним стандартам якості та кращим практикам. Це особливо важливо великим командам розробників, де консистентність коду є ключовим фактором успіху.

Але що робить автоматизовану генерацію коду дійсно особливою, так це її спроможність адаптуватися. Зміни в бізнес-логіці або технічних вимогах часто призводять до потреби в корекції коду. Інструменти автоматизованої генерації можуть швидко відреагувати на такі зміни, що забезпечує високу гнучкість розробки.

Тим не менше, важливо розуміти, що автоматизована генерація коду не є універсальним рішенням для всіх сценаріїв. Існують ситуації, коли специфічні

вимоги або унікальна бізнес-логіка вимагають ручного кодування. Тому цей підхід краще розглядати як доповнення до традиційних методів розробки, а не як їх заміну.

Переваги автоматизованої генерації коду:

1. Автоматизація рутинних аспектів кодування суттєво скоротить час розробки, дозволяючи командам швидше виводити продукти на ринок.

2. Машини менш схильні до помилок у порівнянні з людьми, особливо при генерації рутинного або шаблонного коду. Це підвищить якість коду та зменшити кількість дефектів.

3. Автоматична генерація коду допоможе забезпечити, що весь код відповідає визначеним стандартам, забезпечуючи узгодженість і консистентність у всьому проекті.

4. Завдяки використанню визначених шаблонів та специфікацій можливо одержати однаковий код при кожному запуску генератора, що полегшує повторне використання та розширення.

5. Деякі сучасні системи генерації коду можуть адаптуватися до змін у вимогах, автоматично оновлюючи відповідні частини коду.

6. Використання автоматизованої генерації коду зменшить потребу в великих командах розробників для рутинних задач, що дозволяє перерозподілити ресурси на більш складні частини проекту.

7. Сучасні інструменти генерації коду часто мають можливість інтеграції з системами управління базами даних, дизайнерськими інструментами та іншими програмними рішеннями, спрощуючи процес розробки.

8. За допомогою автоматичної генерації коду можливе також створення юніт-тестів або інших тестових сценаріїв, що забезпечують надійність коду.

Ці переваги роблять автоматизовану генерацію коду цінним інструментом у сучасній індустрії програмної розробки. Тим не менше, важливо пам'ятати про можливі обмеження та виклики, а також вибирати правильний інструмент для конкретного проекту.

Недоліки автоматизованої генерації коду:

1. Незважаючи на те, що автоматизована генерація коду може бути ефективною для стандартизованих задач, вона є менш гнучкою у відповіді на унікальні вимоги або специфічні сценарії. Іншими словами, не завжди можливо автоматично генерувати код, який відповідатиме всім потребам проекту.

2. Автоматизовані інструменти часто генерують код, який є надмірно складним або непотрібним для конкретних потреб проекту. Це ускладнить процес налагодження і підтримки програми.

3. При зміні або виборі іншого інструменту для автоматизованої генерації коду може виникнути необхідність перегляду або переписання частини коду, що були згенеровані попереднім інструментом.

4. Проблеми із навчанням. Хоча інструменти автоматизованої генерації коду можуть спрощувати розробку, вони також вимагають від розробників спеціальних знань для їх ефективного використання. Це збільшить час навчання та адаптації.

5. Відсутність повного контролю. Розробники можуть втратити частину контролю над тим, як саме код генерується. Це призведе до ситуацій, коли згенерований код не повністю відповідає очікуванням або стандартам команди.

6. Потенційні проблеми із продуктивністю. У деяких випадках спроби автоматизувати генерацію коду можуть виявитися менш продуктивними, ніж ручна розробка, особливо якщо інструмент не відповідає конкретним потребам проекту.

7. Ризик знеособлення. Занадто сильна залежність від автоматизованих інструментів може призвести до втрати креативного підходу і глибокого розуміння коду серед розробників.

Основуючись на зазначених недоліках, важливо зрозуміти, що автоматизована генерація коду має своє місце в арсеналі інструментів розробника, але вона повинна застосовуватися обачно і в комбінації з іншими методами розробки.

Автоматизована генерація коду знаходить застосування в численних аспектах сучасної програмної інженерії. Ось деякі з основних областей застосування:

1. Багаторазове використання коду. Генерація коду може використовуватися для створення бібліотек коду, які можна повторно використовувати в різних проектах. Це забезпечує консистентність та скорочує час розробки.

2. Графічні інтерфейси користувача (GUI). Автоматична генерація коду для створення GUI дозволяє розробникам фокусуватися на логіці застосунку, а не на мануальному кодуванні елементів інтерфейсу.

3. Бази даних. Інструменти автоматизованої генерації можуть створювати SQL запити, скрипти міграції та інші компоненти роботи з базами даних, базуючись на моделях даних.

4. Інтеграція систем. Генерація коду може спрощувати процес інтеграції різних систем, автоматично створюючи мости між ними через API, міжплатформені шлюзи тощо.

5. Розробка для різних платформ. Зокрема в мобільній розробці, де потрібно підтримувати кілька платформ (наприклад, Android та iOS), автоматизована генерація коду може допомогти у створенні крос-платформних рішень.

6. Прототипування. Швидке створення прототипів застосунків з можливістю подальшого їх розширення та модифікації.

7. Системи управління вмістом (CMS). Автоматична генерація коду може використовуватися для створення плагінів, тем або інших компонентів для CMS.

8. Ігрова розробка. Інструменти генерації коду можуть автоматизувати створення повторюваних аспектів ігрової механіки або ресурсів.

9. Вбудовані системи. В сфері вбудованих систем, де ресурси обмежені, автоматизована генерація коду може допомогти оптимізувати розробку, забезпечуючи ефективність та відповідність специфікаціям.

Кожен з цих застосувань має свої особливості та вимоги, тому при виборі інструменту для автоматизованої генерації коду важливо враховувати конкретний контекст проекту та його потреби.

Природні мови використовуються для спілкування між людьми і є відмінними від формальних мов, таких як мови програмування або математичні нотації. Основною відмінністю є те, що природні мови не створені з нуля за певними правилами, а розвиваються і модифікуються під впливом культурних, соціальних та історичних процесів.

Однією з ключових характеристик природних мов є багатозначність. Одне й те ж слово чи фраза можуть мати різні значення в різних контекстах. Наприклад, слово "банк" може означати фінансову установу або річковий берег. Ця особливість робить аналіз природних мов особливо складним завданням.

Синтаксична складність також є дуже важливою особливістю природної мови. Граматичні структури можуть бути дуже складними, і одне й те ж повідомлення можна висловити безліччю способів, використовуючи різні граматичні конструкції.

Прагматичні аспекти мови пов'язані з тим, як мова використовується в певному контексті. Це включає в себе такі речі, як сарказм, жарти та інші аспекти, які можуть змінювати зміст висловлювання залежно від ситуації.

Ідіоматичні вирази — це ще одна особливість природних мов. Ці вирази використовуються у мовленні носіїв мови, і їх розуміння часто вимагає знання конкретного культурного або мовного контексту. Вони додають виразності та кольору мовленню, і їх використання допомагає розмовнику виражати свої думки більш ефективно.

Що стосується аналізу природних мов, тут зіштовхуємося з рядом завдань, починаючи від простої токенізації, коли текст розбивається на окремі слова, до складних завдань, таких як семантичний аналіз, де має бути визначено зміст речення або фрази. Також важливо враховувати мовні відмінності, коли аналізується текст на різних мовах.

В цілому, природні мови представляють величезний інтерес для дослідників, оскільки їх розуміння та обробка може призвести до революційних змін в технологіях спілкування та інтерфейсах користувача.

1.1 Природні мови їх характеристики та особливості аналізу

Природна мова — це мова, яку використовують для спілкування між людьми. Відмінно від формальних мов, таких як мови програмування або математичні нотації, природні мови не створені за певним набором правил, але, натомість, розвиваються і модифікуються під впливом культурних, соціальних та історичних факторів.

Багатозначність є іншою важливою характеристикою природних мов. Те ж саме слово може мати різні значення в залежності від контексту. Ця особливість може створювати виклики під час автоматичного аналізу мови, так як машині важко визначити, яке саме значення має бути вибрано в конкретній ситуації. Приклад багатозадачності: "Лисичка" може вказувати на тварину, або бути прізвищем людини, або назвою різновиду грибів.

Природні мови також мають складну граматичну структуру, що включає в себе морфологію, синтаксис та семантику. Кожна мова має свої власні граматичні правила, які диктують порядок слів, часів, відмінки та інші аспекти структури речень.

«Повнофункціональна» автоматична обробка природної мови передбачає розуміння контексту, врахування культурних та соціальних особливостей мови, а також здатність до адаптації до динамічних змін в мові. Така обробка вимагає застосування розширених технік машинного навчання, семантичного аналізу та інших методів, які можуть враховувати всю багатогранність та глибину природних мов.

Синтаксична структура природних мов часто є дуже складною, дозволяючи велику гнучкість у формуванні висловлювань. Це може створювати виклики при спробах аналізу таких мов, особливо коли мова містить велику кількість винятків з правил.

Природні мови також відзначаються наявністю ідіоматичних виразів — фраз, значення яких не завжди може бути зрозумілим з аналізу окремих слів, що їх

складають. Наприклад, вираз "взяти бика за роги" вказує на дії, спрямовані на активне рішення проблеми, а не на буквальне взяття бика. Або ж "Іван купив м'яч, і він був червоним." Тут "він" вказує на м'яч, а не на Івана.

Деякі слова чи вирази можуть мати різне значення в різних культурних контекстах. Наприклад, "червоний" в західній культурі часто асоціюється з любов'ю, небезпекою або гнівом, тоді як в китайській культурі - зі щасливими подіями та святами.

Нові слова або вирази постійно з'являються в мові, і старі можуть отримувати нові значення. Наприклад, слово "твіт" раніше асоціювалося з пташиним співом, а тепер частіше вживається як термін у соціальних мережах.

Аналіз природних мов (Natural Language Processing, NLP) включає вивчення та розуміння природних мов з точки зору комп'ютера. Однією з основних складнощів у цьому є полісемія, явище, коли одне слово має декілька значень. Контекстуальні вказівки - ключові для розуміння, яке саме значення мається на увазі в даному випадку, але інтерпретація цих вказівок може бути складною для алгоритмів.

Прогрес в галузі NLP досягнутий завдяки розвитку алгоритмів машинного навчання, особливо глибокого навчання. Моделі, як BERT (Bidirectional Encoder Representations from Transformers) або GPT (Generative Pre-trained Transformer), використовують обробку природної мови для того, щоб краще зрозуміти людську мову, використовуючи величезні набори даних для тренування та вдосконалення своїх алгоритмів.

З появою комп'ютерних технологій та штучного інтелекту методи аналізу природних мов стали ще більш розширеними. Завдяки алгоритмам машинного навчання можна визначати емоційний колір тексту, виявляти ключові слова, або навіть перекладати текст з однієї мови на іншу. Тим не менше, аналіз природних мов залишається складною задачею через величезну варіативність та багатогранність мовних явищ.

Імперативні мови програмування - це підхід до програмування, де програміст вказує "як" комп'ютер повинен виконувати задачу крок за кроком. В імперативному

стилі програмування алгоритми описуються як послідовність команд для комп'ютера. Серед імперативних мов можна знайти дуже високорівневі мови, такі як Python, та більш низькорівневі мови, такі як C.

Ось декілька особливостей імперативних мов:

1. Послідовність команд. Програми, написані на імперативних мовах, зазвичай складаються з послідовності команд, які виконуються одна за одною.

2. Змінні. Імперативні мови використовують змінні для зберігання даних. Ці змінні можуть змінювати своє значення в процесі виконання програми.

3. Керуючі структури. Імперативні мови містять керуючі структури для управління порядком виконання команд, такі як цикли (for, while), умовні оператори (if-else) та ін.

4. Підпрограми. Імперативні мови часто використовують функції або процедури для групування команд та повторного використання коду.

5. Безпосереднє взаємодійство з пам'яттю. Деякі імперативні мови, особливо низькорівневі, дозволяють безпосередньо взаємодіяти з пам'яттю, що надає гнучкість, але також збільшує ризик помилок.

Приклади імперативних мов:

- C — відома своєю ефективністю і безпосереднім доступом до апаратних засобів.
- C++ — розширення мови C з додатковими об'єктно-орієнтованими можливостями.
- Java — забезпечує платформо-незалежність завдяки використанню віртуальної машини.
- Python — відзначається простотою синтаксису та широким застосуванням.

Імперативне програмування є однією з найпоширеніших парадигм програмування, особливо для системного та застосункового програмування, де потрібен високий рівень контролю над ресурсами комп'ютера. У цьому стилі програмування програміст описує послідовність операцій, які повинні бути виконані комп'ютером для досягнення конкретної мети.

1.2 Мови програмування високого рівня класифікації та сфери застосування

Мови програмування високого рівня (ВРМ) - це мови, розроблені таким чином, щоб бути більш зрозумілими для людей, ніж для машин. Вони віддаляють програміста від деталей апаратної частини комп'ютера та спрощують процес програмування завдяки абстрактному синтаксису і річному набору вбудованих функцій.

Сучасні мови програмування високого рівня можна класифікувати за різними критеріями. Однією з таких класифікацій є за призначенням: системні мови, призначені для написання операційних систем, драйверів та інших програм, які працюють безпосередньо з апаратурою, та прикладні мови, які використовуються для створення різноманітних програмних застосунків.

Також є мови об'єктно-орієнтованого програмування, які зосереджуються на створенні об'єктів та класів, та процедурні мови, де головний акцент робиться на функціях і процедурах. Функціональні мови програмування базуються на концепціях математичного функціонального аналізу.

Використовуючи високорівневі мови, можна ефективно використовувати модулі та бібліотеки, що дозволяє швидко інтегрувати готові рішення у свої проекти. Це зокрема актуально для розробки складних систем, де необхідно оперативно вносити зміни та додавати новий функціонал.

З ростом обсягів даних і потреб в обробці цих даних, мови програмування високого рівня також адаптуються для оптимальної роботи з великими базами даних, хмарними рішеннями і розподіленими системами. Так, наприклад, мова Python стала особливо популярною у сфері аналітики даних та штучного інтелекту завдяки своєму багатому екосистемі бібліотек та інструментів.

Також варто зазначити, що з розвитком технологій і зростанням потреб користувачів, мови високого рівня постійно модернізуються. Це призводить до виникнення нових стандартів, версій та діалектів цих мов. Зокрема, мови, які були

популярні декілька десятиліть тому, можуть залишитися в історії, а на їхнє місце приходять нові, більш гнучкі та потужні мови програмування.

Тим не менше, важливо пам'ятати, що вибір мови програмування завжди повинен базуватися на конкретних вимогах і задачах проекту. Незважаючи на універсальність багатьох сучасних мов, кожна з них має свої особливості, які роблять її особливо підходящою для певних завдань.

Сфери застосування мов високого рівня надзвичайно різноманітні. Наприклад, Python часто використовується в наукових розрахунках, веб-розробці, розробці ігор та штучному інтелекті. Java є основною мовою для розробки Android-додатків. JavaScript домінує у веб-розробці. C++, завдяки своїй продуктивності та гнучкості, використовується у різних областях, включаючи системну розробку, ігри та робототехніку.

У величезному світі мов програмування високого рівня існують і такі, що фокусуються на конкретних доменах або проблемах. Наприклад, R став популярним завдяки своїм можливостям у статистиці та аналізі даних. SQL, хоча і не є традиційною мовою програмування в повному розумінні цього терміну, спеціалізується на роботі з базами даних.

Кожна мова має свої особливості, що визначають її підхід до розробки. Ruby, наприклад, славиться своєю чистотою синтаксису та елегантністю, а також широко використовується для розробки веб-додатків завдяки фреймворку Ruby on Rails. Swift був розроблений компанією Apple для створення додатків для iOS та macOS, замінивши при цьому старіший Objective-C.

Останнім часом набирає популярності тенденція до використання мов програмування, що підтримують одночасне виконання коду. Такі мови, як Go або Rust, надають можливості для розпаралелювання завдань і забезпечують більшу продуктивність на сучасних багатоядерних процесорах.

В сучасному світі програмування є велика кількість мов програмування, і вибір підходящої мови може значно впливати на продуктивність розробки. Уявімо ситуацію: IT-компанія "TechSolution" вирішила проаналізувати ефективність використання різних мов програмування у своїх проектах. "TechSolution"

зосередила свою увагу на п'яти ключових мовах, які використовуються в їхніх проектах: Python, Java, C++, Ruby та JavaScript. Вони зібрали дані з останніх 50 проектів, виконаних для кожної мови, щоб визначити середній час розробки.

На основі зібраних даних стало видно, що проекти на Python та Ruby, в середньому, завершуються за 3 місяці, тоді як проекти на Java та C++ можуть тривати до 6 місяців. JavaScript проекти в середньому займають близько 4 місяців.

Дослідження також показало, що складність проектів могла відрізнятися. Проекти на Python часто були пов'язані з аналізом даних та машинним навчанням, тоді як проекти на C++ в основному стосувалися розробки вбудованого ПЗ та системного програмування.

Інше дослідження продемонструє пам’ять та час виконання, які використовуються 15-ма різними мовами програмування для вирішення гри, а саме: C, C++, C#, Java, Python, JavaScript, TypeScript, PHP, Kotlin, Swift, Rust, Go, Scala, Dart, Ruby. Умова гри: Петро та Степан по черзі грають у гру, Петро — перший. На дошці написане число n, кожного ходу гравець:

1. Вибирає будь-яке x, де $0 < x < n$ і $n \% x == 0$.
2. Замінює число n на дошці на $n - x$. Також, якщо гравець не може зробити хід, він програє гру.

Повернути " true", якщо Петро виграє гру, припускаючи, що обидва гравці грають оптимально.

Рішення: Якщо n парне, повернути "правда", інакше "неправда".

У додатку А рішення поставленої задачі усіма вказаними вище мовами програмування.

Також було виявлено, що Ruby та JavaScript часто використовуються для веб-розробки. Це може пояснити, чому проекти на Ruby завершувались швидше: вони могли бути менш складними порівняно з іншими мовами.

Основна думка дослідження полягала в тому, що вибір мови програмування може значно впливати на час реалізації проекту, але також важливо враховувати характеристики конкретного проекту.

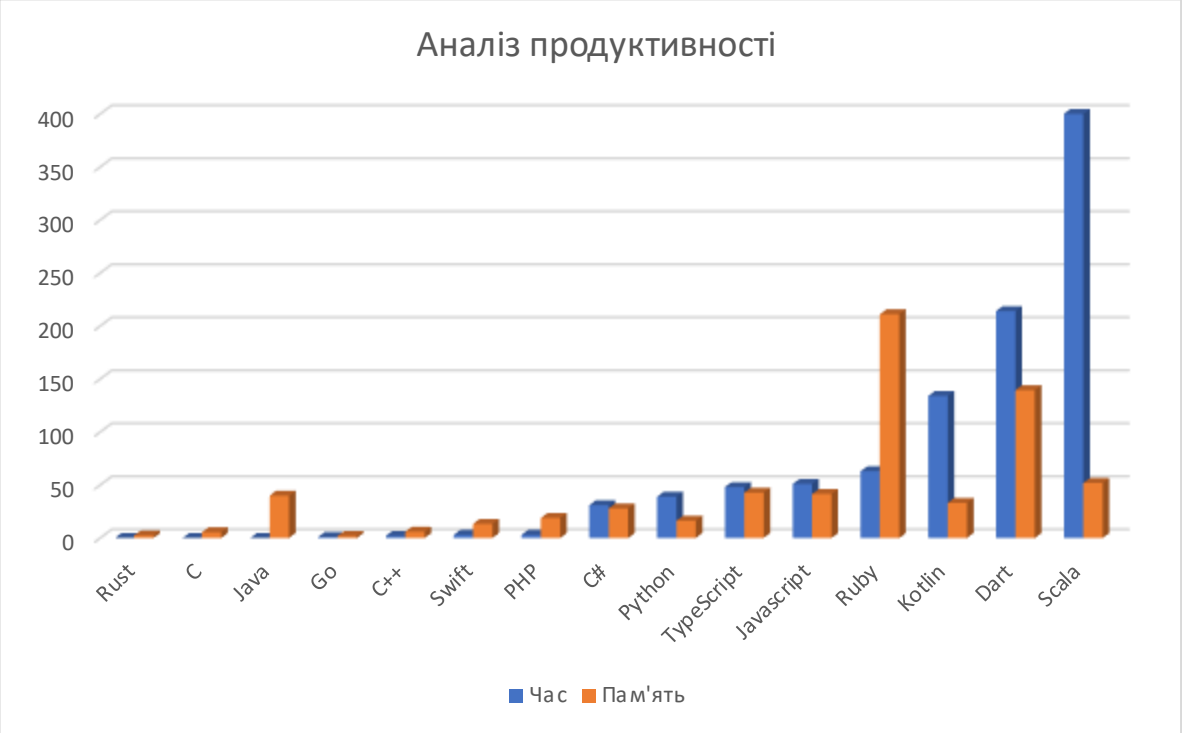


Рисунок 1.1 – Аналіз продуктивності 15-ти мов програмування

Висновок досліду:

- C та C++ – це компільовані мови, їхній вихідний код перекладається у машинний код під час компіляції. Це призводить до швидших часів виконання порівняно з інтерпретованими мовами.
- Rust – це компільована мова. Її дизайн спрямований на надання контролю низького рівня як у C та C++, а також надання абстракцій високого рівня та гарантій безпеки. Це призводить до швидкої продуктивності в режимі реального часу та низького використання пам'яті.
- Java, C# та Kotlin компілюються до проміжного байт-коду, який потім виконується віртуальною машиною. Це призводить до повільніших часів роботи. Java компілюється швидше, ніж C# та Kotlin, через меншу кількість функцій. Ці мови також мають автоматичне керування пам'яттю, що призводить до вищого використання пам'яті.
- Python, Ruby, PHP та JavaScript – це інтерпретовані мови, що означає, що їх вихідний код декодується в режимі реального часу кожного разу, коли виконується команда. Це може призвести до повільної продуктивності виконання

порівняно з компільованими мовами. Ці мови також мають автоматичне керування пам'яттю, що може призвести до вищого використання пам'яті порівняно з мовами без автоматичного керування пам'яттю.

- Go – це компільована мова з акцентом на простоту та паралельність. Її дизайн спрямований на надання швидких часів компіляції та ефективного виконання.
- Swift – це компільована мова з акцентом на безпеку та виразність. Її дизайн спрямований на надання швидкої продуктивності виконання, будучи при цьому зручною для читання та написання.
- TypeScript – це строгий синтаксичний надмножина JavaScript, яка додає необов'язкові типові анотації. Код TypeScript компілюється у JavaScript, який потім виконується JavaScript-двигуном.

Різні мови програмування мають свої особливості компіляції, виконання та управління пам'яттю. Мови C та C++ компілюються безпосередньо у машинний код, що гарантує високу швидкість виконання, тоді як інтерпретовані мови, такі як Python та JavaScript, декодуються в реальному часі, що може знижувати продуктивність. Rust і Swift комбінують низькорівневий контроль з високорівневими абстракціями та безпекою. З іншого боку, мови, такі як Java та Kotlin, використовують проміжний байт-код для виконання віртуальною машиною, що може впливати на час виконання.

1.3 Інтегровані середовища створення програмних кодів

Інтегроване середовище розробки (ICP, або IDE - Integrated Development Environment) — це програмний пакет, який надає програмістам комплексні функції для розробки програмного забезпечення.

Інтегровані середовища розробки (IDE) не лише полегшують сам процес написання коду, але й активно сприяють підвищенню якості програмного продукту завдяки комплексу вбудованих інструментів. Особливо важливим є здатність IDE автоматично ідентифікувати та попереджати про потенційні помилки, робити пропозиції щодо оптимізації коду та надавати рекомендації по його структуризації.

Основна ідея IDE полягає в тому, щоб надати універсальний "одно-віконний" інтерфейс для всіх етапів процесу розробки: написання коду, його відлагодження, тестування та, за потреби, деплоймент. Зазвичай, середовища розробки включають в себе текстовий редактор з підсвічуванням синтаксису, інструменти для автоматичного аналізу коду, відлагоджувач, компілятор або інтерпретатор, засоби для автоматичної збірки програми та інші корисні інструменти.

Традиційно, ІСР надає текстовий редактор для написання коду, компілятор та/або інтерпретатор для його виконання, а також засоби для відлагодження. Останнім часом більшість сучасних ІСР також інтегрували підтримку систем контролю версій, автоматизацію збірки, управління залежностями, інструменти для автоматизованого тестування та інше.

Додатково, деякі ІСР мають специфічну спрямованість, наприклад, для розробки мобільних додатків, веб-додатків, наукових досліджень чи графічних інтерфейсів. Інші ж розроблені так, щоб бути якнайбільш універсальними і підтримувати різні мови програмування та платформи.

Інтегровані середовища розробки містять низку ключових компонентів, які спільно сприяють ефективності та продуктивності розробників:

- Текстовий редактор: Центральна частина будь-якого ІСР, яка дозволяє розробникам писати та редагувати програмний код. Він часто має підсвітку синтаксису, автодоповнення та інші функції, що полегшують написання коду.
- Компілятор та/або інтерпретатор: Ці інструменти перетворюють вихідний код, написаний розробниками, на машинний код або інтерпретують його в режимі реального часу. Це дозволяє розробникам виконувати та тестувати їх програми безпосередньо в ІСР.

- Засоби для відлагодження: Ці інструменти дозволяють розробникам виявляти, локалізувати та усувати помилки у своєму коді. Вони можуть надавати інформацію про стан програми, змінні, стек викликів та інше.
- Інтеграція з системами контролю версій: Багато ICP мають вбудовані засоби для роботи з системами контролю версій, такими як Git. Це дозволяє розробникам легко стежити за змінами у коді, співпрацювати з іншими та управляти версіями своїх проектів.
- Автоматизація збірки та розгортання: Деякі ICP мають інструменти, які автоматично компілюють та розгортають програми, допомагаючи розробникам ефективно виводити свої продукти на ринок.
- Управління залежностями: Дозволяє розробникам додавати, оновлювати та видаляти бібліотеки та пакети, які використовуються у їхньому проекті, автоматизуючи процес управління необхідними ресурсами.
- Графічні дизайнери: Для тих, хто розробляє графічні інтерфейси користувача, деякі ICP містять візуальні редактори, які дозволяють створювати інтерфейси за допомогою перетягування елементів.

Інтелектуальне завершення коду є однією з найбільш корисних особливостей сучасних інтегрованих середовищ розробки. Ця функція автоматично пропонує варіанти для завершення рядка коду, коли розробник починає його вводити. Інтелектуальне завершення коду базується на аналізі поточного контексту коду, включаючи використані змінні, методи та бібліотеки.

Ця особливість має ряд важливих переваг:

1. Підвищення продуктивності: Розробники можуть швидше писати код, користуючись пропозиціями, ніж якби вони вводили весь код вручну.
2. Зменшення кількості помилок: Автоматичне завершення допомагає уникнути опечаток або невірного використання методів чи функцій.
3. Покращення навчання: Для новачків або тих, хто не добре знайомий з певною бібліотекою чи мовою програмування, інтелектуальне завершення коду може служити навчальним засобом, показуючи доступні методи або властивості.

4. Стандартизація коду: При пропонуванні варіантів завершення коду ICP може пропонувати найбільш прийнятні або стандартні варіанти, що сприяє написанню більш стандартного та чистого коду.

5. Швидке виявлення методів та змінних: Замість перегляду документації або пошуку в Інтернеті розробники можуть просто почати вводити ім'я класу або об'єкта, щоб побачити, які методи або властивості доступні для нього.

Підтримка плагінів та розширень в інтегрованих середовищах розробки (ICP) стала критично важливою, так як вона надає можливість розробникам індивідуально налаштовувати та модифікувати своє робоче середовище, враховуючи особисті потреби та вимоги. Плагіни та розширення додають нові можливості до основної програми ICP. Вони можуть варіюватися від простих інструментів для форматування коду до більш складних систем для аналізу коду або підтримки нових мов програмування.

Адаптація середовища під потреби розробника надає гнучкість і персоналізацію, дозволяючи розробникам вибирати та використовувати лише ті інструменти та функції, які їм дійсно потрібні. Крім того, завдяки плагінам, розробники можуть швидко адаптуватися до нових технологій, отримуючи підтримку нових мов програмування, фреймворків або інструментів. Це також забезпечує оптимізацію робочого процесу, автоматизуючи рутинні завдання і підвищуючи продуктивність. Плагіни та розширення можуть інтегрувати ICP з іншими зовнішніми інструментами та сервісами, такими як системи контролю версій або хмарні платформи.

Така підтримка дозволяє розробникам створювати робоче середовище, яке ідеально підходить для їхніх потреб, сприяє швидкому пристосуванню до нових технологічних трендів та гарантує високий рівень продуктивності.

Інтегровані середовища розробки часто надають інтеграцію з системами контролю версій, що забезпечує розробникам зручний доступ та управління кодом прямо з середовища розробки. Однією з найпопулярніших систем контролю версій є Git, який дозволяє командам ефективно управляти змінами у коді, слідкувати за історією модифікацій та співпрацювати над проектами.

Інтеграція ICP з Git дає можливість розробникам здійснювати коміти, створювати гілки, злиття змін та інші операції Git без необхідності переключення між консолью та редактором коду. Це спрощує процес розробки, так як весь робочий процес від редагування коду до його відправки на сервер може відбуватися в одному вікні.

Крім Git, багато ICP також підтримують інтеграцію з іншими системами контролю версій, такими як Mercurial, Subversion чи Perforce. Така інтеграція надає розробникам гнучкість у виборі найбільш підходящої системи для їхнього проекту та забезпечує максимальну ефективність роботи.

Взаємодія з системами контролю версій через ICP забезпечує розробникам не тільки зручність, але й продуктивність, допомагаючи їм заощадити час та уникнути можливих помилок, які можуть виникнути під час ручного управління версіями коду.

Інтегровані середовища розробки (ICP) стали незамінним інструментом для розробників завдяки своїй універсальності та мультиплатформенності. Мультиплатформність означає, що ICP може працювати на різних операційних системах, таких як Windows, macOS або Linux, надаючи користувачам однаковий досвід незалежно від їхнього вибору ОС. Це дозволяє командам розробників, які користуються різними платформами, легко співпрацювати над спільними проектами.

Що стосується універсальності, багато сучасних ICP можуть підтримувати кілька мов програмування. Це дозволяє розробникам працювати з різними проектами, які використовують різні технології, без необхідності переключатися між різними середовищами розробки. Наприклад, розробник може працювати над веб-додатком на JavaScript, мобільним додатком на Kotlin і системним додатком на C++ у тому ж самому ICP.

Крім того, підтримка різних платформ та мов часто розширюється завдяки плагінам та розширенням, які розробляються спільнотою. Це дозволяє ICP залишатися актуальними та адаптованими до найновіших технологій та стандартів розробки.

Така мультиплатформність та універсальність дозволяє розробникам зосередитися на своїй роботі, замість того щоб турбуватися про технічні деталі або пошук відповідного інструментарію для кожного окремого проекту. Завдяки цьому процес розробки стає більш ефективним і продуктивним.

Інтегровані середовища розробки (ICP) відіграють ключову роль у сучасному програмуванні, допомагаючи розробникам ефективно створювати, тестувати та впроваджувати код. Деякі з найпопулярніших ICP, які використовуються в індустрії програмного забезпечення, включають Visual Studio, Eclipse, IntelliJ IDEA, Xcode та PyCharm.

Visual Studio від Microsoft є одним з найбільш універсальних ICP, підтримуючи ряд мов програмування та платформ. Його можна використовувати для розробки додатків на C#, VB.NET, C++, і багато інших мов.

Eclipse, що розроблено Eclipse Foundation, спершу було спрямовано на розробку Java-додатків, але з часом воно розширилося завдяки плагінам, щоб підтримувати ряд інших мов і технологій.

IntelliJ IDEA від JetBrains є популярним вибором серед розробників Java. Це середовище відоме своєю потужною системою інтелектуального завершення коду та глибокою інтеграцією з різними фреймворками.

Xcode - це ICP від Apple, що спеціалізується на розробці для платформ iOS, macOS, watchOS і tvOS, зокрема використовуючи мови Swift та Objective-C.

PyCharm також від JetBrains спеціалізується на розробці на Python і включає в себе ряд інструментів для професійної розробки, таких як відладка, тестування та профілювання.

Ці приклади ICP представляють лише верхівку айсберга великої кількості доступних інструментів. Вибір певного ICP часто залежить від конкретних потреб проекту, комфорту розробника та специфіки мови або платформи, для якої ведеться розробка.

Однією з найбільших переваг використання IDE є збільшення продуктивності розробника завдяки автоматизації рутинних задач, швидкому доступу до документації, а також можливості відлагодження коду безпосередньо в середовищі

розробки. Ці інструменти також часто інтегруються з системами контролю версій, як-от Git, що полегшує процес управління змінами в коді та співпрацю в команді.

Багато сучасних IDE інтегруються з зовнішніми інструментами і сервісами. Наприклад, з системами неперервної інтеграції та доставки (CI/CD), що дозволяє автоматизувати процес тестування і випуску нових версій програми.

Також слід відзначити, що сучасні IDE активно використовують хмарні технології, дозволяючи розробникам працювати над проектами з будь-якої точки світу, мати доступ до потужних обчислювальних ресурсів для відлагодження та тестування, а також зберігати всі необхідні налаштування та інструменти у хмарі, що робить робоче місце розробника максимально гнучким і мобільним.

Співпраця і командна робота також стають простішими завдяки інтеграції з платформами для командної роботи, що дозволяє обговорювати код, ділитися ідеями та виконувати колективне код-рев'ю безпосередньо в рамках одного середовища.

За допомогою ICP програмісти можуть ефективно розробляти, тестувати та випускати програми, використовуючи одне централізоване середовище, яке забезпечує всі необхідні інструменти.

Сучасні ICP забезпечують набір інструментів, які допомагають розробникам зосередитися на тому, що вони роблять найкраще, – писати якісний код.

Основною цінністю ICP є їх здатність спрощувати комплексні задачі. Вони забезпечують інтелектуальне завершення коду, що значно прискорює процес написання коду, інтеграцію з системами контролю версій, яка сприяє ефективній командній роботі, а також можливість швидкого переходу між файлами проекту, класами або методами.

Більше того, ICP відкривають двері для розширення та налаштування завдяки плагінам, що дозволяє командам адаптувати середовище під конкретні потреби проекту. Це, у свою чергу, допомагає підвищити продуктивність розробки.

У контексті неперервного росту та еволюції технологій, ICP залишаються в центрі уваги, надаючи розробникам потужні інструменти для реалізації своїх ідей. Таким чином, їх важливість у сфері програмної інженерії можна лише посилити,

оскільки вони продовжують адаптуватися та реагувати на нові виклики та потреби індустрії.

1.4 Висновки та постановка задачі

Підсумовуючи інформацію щодо програмних систем автоматизованої генерації коду, можна визначити декілька ключових аспектів. Автоматизована генерація коду є важливим напрямком у сфері програмування, який спрямований на підвищення ефективності та продуктивності розробки за допомогою автоматизації рутинних процесів. Це не тільки прискорює процес створення програм, але й зменшує кількість помилок, які можуть виникнути під час ручного кодування.

Природні мови відрізняються своєю складністю та багатоманітністю, що ставить певні виклики для їх аналізу та обробки комп'ютерними системами. Проте, завдяки сучасним методам аналізу, можливо ефективно перетворювати природний мовний запит на програмний код.

Мови програмування високого рівня дозволяють розробникам фокусуватися на бізнес-логіці програми, не заглиблюючись у низькорівневі аспекти роботи системи. Інтегровані середовища створення програмних кодів в свою чергу забезпечують зручний інтерфейс та набір інструментів для ефективної розробки програм на цих мовах.

Отже, комбінація цих компонентів – автоматизована генерація коду, аналіз природних мов та використання мов програмування високого рівня в інтегрованих середовищах – відкриває нові горизонти для сучасної розробки програмного забезпечення.

На основі проведеного аналізу можна сформулювати основну задачу дослідження: розробити систему автоматизованої генерації програмного коду на основі аналізу природних мов. Ця система повинна враховувати особливості

природних мов, використовувати мови програмування високого рівня та інтегровані середовища створення програмних кодів для оптимальної та ефективної реалізації завдань.

Специфічні завдання включають:

1. Аналіз та вибір оптимальних методів аналізу природних мов для перетворення тексту в програмний код.
2. Розробка алгоритмів автоматизованої генерації коду.
3. Інтеграція розробленої системи в існуючі інтегровані середовища програмування.
4. Тестування та оптимізація системи з метою забезпечення її надійності та продуктивності.

2. МЕТОДИ ТА АЛГОРИТМИ РОЗПІЗНАВАННЯ ПРИРОДНИХ МОВ

Методи розпізнавання природних мов (NLP) стали основою для обробки та аналізу текстової інформації в сучасному світі. Вони дозволяють машинам "розуміти", обробляти та генерувати текст подібно до людської мови.

Однією з основних характеристик NLP є використання статистичних методів, таких як N-грами, які аналізують ймовірності послідовностей слів у тексті. Інший популярний статистичний метод — TF-IDF, що вимірює важливість конкретного слова в документі порівняно з усім текстовим корпусом.

Однак, не менш важливим є підхід на основі правил. Ці системи використовують визначений набір правил, щоб "зрозуміти" мову. Наприклад, граматичні дерева можуть бути використані для ілюстрації структурних і синтаксичних відносин між різними частинами речення.

З розвитком технологій глибокого навчання, нейронні мережі стали ключовими в NLP. Рекурентні нейронні мережі (RNN) вважаються особливо корисними для аналізу тексту, оскільки вони можуть враховувати послідовність вхідних даних. Трансформатори, такі як BERT та GPT, стали новим стандартом в галузі завдяки їхній здатності виконувати розширений семантичний аналіз тексту.

На заключення, семантичний аналіз в NLP зосереджується на визначенні значення слів і фраз. Тут методи, такі як Latent Semantic Analysis (LSA) або використання баз даних, як WordNet, можуть допомогти у визначенні семантичних відносин між словами.

Головною метою є перетворення тексту на математичні моделі та вирази для глибшого зрозуміння його змісту та подальшої роботи, такої як класифікація чи розбиття на частини.

Порівняння інструментів NPL наведено у таблиці 1, де "Швидкість" відноситься до швидкості обробки, "Мови" до підтримуваних мов, "Моделювання"

до наявності інструментів для моделювання тем або інших аналітичних задач, а "Інтеграція" до можливості інтеграції з іншими платформами та інструментами.

Таблиця 1.1 – Порівняння інструментів NPL

Інструмент	Основний фокус	Швидкість	Мови	Моделювання	Інтеграція
NLTK	Загальна обробка тексту	Середня	Багато	Так	Помірна
spaCy	Глибоке навчання, аналіз тексту	Висока	Декілька	Ні	Висока
Gensim	Моделювання тем	Висока	Декілька	Так	Помірна
Transformer	Сучасні архітектури (BERT, GPT-2)	Середня	Декілька	Так	Висока
Stanford NLP	Розпізнавання сутностей, парсинг	Помірна	Багато	Так	Середня

Існує ряд потужних інструментів для обробки природної мови, кожен з яких має свої унікальні особливості. Вибір конкретного інструмента буде залежати від конкретних завдань, вимог до швидкості, підтримки мов, а також необхідності інтеграції.

Обробка природної мови часто пов'язана із обробкою текстової даних, включаючи аудіо та відео. Центральним моментом є приведення різних варіантів слів до їх базової форми. Також важливим є визначення ступеня схожості текстових рядків. Для цього використовуємо ряд метрик, які дозволяють зрозуміти різницю між рядками.

Метрика Edit distance (інакше називається відстанню Левенштейна) є однією з таких методик. Цей алгоритм вимірює схожість двох текстових послідовностей, базуючись на мінімальній кількості дій, потрібних для перетворення одного тексту в інший.

Остання комірка матриці (нижній правий кут) показує відстань редагування між двома словами, у нашому випадку це 3. Тобто потрібно 3 операції, щоб перетворити слово "кіт" на слово "книга" або навпаки.

Цей підхід широко використовується у програмах для обробки природної мови, зокрема:

- у системах автоматичної корекції орфографії;
- у біоінформатиці для порівняння подібності ДНК-послідовностей;
- при аналізі тексту, щоб визначити схожість слів, які розташовані поблизу певних текстових елементів.

Косинусна схожість — це метод, призначений для визначення подібності текстів у різноманітних документах. Для обчислення цього показника використовують формулу косинусного співвідношення векторів.

Припустимо,є два короткі текстові документи:

Документ А: "Я люблю футбол".

Документ В: "Я люблю баскетбол".

Спочатку перетворюємо кожен документ у вектор, використовуючи модель "мішок слів" (bag of words):

Вектор *A: [1, 1, 1]* (де 1 - це входження слів "Я", "люблю" і "футбол") Вектор *B: [1, 1, 1]* (де 1 - це входження слів "Я", "люблю" і "баскетбол")

Тепер, використовуючи формулу косинусної схожості,визначаємо схожість між цими двома векторами:

$$\cos(\theta) = \frac{\overline{A * B}}{\overline{\|A\| * \|B\|}},$$

де "*A * B*" - це добуток векторів, *a "||A||" і "||B||"* - це норми векторів *A* та *B* відповідно.

У прикладі, косинусна схожість буде близькою до 1 (але не рівною 1, оскільки слова "футбол" і "баскетбол" відрізняються), що свідчить про високу схожість між двома документами.

Цей приклад є дуже спрощеним і в реальних умовах документи містять значно більше слів, а використовувані методи векторизації можуть бути набагато складнішими.

Обчислення за допомогою косинусної схожості вказують на рівень співпадіння між текстами і можуть бути виражені через косинусні або кутові величини.

Тому вираховані показники косинусної схожості можуть служити інструментом для базового семантичного аналізу тексту.

Векторизація — це процес перетворення тексту або будь-якої іншої інформації в числові вектори фіксованого розміру, які можна обробляти математичними та статистичними методами. У контексті обробки природної мови (NLP) векторизація дозволяє перевести слова, фрази або цілі текстові документи в числову форму, зробивши їх придатними для аналізу та обробки за допомогою машинного навчання.

TF-IDF є скороченням для "частоти терміну - зворотньої частоти документа" і представляє собою відомий метод вирішення задач обробки природної мови. Ця методика дає можливість визначати релевантність конкретного слова або терміну до тексту у наборі документів.

Суть методу: якщо певний термін часто зустрічається в одному документі, але рідко в інших документах, то він вважається особливо значущим для цього документа.

Для реалізації TF-IDF використовуються два підходи:

1. TF (частота терміну), як часто слово з'являється в документі. Формула розрахунку:

$$TF(t) = \frac{\text{Кількість разів слово } t \text{ з}' \text{являється в документі}}{\text{Загальна кількість слів у документі}}.$$

2. IDF (зворотня частота документа) — визначає інверсне відношення кількості документів, в яких з'являється слово, до загальної кількості документів. Формула розрахунку:

3.

$$IDF(t) = \log\left(\frac{\text{Загальна кількість документів}}{\text{Кількість документів, дез'являється } t}\right).$$

TF-IDF для слова у документі розраховується як:

$$TFIDF(t) = TF(t) * IDF(t).$$

Розглянемо невеликий корпус, який містить три документи:

1. Документ 1: "Яблуко випало з дерева."

2. Документ 2: "Сонце світить над яблуневим садом."

3. Документ 3: "Сонце завжди світить яскраво."

Зосередимось на слові "сонце".

TF (частота терміну) для слова "сонце":

- Документ 1: 0 (тому що слово "сонце" не з'являється в документі)
- Документ 2: 1/6 (тому що слово "сонце" з'являється 1 раз з 6 слів)
- Документ 3: 1/4 (тому що слово "сонце" з'являється 1 раз з 4 слів)

IDF (зворотня частота документа) для слова "сонце":

$$IDF(\text{сонце}) = \log\left(\frac{\text{Загальна кількість документів (3)}}{\text{Кількість документів, де з'являється "сонце" (2)}}\right).$$

Отже, TF-IDF для слова "сонце" буде:

- Документ 1: 0 (тому що слово "сонце" не з'являється в документі)
- Документ 2: (1/6) * IDF("сонце")
- Документ 3: (1/4) * IDF("сонце")

Виходячи з розрахунків, слово "сонце" має більший коефіцієнт TF-IDF у Документі 3, ніж у Документі 2, що вказує на його більшу важливість для Документа 3 в рамках даного корпусу.

Алгоритм TF-IDF – це ключова методика у світі обробки природної мови, особливо у задачах, які пов'язані з видобутком інформації та аналізом тексту. Ця

методика дозволяє виміряти "важливість" слова у конкретному документі на тлі всього корпусу документів. Ця "важливість" вимірюється на основі того, як часто термін зустрічається в документі відносно його з'явлення у інших документах.

У підсумку, алгоритм TF-IDF є дуже корисним універсальним інструментом для вирішення численних задач на текстових даних, дозволяючи витягти суттєву інформацію з великих обсягів тексту та відсіяти незначущі елементи.

Нормалізація тексту – це процес перетворення тексту в його "стандартний" або "нормалізований" вигляд, що допомагає поліпшити консистентність, порівнюваність та аналіз даних. Це часто використовується в задачах обробки природної мови. Ось декілька поширених методів нормалізації:

1. Приведення до нижнього регістру, зміна всіх символів тексту на маленькі літери. Приклад: "ТЕКСТ" → "текст"

2. Видалення зайвих символів, таких як пунктуація, числа, спеціальні символи тощо. Приклад: "Привіт, світе!" → "Привіт світе"

3. Стемінг, відсічення словоформ до їх кореневої форми. Приклад: "граючи" → "гра"

4. Лематизація, приведення слова до його словникової (базової) форми з використанням словників та морфологічного аналізу. Приклад: "пішли" → "йти"

5. Видалення стоп-слів, які часто не мають значущої інформативної вартості для аналізу тексту (наприклад, "і", "та", "в", "на" тощо в українській мові). Приклад: "він її дуже любить" → "він дуже любить"

6. Токенізація, розбиття тексту на окремі слова або "токени". Приклад: "Я люблю музику." → ["Я", "люблю", "музику"]

Нормалізація тексту є важливим етапом підготовки даних у багатьох задачах обробки природної мови. Застосування відповідних методів нормалізації може суттєво покращити якість і консистентність аналізу тексту.

Стемінг і лематизація. Обидва ці методи мають на меті зводити словоформи до їх базової форми, але вони роблять це різними способами і за різними принципами.

Стемінг – це процес зменшення словоформи до її кореневої форми. Метод зазвичай базується на відсіченні афіксів (приставок, закінчень) без врахування контексту. Переваги: швидкість, не потребує глибокого аналізу або великих словників. Недоліки: Може призводити до неточностей, оскільки не враховує контекст або морфологічні зв'язки. Приклад:

- Українська мова: "граючи" → "гра"
- Англійська мова: "running" → "run"

Лематизація зводить слово до його словникової (базової) форми з використанням словників та морфологічного аналізу. Це більш комплексний процес, ніж стемінг, і враховує контекст. Переваги: висока точність, оскільки враховується морфологія та контекст слова. Недоліки: більш повільний, вимагає більших обчислювальних ресурсів та наявність великих морфологічних баз даних. Приклад:

- Українська мова: "пішли" → "йти"
- Англійська мова: "better" → "good"

Хоча стемінг та лематизація мають спільну мету зведення словоформ до базової форми, лематизація надає більш точний і надійний результат, оскільки вона враховує контекст та морфологічні особливості слова.

Наївний алгоритм Баєса – це метод класифікації, заснований на теоремі Баєса, що із "наївним" припущенням про незалежність між ознаками. За допомогою теореми Баєса алгоритм вчиться прогнозувати ймовірність того, що даний елемент належить до певного класу, на основі певних ознак (або характеристик). Формула теореми Баєса:

$$P\ (A|B) = \frac{P\ (B|A) * P(A)}{P(B)}.$$

де: $P(A|B)$ – апостеріорна ймовірність, $P(B|A)$ – ймовірність спостереження ознаки при умові, що гіпотеза вірна, $P(A)$ – апріорна ймовірність гіпотези, $P(B)$ – апріорна ймовірність спостереження ознаки.

Переваги:

1. Швидкість навчання та прогнозування.
2. Добре працює на великих наборах даних.
3. Простота у використанні.

Недоліки:

1. Наївне припущення про незалежність ознак може не завжди відображати реальність.
2. Може бути менш точним у порівнянні з іншими алгоритмами для деяких задач.

Приклад: Класифікація спаму. Маємо листи з відомими позначеннями "спам" або "не спам".

Для нового листа алгоритм розглядає слова в ньому і визначає ймовірність того, що лист є спамом, базуючись на тому, як часто ці слова зустрічались у спам-листах у навчальному наборі.

Наприклад, якщо слово "лотерея" часто зустрічається у спам-листах, то лист, що містить це слово, матиме більшу ймовірність бути визначеним як спам.

Наївний алгоритм Баєса є простим, але ефективним методом класифікації для ряду задач. Незважаючи на його "наївність", він часто виявляється дуже конкурентоспроможним у порівнянні з більш складними алгоритмами.

Вбудовування слів – це модельний спосіб представлення текстової інформації, де слова або фрази з корпусу тексту представлені векторами фіксованого розміру. Такі вектори розраховуються таким чином, щоб відображати семантичні відносини між словами: слова з схожими значеннями або контекстами мають більш схожі вектори. Мета – перетворити слова в числові вектори, зберігаючи при цьому семантичні відносини між ними.

За допомогою моделі Word2Vec отримуємо векторне представлення слова "король". Тоді, використовуючи алгебраїчні операції, можна виконати наступне: вектор("король") - вектор("чоловік") + вектор("жінка"). Отриманий результат, буде близьким до вектора слова "королева", відображаючи семантичні відносини між цими словами.

Вбудовування слів – ключова концепція в сучасних методах обробки природної мови. Завдяки здатності уявляти слова в якості векторів, які відображають їх семантичний зміст, моделі, які використовують вбудовування слів, можуть ефективно обробляти мовну інформацію для ряду завдань, від аналізу емоцій до машинного перекладу.

LSTM – це вид рекурентних нейронних мереж (RNN), спроектований так, щоб боротися з проблемами втрати інформації в традиційних RNN. LSTM добре підходить для класифікації, обробки та прогнозування часових рядів з великими проміжками.

Основна ідея: LSTM використовує спеціальні модулі, які дозволяють інформації "плинути" без перешкод. Ці модулі містять три важливі компоненти: вентиль забуття, вентиль введення та вентиль виведення. Ці вентилі разом допомагають LSTM вирішити, яку інформацію треба зберігати, оновлювати або викидати.

Приклад: Припустимо, створюєте систему перекладу машинної мови, яка перекладає речення з української мови на англійську. Речення можуть мати різну довжину та структуру. LSTM може бути корисним у цьому випадку, оскільки його рекурентна структура дозволяє йому обробляти послідовності даних (речення) і враховувати контекстуальну інформацію, що знаходиться як на початку, так і в кінці послідовності.

LSTM – це потужний тип рекурентної нейронної мережі, який здатний вивчати довготривалі залежності і використовується у багатьох сучасних завданнях обробки природної мови. Його специфічна структура робить його здатним зберігати, оновлювати та викидати інформацію ефективно, що робить його вибором номер один для багатьох завдань, які вимагають обробки послідовних даних.

Кожен із згаданих методів має свої переваги, важливо підкреслити, що використання конкретного методу або алгоритму залежить від конкретної задачі та наявних даних. Іноді найкращі результати досягаються шляхом комбінування кількох методів. Сучасні техніки обробки природної мови продовжують розвиватися, враховуючи виклики та потреби сучасного світу.

2.1 Алгоритм розпізнавання звукових сигналів

У світі, що оточує нас, звуки відіграють неймовірно важливу роль, надаючи нам інформацію про все, що відбувається навколо. Від дзвінка телефону до шуму лісу - кожний звуковий сигнал несе в собі інформацію. Алгоритм розпізнавання звукових сигналів — це ключовий елемент у цьому процесі.

У цьому дослідженні детально описано, як працюють сучасні алгоритми розпізнавання звукових сигналів, які типи задач можна вирішити за допомогою цих алгоритмів, а також описані виклики та перспективи розвитку в цій області. у дослідженні докладно розглянуто ключові концепції, методології та підходи, що стосуються алгоритмів розпізнавання звукових сигналів.

Звук — це вібраційна хвиля, яка поширюється через рідину, газ або тверде середовище. Для сприйняття звуку людиною йому потрібно рухатися повітрям або іншим матеріалом, що знаходиться в безпосередній близькості до нашого слухового апарату. Говорячи про аналіз або розпізнавання звукових сигналів у цифровому світі, мається на увазі представлення цих вібрацій у вигляді числових даних.

Звуковий сигнал — це відображення звукової хвилі в числовій формі. У цифровому аудіо, такі сигнали зазвичай представлені як послідовність чисел, що відображають величину амплітуди звуку в певний момент часу. Ця послідовність чисел потім може бути оброблена, проаналізована або відтворена для відновлення оригінального звуку.

При класифікації звуків важливим є належне врахування різноманітних джерел шуму. Це може включати фоновий шум, ехо, інтерференцію тощо. Якість запису, особливості акустичного середовища та інші зовнішні чинники також можуть впливати на процес класифікації.

Сучасні системи розпізнавання мови зазвичай базуються на глибокому навчанні та нейронних мережах, які виказали високу ефективність у вирішенні цієї задачі. Зокрема, рекурентні нейронні мережі (RNN) та трансформери стали популярними архітектурами для розпізнавання мови завдяки їх здатності враховувати контекстуальну інформацію у великих послідовностях даних.

Передобробка звукових даних є важливим етапом у процесі розпізнавання аудіоінформації. Вона включає в себе ряд методів та технік, які дозволяють перетворити первинний звуковий сигнал у форму, оптимальну для аналітичних завдань. Це може включати дії як нормалізація гучності, видалення шумів, фільтрація, а також застосування математичних перетворень, таких як перетворення Фур'є, щоб отримати частотний спектр сигналу. Правильно виконана перед обробка полегшує подальший аналіз та підвищує ефективність систем розпізнавання звукових сигналів.

Перетворення Фур'є — це математична операція, яка дозволяє аналізувати різні частотні компоненти звукового сигналу. Вона розбиває сигнал на суму синусоїд, кожна з яких має певну частоту, амплітуду та фазу.

Швидке перетворення Фур'є (FFT). Це алгоритм, який обчислює DFT ефективніше та швидше. Він використовується у більшості реалізацій, що аналізують звукові дані.

Спектрограма відображає частотний спектр сигналу як функцію часу. Спектрограми часто корисні для візуального виявлення подій у звуковому сигналі та для аналізу частотних характеристик звукових подій на протязі часу.

Віконний метод. Перед застосуванням DFT або FFT, звичайно, використовують віконну функцію (наприклад, вікно Гемінга) для розділення сигналу на короткі відрізки часу. Це допомагає зменшити вплив кінцевих точок на результуючий спектр.

Частотна роздільність. Важливо вибрати правильний розмір вікна при обчисленні DFT або FFT. Більше вікно дасть вам кращу частотну роздільність, але меншу часову роздільність, і навпаки.

Коли обробка звукового сигналу здійснюється, основною метою перетворення Фур'є є визначення частотних компонентів сигналу та їх інтенсивності. Це особливо корисно для розпізнавання мови, музичного аналізу, виявлення аномалій у звуках та багатьох інших застосувань.

Приклад використання перетворення Фур'є для аналізу звукового сигналу в Python з допомогою бібліотеки NumPy (рисунок 2.1). Генерація синтетичного

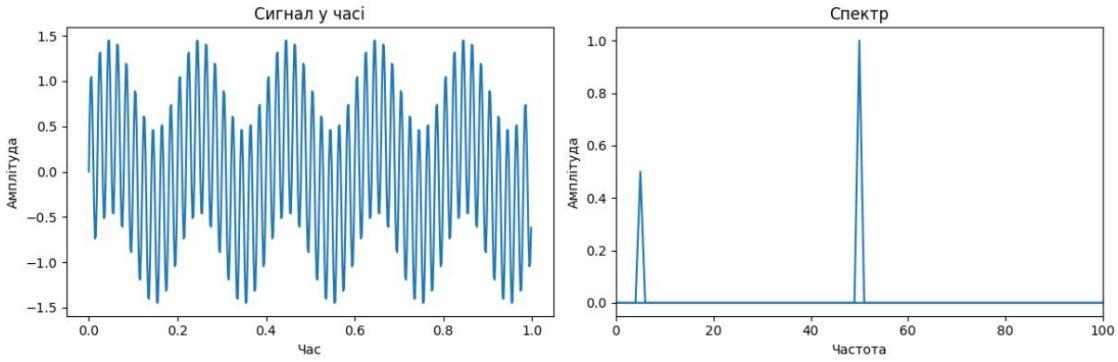
сигналу. Створюємо синусоїдний сигнал з двома частотами. Застосування FFT — обчислимо швидке перетворення Фур'є для нашого сигналу. Відображення спектра — побудова графіку амплітуди проти частоти для отриманого спектра.

```python
import numpy as np
import matplotlib.pyplot as plt

# Створення сигналу
fs = 500  # частота дискретизації
t = np.linspace(0, 1, fs, endpoint=False)  # часова вісь
f1, f2 = 5, 50  # частоти
y = 0.5 * np.sin(2 * np.pi * f1 * t) + np.sin(2 * np.pi * f2 * t)

# Перетворення Фур'є
yf = np.fft.fft(y)
xf = np.fft.fftfreq(t.size, t[1] - t[0])

# Графік сигналу у часі
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(t, y)
plt.title("Сигнал у часі")
plt.xlabel("Час")
plt.ylabel("Амплітуда")

# Графік спектра
plt.subplot(1, 2, 2)
plt.plot(xf, 2.0/fs * np.abs(yf))
plt.title("Спектр")
plt.xlabel("Частота")
plt.ylabel("Амплітуда")
plt.xlim(0, 100)  # обмеження вісі X для кращого відображення
plt.tight_layout()

plt.show()
```

Рисунок 2.1 – Перетворення звукових сигналів за допомогою Фур'є

В результаті отримаємо два графіки (рисунок 2.2): перший показує синтетичний сигнал в часі, а другий - частотний спектр цього сигналу. На спектровому графіку можна побачити дві піки, які відповідають двом частотам нашого синусоїдного сигналу (5 Hz та 50 Hz).

Графік "Сигнал у часі" показує, як амплітуда сигналу змінюється протягом певного проміжку часу. У даному випадку створюється композитний сигнал, що є сумою двох синусоїдних хвиль з різними частотами (5 Гц та 50 Гц). Отже, на графіку ви бачите суперпозицію двох хвиль: одна з них має велику амплітуду та низьку частоту (5 Гц), а інша має вищу частоту (50 Гц). Висота кожного піку вказує на амплітуду відповідної частоти в сигналі. Таким чином, аналізуючи частотний спектр, можна точно визначити складові синусоїди в композитному сигналі та їхні амплітуди.



а) б)

Рисунок 2.2 – Графік а) показують синтетичний сигнал в часі, графік б) показує частотний спектр

Графік "Спектр" показує результат перетворення Фур'є сигналу, яке перетворює сигнал з часової області в частотну область. Це допомагає визначити основні частоти, які присутні у сигналі.

На графіку спектра зображено два яскраво виражених піки. Один пік відповідає частоті 5 Гц, а інший пік — частоті 50 Гц. Ці піки підтверджують наші початкові частоти сигналу. Амплітуди цих піків вказують на інтенсивність кожної з цих частот у сигналі. Ці піки являють собою основну інформацію, яка допомагає нам розуміти, які частоти домінують у нашому сигналі.

Отже, спектральний аналіз сигналу, зокрема використання перетворення Фур'є, є дуже корисним інструментом для аналізу характеристик сигналу, зокрема його частотних складових.

Спектрограми стали незамінними в інструментах аналізу звуку, таких як розпізнавання мови, музичний аналіз, виявлення подій у звуковій доріжці тощо.

Приклад створення спектрограми в Python з використанням бібліотеки matplotlib можна побачити на рисунку 2.3.

Цей графік відображає частотний склад сигналу в залежності від часу, що робить видимими зміни в частотному складі сигналу протягом тривалості його відтворення

Створення спектрограми в Python включає використання різних бібліотек, таких як matplotlib та numpy.

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal

# Генеруємо тестовий сигнал
Fs = 1000
t = np.linspace(0, 1, Fs, endpoint=False)
x = np.sin(2 * np.pi * 7 * t) + 0.5 * np.sin(2 * np.pi * 13 * t)

# Обчислюємо спектрограму
frequencies, times, spectrogram = scipy.signal.spectrogram(x, fs=Fs)

# Візуалізуємо спектрограму
plt.pcolormesh(times, frequencies, 10 * np.log10(spectrogram), shading='gouraud')
plt.colorbar(label='Intensity [dB]')
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.title('Spectrogram')
plt.tight_layout()
plt.show()
```

Рисунок 2.3 – Створення спектрограми в Python

На результаті цього коду отримано графік спектрограми (рисунок 2.4), де по вертикальній осі показані частоти, по горизонтальній - час, а інтенсивність різних частот у різний час відображається різними кольорами (більш темний колір відповідає більшій інтенсивності).

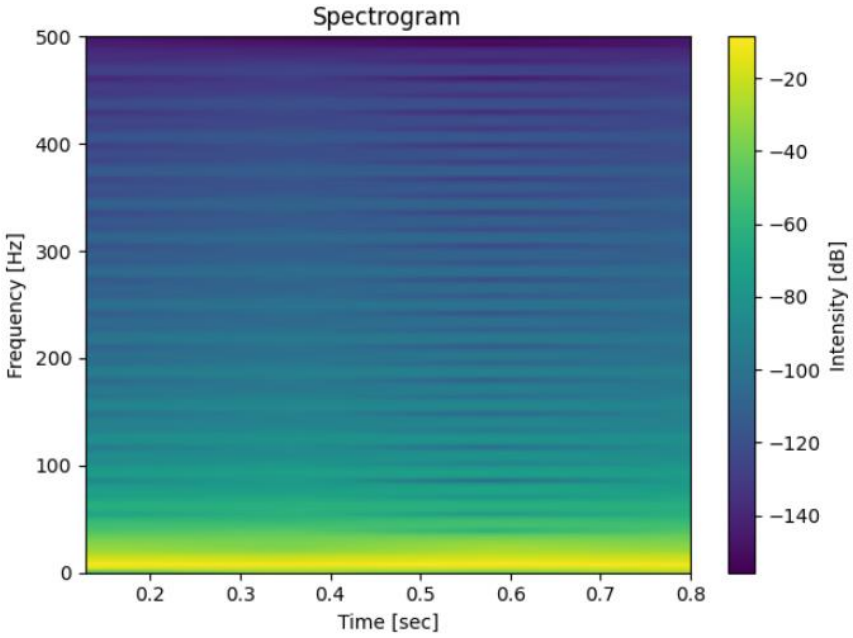


Рисунок 2.4 – Графік спектрограми відслідковування змін в частковому складі звукового сигналу протягом часу

Спектрограми є корисними, коли потрібно відслідкувати зміни в частотному складі звукового сигналу протягом часу, такі як переходи в музиці, мовні сигнали або інші звукові події.

Звукові дані часто супроводжуються шумами, такими як фонові звуки, говоріння інших людей, дорожній шум тощо. Це може ускладнити процес розпізнавання або знизити точність моделі. Хоча існує багато звукових даних, не всі з них мають анотації. Тренування моделей на невідомих даних може призвести до невірних результатів. Обробка та аналіз звукових даних, особливо великих датасетів, може вимагати значних обчислювальних ресурсів.

З розвитком технологій розпізнавання мови та звукових сигналів можливі застосування у сферах медицини (діагностика за звуками), безпеки (виявлення небезпечних звукових сигналів) та розваг (створення інтерактивних мультимедійних застосунків).

Розпізнавання звукових сигналів є ключовою частиною досліджень в галузі машинного навчання та обробки сигналів.. Основні виклики, такі як шум у даних, брак анотованих даних та висока вартість обчислень, підкреслюють складність завдань, але також вказують на величезний потенціал для подальших досліджень і покращень.

В умовах стрімкого розвитку інформаційних технологій розпізнавання звукових сигналів відіграє ключову роль, допомагаючи машинам краще розуміти навколишній світ та адаптуватися до потреб людства. Це напрямок, який, безумовно, продовжує рости та розвиватися, відкриваючи нові горизонти для наукових досліджень та практичних інновацій.

2.2 Структура особливості формування коду мовою Python

Мова програмування Python за замовчуванням здобула популярність серед розробників завдяки своєму гнучкому синтаксису, багатофункціональності та спільноті, що швидко росте. З особливістю написання коду, який легко читається,

Python відомий своєю здатністю спрощувати комплексні завдання, зокрема в областях обробки даних, штучного інтелекту та, зокрема, розпізнавання природної мови.

Розпізнавання природної мови (NLP) є однією з ключових областей досліджень у галузі комп'ютерних наук, яка присвячена обробці та аналізу мовних даних. Оскільки Python має широкий набір інструментів та бібліотек для NLP, це робить його відмінним вибором для розробки алгоритмів та систем, що здатні розуміти та генерувати природну мову.

У контексті цієї теми особливий інтерес представляє можливість перекладу текстових даних природної мови в конкретний і виконуваний код на Python. Такий підхід може знайти застосування в автоматизації написання скриптів, інтерактивних систем програмування або навіть в освітніх платформах для навчання програмуванню.

Python є динамічною, високорівневою мовою програмування, що пропонує чіткий синтаксис і семантику, сприяючи читабельності коду. Розглянемо декілька ключових концепцій та структурних елементів Python, які важливо знати при спробі перетворити природну мову в програмний код:

Python є однією з найбільш популярних мов програмування, частково завдяки широкому спектру бібліотек та фреймворків, які значно спрощують розробку програмного забезпечення. Ось деякі ключові бібліотеки та фреймворки, які є особливо корисними при роботі з проектами, що включають перетворення природної мови в код:

TensorFlow і PyTorch. Дві відомі бібліотеки для машинного навчання, які надають розробникам інструменти для створення, тренування та впровадження штучних нейронних мереж. Ці бібліотеки часто використовуються в задачах обробки природної мови та розпізнавання мови.

Natural Language Toolkit (NLTK) і spaCy — бібліотеки для роботи з людською мовою. Вони надають можливості для різних задач NLP, таких як токенізація, розпізнавання іменованих сутностей, тегування частин мови та синтаксичний аналіз.

Flask і Django — фреймворки для веб-розробки, що дозволяють швидко створювати веб-застосування. Flask є легким фреймворком, ідеальним для малих застосувань або мікросервісів, тоді як Django пропонує більш повноцінний набір функцій для розробки складних веб-застосувань.

Pandas і NumPy — ці бібліотеки надають розширені можливості для обробки та аналізу даних. Pandas спрощує роботу з табличними даними, тоді як NumPy є фундаментальною бібліотекою для наукових обчислень в Python, що надає підтримку великих масивів та матриць, разом із великою колекцією математичних функцій для їх опрацювання.

Процес перетворення тексту природної мови на виконуваний код є величезним викликом в галузі обробки природної мови. Це полягає в здатності комп'ютера розуміти та інтерпретувати природний текст так, щоб він міг бути перетворений на код, який може бути виконаний. Основні етапи цього процесу включають:

1. Аналіз тексту: Початковий текст, зазвичай, надається у форматі природної мови, аналізується з метою виявлення ключових інтентів та сутностей, які він містить.

2. Синтаксичний аналіз: Після ідентифікації ключових елементів тексту наступний крок полягає в тому, щоб розібрати його структурно, використовуючи дерева залежностей або інші методики для визначення зв'язків між словами та фразами.

3. Генерація коду: За допомогою інформації, отриманої з аналізу тексту та синтаксичного аналізу, система тоді намагається сформувати відповідний програмний код.

4. Валідація: Після генерації коду він перевіряється на коректність та функціональність.

5. Приклад: Нехай є речення на природній мові: "створи список із числами від 1 до 10". Ідеальний процес перетворення цього речення на код на Python був би таким:

1. Аналіз тексту виявить інтент "створити список" та сутності "числа", "від 1", "до 10".

2. Синтаксичний аналіз допоможе зрозуміти, що користувач хоче створити список, який починається з 1 і закінчується 10.

3. Генерація коду на основі отриманої інформації дасть наступний результат:

4. Валідація підтвердить, що сформований код є дійсним і відображає наміри користувача.

Цей процес, звісно, є значно складнішим для більш складних запитів та вимагає високої точності та глибокого розуміння контексту з боку системи.

Формування коду на основі аналізу природної мови, особливості:

- Динамічність. Завдяки невербальному характеру програмування, текст на природній мові може мати безліч варіантів висловлювання для того ж самого концепту або завдання. Система повинна розуміти різні способи формулювання одного й того ж питання.
- Контекст. У природній мові значення слова або фрази може змінюватися в залежності від контексту. Система повинна мати змогу розуміти цей контекст, щоб генерувати правильний код.
- Семантична точність. Система повинна точно визначати семантичний зміст запиту для перетворення його на відповідний код.

Нейромережеві методи можуть адаптуватися до нових структур та виразів, які не були явно вказані під час навчання. Щоб досягти високої точності, нейромережі потребують великих наборів даних для навчання. Навчання глибоких нейромереж вимагає значних обчислювальних ресурсів. Ці методи вже застосовуються в таких системах, як Google Translate та інших автоматичних перекладаць.

Використання глибокої нейромережі для перекладу фрази "додайте два числа разом" в Python код. Після тренування на великому наборі даних, де фрази природної мови відповідають відповідним кодам, модель може перекласти цю фразу в:

```
def add_two_numbers(a, b):
    return a + b
```

Цей приклад демонструє, як модель може розуміти контекст фрази та перетворювати її на відповідний код.

Обидва методи мають свої переваги та обмеження. Вибір методу залежить від конкретних завдань, доступних даних та ресурсів.

Вхід: "Функція, яка отримує список чисел і повертає їх середнє арифметичне." Вихід:

```
def average(numbers):
    return sum(numbers) / len(numbers)
```

Для створення власних систем, ви можете використовувати попередньо натреновані моделі та бібліотеки, такі як Transformers від Hugging Face. Вони надають широкий спектр попередньо навчених моделей для завдань NLP, які можна додатково налаштовувати для конкретних завдань. Приклад:

Завантажуємо модель і токенізатор:

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("gpt2-medium")
model = GPT2LMHeadModel.from_pretrained("gpt2-medium")
```

Функція для генерації коду:

```
def generate_code(prompt):
    inputs = tokenizer.encode(prompt, return_tensors="pt")
    outputs = model.generate(inputs, max_length=150, num_return_sequences=5,
temperature=0.7)

    for i, output in enumerate(outputs):
        decoded = tokenizer.decode(output, skip_special_tokens=True)
        print(f"Generated Code {i + 1}:\n{decoded}\n")

prompt = "Write a Python function to find the maximum of three numbers."
generate_code(prompt)
```

Це дозволяє створити систему на основі відкритих джерел, яка може перетворювати природну мову в код.

Розпізнавання природної мови та її перетворення на програмний код на Python є перспективним та динамічно розвиваючимся напрямком. Головною його

метою є автоматизація процесу програмування та підвищення доступності кодування для широкого кола людей, які можуть не мати глибоких знань у мовах програмування. Основні виклики, пов'язані з цією областю, включають амбівалентність природної мови, відсутність контексту та обмеження поточних методів. Однак, завдяки нейромережевим і символьним методам, можливість точної трансляції природної мови в код поступово збільшується.

Зростання кількості даних, вдосконалення моделей машинного навчання та розвиток технологій зроблять системи розпізнавання природної мови ще більш ефективними.

Таким чином, розпізнавання природної мови в контексті формування програмного коду на Python відкриває нові горизонти для розробки програмного забезпечення, роблячи цей процес більш інтуїтивним та демократичним.

2.3 Алгоритми генерації програмного коду та основи розпізнавання природної мови

Алгоритми генерації програмного коду спрямовані на автоматизацію процесу створення коду, дозволяючи розробникам вказувати завдання на високому рівні, які мають бути виконані, замість того, щоб детально прописувати кожний крок.

Шаблонне програмування — цей підхід базується на використанні готових шаблонів коду, які динамічно заповнюються або модифікуються відповідно до специфічних вимог. Приклад. Вхід: "Створити функцію, яка приймає два числа і повертає їх суму." Вивід:

```
def add_numbers(a, b):
    return a + b
```

Програмування на основі правил. В цьому підході використовуються переосмислення правила для генерації коду. Наприклад, "якщо користувач каже 'створити функцію', то генеруємо базовий шаблон функції".

Компіляція вищого рівня. Ідея полягає в тому, щоб використовувати мову вищого рівня для опису програми, а потім автоматично перекладати її на мову програмування. Приклад. Вхід: Візуальна блокова структура, яка представляє додавання двох чисел. Вивід:

```
def add(x, y):
    return x + y
```

Останнім часом з'явилася можливість використовувати глибокі нейронні мережі, особливо послідовні моделі типу Transformer, для генерації коду на основі введеного тексту. Приклад. Вхід: "Функція для знаходження максимального елемента в списку." Вивід:

```
def find_max(lst):
    return max(lst)
```

В автоматизованому світі програмування важливо мати засоби, які допомагають перетворювати природну мову на відповідний код програми. Це не тільки полегшує життя розробників, але й дозволяє непрофесіоналам у галузі IT створювати програми, використовуючи природні мовні запити. Описаний нижче алгоритм демонструє процес автоматичного перекладу англійської мови на мову програмування. Створено блок-схему, яка крок за кроком показує, як введений текст обробляється та перетворюється на відповідний код.

Опис алгоритму автоматичного перекладу англійської мови на мову програмування:

1. Введення тексту

На цьому етапі користувач вводить рядок тексту, який потрібно перетворити на програмний код. Наприклад: "зробити функцію, яка друкує 'Привіт, Світ!'".

2. Попередня обробка тексту. Цей етап включає в себе три підкроки:

- Токенізація - розбиття введеного тексту на окремі слова або фрази (токени).
- Лематизація - конвертація слів до їхньої кореневої форми. Наприклад, "друкує" може бути змінено на "друк".
- Видалення стоп-слів - видалення слів, які є зайвими для аналізу. Зазвичай це загальновживані слова, як "та", "або", "і" тощо.

3. Семантичний аналіз. На цьому етапі визначається загальний зміст тексту:

- Визначення контексту - з'ясування основної ідеї або завдання, яке користувач хоче виконати.
- Отримання семантичних відображень - конвертація слів у числове представлення, яке відображає їхнє значення в контексті.

4. Кодування тексту. Текст конвертується в проміжне представлення за допомогою кодера. Це може бути вектор або інша структура даних, яка відображає семантику введеного тексту.

5. Генерація коду. На основі проміжного представлення створюється код. Тут використовуються різні механізми, такі як декодери та механізми уваги, для точного відображення семантики тексту у коді.

6. Валідація коду. Згенерований код перевіряється на наявність помилок. Логічна перевірка - виявлення помилок у логіці коду. Синтаксична перевірка - виявлення помилок у синтаксисі коду.

7. Оптимізація коду. Код оптимізується для поліпшення його продуктивності та читабельності. Наприклад, видалення зайвих змінних або спрощення логіки.

8. Умовні перевірки. Після оптимізації коду він знову перевіряється. Якщо код не відповідає вимогам або містить помилки, процес може повертатися до попередніх кроків.

9. Виведення результату. Код представляється користувачеві. Користувач може прийняти цей код або внести зміни та запустити процес знову.

Цей алгоритм являє собою послідовність кроків, спрямованих на перетворення тексту на програмний код, з урахуванням семантики та контексту введеного тексту.

2.4 Висновки до розділу

В розділі 2.1 досліджено алгоритм розпізнавання звукових сигналів, який є важливою частиною перетворення голосових команд або діалогів на текстовий формат. Цей процес вимагає точного виявлення та інтерпретації звукових хвиль для відтворення природної мови.

Інтеграція технологій розпізнавання звукових сигналів із сучасними методами програмування відкриває нові можливості для розробки голосових асистентів і інших застосунків, які залучають природну мову. Додатково, ефективність обробки звуку може суттєво впливати на точність розпізнавання і, відповідно, на якість користувацького досвіду.

В розділі 2.2 зосереджено увагу на структурі особливостей формування коду мовою Python. Python є однією з найбільш популярних мов програмування, завдяки своїй читабельності та гнучкості, і знання його структурних особливостей є важливим для ефективного перетворення природної мови на програмний код.

У розділі 2.3 наголошено на алгоритмах генерації програмного коду та основах розпізнавання природної мови. Тут було розглянуто, як сучасні алгоритми можуть перетворювати високо рівневі інструкції на виконуваний код, а також ключові принципи роботи з текстовими даними у сфері NLP.

Зростаюча увага до NLP та генерації коду свідчить про те, що людство стоїть на порозі нової ери у розробці програмного забезпечення, де машини будуть активними партнерами людей у процесі створення програм. Враховуючи швидкість розвитку та досліджень в цій області, можна очікувати ще більшої інтеграції та синергії між розпізнаванням природних мов і процесами програмування в найближчому майбутньому.

В цілому, цей розділ підкреслює значущість і глибину дослідження в області розпізнавання природних мов, а також потенціал цих технологій у галузі автоматизації програмування.

3 ПРОГРАМНИЙ ДОДАТОК ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ НА ОСНОВІ РОЗПІЗНАВАННЯ ПРИРОДНОЇ МОВИ

3.1 Структура, інтерфейс моделювання програмного додатку

У цьому розділі представлена структура інтерфейсу програмного додатку – “Helpek”, який зосереджений на генерації програмного коду на основі розпізнавання природної мови. На Рисунку 3.1 представлено алгоритм взаємодії користувача з програмним додатком "Helpek". Алгоритм ілюструє послідовність кроків від моменту введення даних користувачем через інтерфейс до обробки введення за допомогою технологій NLP (Natural Language Processing) та подальшого управління завданнями, що передбачає взаємодію з базою даних.

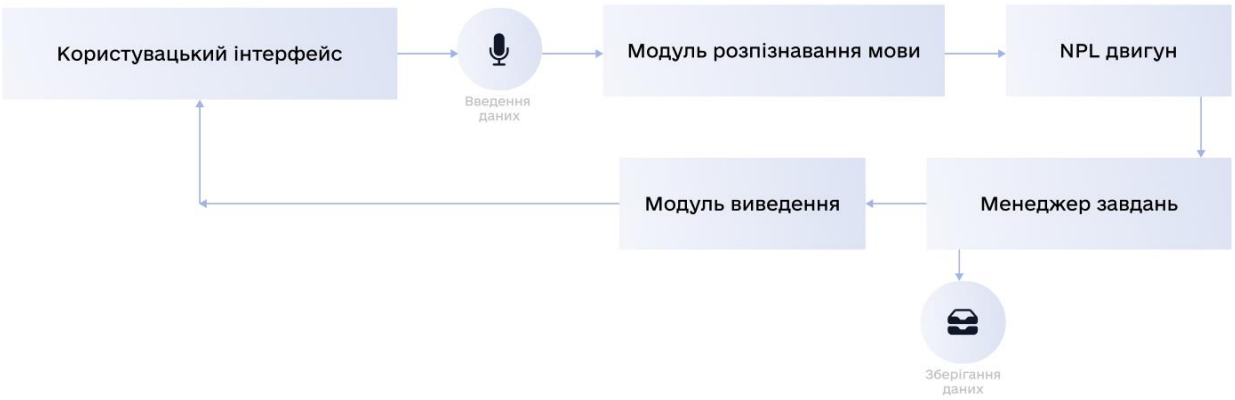


Рисунок 3.1 – Послідовністькроків користувача з програмним додатком "Helpek"

Процес починається з "Користувацького інтерфейсу", де користувач вводить дані за допомогою голосових команд. Наступний крок — "Модуль розпізнавання мови", який трансформує голосові дані у текстову форму. Після цього "NLP двигун" аналізує текстові дані, розуміє інтенції та контекст введення користувача. Потім "Модуль виведення" формує відповідь або здійснює дії, засновані на аналізі NLP двигуна. "Менеджер завдань" відповідає за виконання та управління завданнями, які можуть включати зберігання даних, їх обробку або інші пов'язані операції.

“Helpek” розроблено з використанням мови програмування Dart та фреймворку Flutter, що забезпечує багатий та інтуїтивно зрозумілий користувацький інтерфейс.

Dart — це сучасна, об'єктно-орієнтована мова програмування, розроблена компанією Google. Вона була створена з метою забезпечення високої продуктивності, ефективності та легкості у використанні для розробки мобільних, веб- та серверних додатків. Dart підтримує об'єктно-орієнтоване програмування, забезпечуючи такі можливості, як класи, інтерфейси та змішування. Також використовує сильну типізацію, що допомагає попередити помилки під час компіляції та спрощує розуміння коду. Dart може виконувати код швидко завдяки JIT (Just-In-Time) компіляції у режимі розробника та AOT (Ahead-Of-Time) компіляції для виробництва, забезпечуючи оптимальну продуктивність. Підтримує асинхронне програмування з використанням futures та streams, що дозволяє ефективно виконувати багатозадачність. Dart використовується як для клієнтських (наприклад, веб- та мобільних додатків), так і для серверних додатків.

Flutter — це відкритий UI-фреймворк від Google, який дозволяє створювати візуально привабливі, компільовані нативно додатки з однієї кодової бази для мобільних (iOS і Android), веб- та десктопних платформ. Гаряче перезавантаження (Hot Reload) дозволяє миттєво бачити результати змін у коді без перезапуску додатку. Використовуючи двигун Skia для рендерингу, Flutter дозволяє створювати гладкі та високопродуктивні інтерфейси. Багатий набір вбудованих віджетів та легкість у створенні власних віджетів. Flutter дозволяє розробникам описувати інтерфейси в декларативному стилі, що спрощує процес розробки. Один код для всіх платформ знижує витрати на розробку та підтримку.

Dart забезпечує швидке виконання та продуктивність, а Flutter додає потужні засоби для створення інтерфейсу, роблячи комбінацію цих двох технологій ідеальною для сучасної розробки додатків.

Інтерфейс програмного додатку ілюструє візуалізацію коду, написаного з використанням Flutter та Dart. Інтерфейс показує велике, централізоване діалогове вікно з привітним повідомленням "Привіт. Чим можу допомогти?" та іконкою мікрофона для голосового вводу команд.

Відповідно до коду в Додатку Б, в інтерфейсі можна впізнати наступні елементи:

Контейнери та Форми. Великі прямокутні контейнери, які ймовірно представляють основну частину інтерфейсу з закругленими кутами, де відображається текстове повідомлення та інші елементи.

Текстове повідомлення. Великий та чіткий шрифт для звернення до користувача, що спонукає до взаємодії з додатком через голосові команди.

Кнопка мікрофона. Центрально розміщена іконка мікрофона, що використовує OvalBorder для свого круглого дизайну, стимулюючи користувача активувати голосовий ввід. Запис голосового вводу користувача відбувається через мікрофон (рисунок 3.2 червоним квадратом виділений мікрофон, через який відбувається цей запис) на протязі визначеного періоду (5 секунд) та перетворює аудіо в текст.

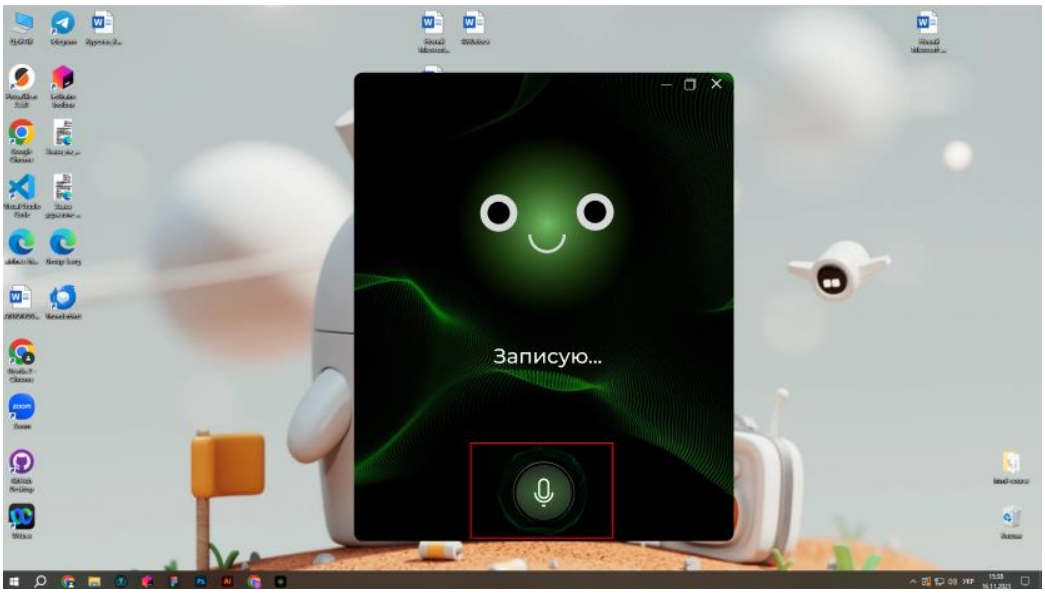


Рисунок 3.2 – Активний елемент активації голосового вводу

Градієнти та візуальні ефекти. Градієнтні ефекти та кольорові переходи, які додають глибину та текстуру фону додатку, можливо виконані за допомогою віджетів Container з ShapeDecoration.

Цей інтерфейс відображає чистоту та сучасний дизайн, що сприяє зручності використання додатку. Візуальні елементи та стиль оформлення відповідають сучасним стандартам UI/UX дизайну, де чіткість, інтуїтивність та естетична привабливість є ключовими.

На Рисунку 3.3 представлено інтерфейс програмного додатку, який відображає згенерований код Python у відповідь на запит користувача.

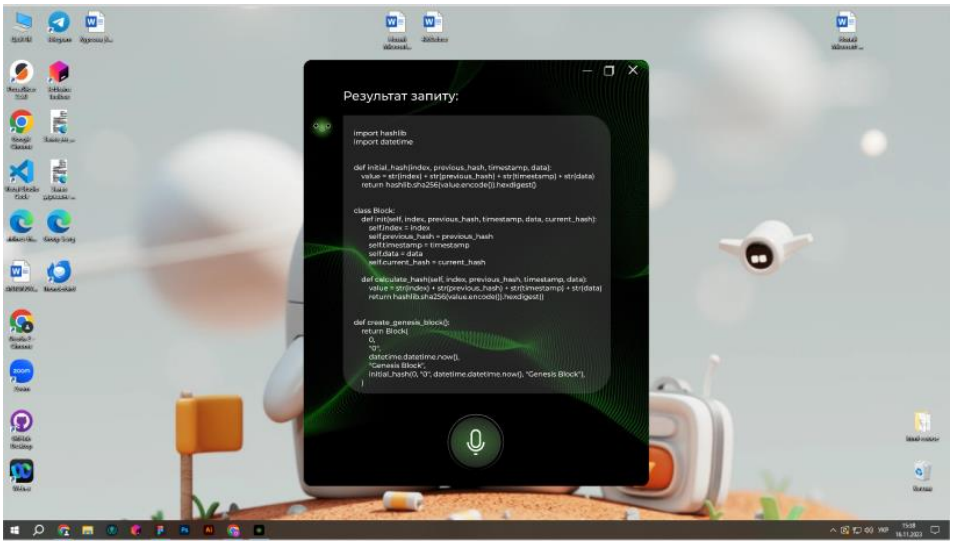


Рисунок 3.3 – Результат обробки запиту користувача

Аналізуючи представлений інтерфейс та згенерований код на основі запиту користувача, можна зробити висновок, що програмний додаток ефективно виконує функцію генерації програмного коду з використанням розпізнавання природної мови. “Helpek” інтегрує голосові команди користувача з функціональністю штучного інтелекту, що дозволяє створювати структурований і функціональний код на мові програмування Python. Це вказує на високий рівень розуміння контексту користувацьких запитів та здатність адекватно реагувати на них з відповідним програмним рішенням.

Інтерфейс “Helpek” забезпечує чітке візуальне представлення результатів, що сприяє легкому розумінню та використанню згенерованого коду. Використання Flutter та Dart дозволяє створити гнучкий і відповідний інтерфейс, який забезпечує плавну взаємодію між користувачем і програмою.

З огляду на ці результати, можна стверджувати, що “Helpek” є цінним інструментом для користувачів, які прагнуть швидко перетворити свої ідеї на робочий програмний код, та може бути особливо корисним для освітніх цілей, автоматизації рутинних задач, або як засіб для новачків, що вивчають основи програмування.

3.2 Програмні Модулі БД Програмного Додатку

Програмні модулі бази даних (БД) даного програмного додатку розроблені для забезпечення ефективної інтеграції розпізнавання мови, синтезу мовлення та можливостей штучного інтелекту (AI), щоб забезпечити точну обробку та відображення введеного тексту. Основний код додатку інтегровано з потужними бібліотеками Python, такими як pyautogui для автоматизації інтерфейсу користувача, speech_recognition для перетворення голосу в текст, gtts для перетворення тексту в мовлення, та openai для залучення передових AI-моделей для обробки мовних запитів.

Алгоритм на Рисунку 3.4 демонструє послідовність дій, які описують взаємодію користувача з передовою системою розпізнавання та обробки природної мови. Використання спеціалізованих зовнішніх API дозволяє ефективно перетворювати голосові команди в текстовий формат, а потім - у зрозуміле мовлення. Весь процес включає транскрибування аудіо, його семантичний аналіз з подальшою генерацією відповіді через віртуального агента Dialogflow, що дозволяє системі відповідати на запити користувача в реальному часі, створюючи враження безперервної діалогової взаємодії.

Рисунок 3.4 – Послідовність дій взаємодії користувача з системою обробки природної мови

Код із рисунку 3.5 виступає як основа для обговорення ключових компонентів системи, що включає в себе інтеграцію з розширеними бібліотеками Python для взаємодії з користувачем на рівні мови.

Код починається з імпорту бібліотек pyautogui, speech_recognition (sr), та gTTS, які відповідають за автоматизацію графічного інтерфейсу користувача, розпізнавання мови та перетворення тексту в мовлення відповідно. Використання OpenAI для залучення можливостей штучного інтелекту до нашої системи.

```python
import pyautogui
import speech_recognition as sr
from gtts import gTTS

from openai import OpenAI


OPENAPI_KEY = "***"
client = OpenAI(api_key=OPENAPI_KEY)

r = sr.Recognizer()

with sr.Microphone() as source:
    # read the audio data from the default microphone
    audio_data = r.record(source, duration=5)
    print("Розпізнавання мови...")
    # convert speech to text
    text = r.recognize_google(audio_data, language="uk-UA")

    print(text)
    tts = gTTS(text, lang="uk")
    tts.save("check.mp3")

    response = client.chat.completions.create(
        messages=[
            {
                "role": "user",
                "content": text,
            }
        ],
        model="gpt-3.5-turbo",
    )
    print(vars(response))

    for choice in response.choices:
        pyautogui.typewrite(choice.message.content)
```

Рисунок 3.5 – Інтеграція з бібліотеками pyautogui, speech_recognition, gtts та OpenAI

Програмний додаток використовує мікрофон як вхідний пристрій для захоплення аудіоданих, що розпізнаються та перетворюються у текстову форму з подальшим аналізом відповіді за допомогою AI. Виведений текст text озвучується за допомогою gTTS і зберігається у файлі, тоді як отримана відповідь з OpenAI вводиться у систему через pyautogui.

Цей програмний додаток, розроблений на мові програмування Python, представляє собою інноваційне рішення, що інтегрує розпізнавання мови, штучний інтелект (AI) та автоматизацію для створення багатофункціонального інструменту. Основна мета додатку - перетворювати голосові команди користувачів на текст, обробляти цей текст за допомогою AI та виконувати різноманітні дії, як-от введення тексту в інші програми. Нижче наведено детальний огляд ключових компонентів і функцій додатку:

1. Розпізнавання Мови:

- Використовується бібліотека speech_recognition для перетворення аудіо (голосу користувача) на текст.
- Додаток слухає користувача через мікрофон на протязі визначеного періоду (наприклад, 5 секунд) та перетворює аудіо в текст.

2. Синтез Мовлення:

- За допомогою gTTS (Google Text-to-Speech) текст, отриманий з аудіо, може бути перетворений назад у мовлення.
- Це дозволяє додатку не тільки розуміти користувачів, але й комунікувати з ними.

3. Інтеграція з Штучним Інтелектом:

- Використання openai для інтеграції з моделями GPT.
- Текст, отриманий від користувача, може бути поданий як вхідні дані для моделей AI, які можуть генерувати відповіді або виконувати специфічні завдання на основі цього вводу.

4. Автоматизація Інтерфейсу:

- Pyautogui використовується для автоматичного введення тексту, отриманого від моделі AI, в інші програми або інтерфейси.
- Ця функціональність розширює можливості додатку, дозволяючи взаємодіяти не тільки з користувачем, але й з іншими програмними системами.

5. Інтерфейс Користувача:

- Хоча основна функціональність додатку зосереджена на взаємодії з голосом, інтерфейс користувача може включати візуальні елементи для відображення статусу, налаштувань та інших важливих інформаційних повідомлень.

Цей додаток може бути використаний у різних сценаріях: підтримка розробників, автоматизація написання коду або документації; допомога у вивченні мов, підготовці; підтримка людей з обмеженими можливостями, надання засобів для спілкування або виконання завдань за допомогою голосових команд.

Цей програмний додаток демонструє потужність інтеграції сучасних технологій розпізнавання мови, штучного інтелекту та автоматизації. Він пропонує

інноваційний підхід до взаємодії з користувачами та автоматизації різноманітних процесів, відкриваючи нові можливості для різних застосувань.

Структура бази даних — це спосіб організації даних у базі даних. Вона визначає, як дані зберігаються, пов'язані між собою та як вони можуть бути доступні та управляються. Структура бази даних складається з різних елементів, таких як таблиці, поля, записи, стовпці та рядки, а також відносини між цими елементами. У даному додатку структура така:

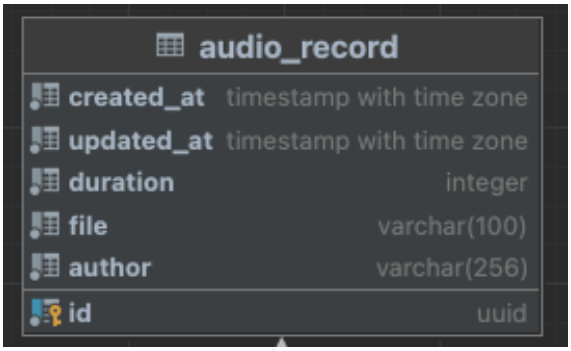


Рисунок 3.6 – Структура таблиці “аудіозаписів”

Таблиця аудіозаписів (рисунок 3.6) зберігає інформацію про кожен запис аудіо, зроблений через мікрофон. Поля: ідентифікатор запису (унікальний); часова мітка запису; довжина запису (наприклад, 5 секунд); посилання на аудіофайл.

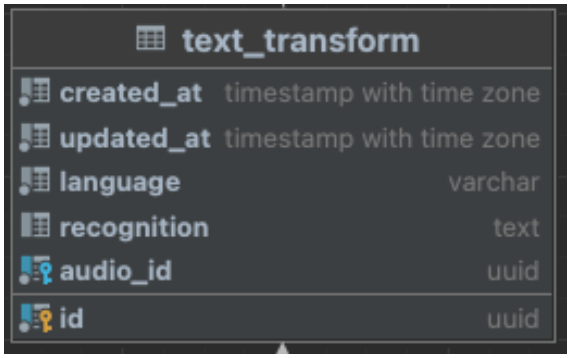


Рисунок 3.7 – Структура таблиці “текстових перетворень”

Таблиця текстових перетворень (рисунок 3.7) зберігає текст, отриманий після перетворення аудіо в текст. Поля: ідентифікатор перетворення; ідентифікатор пов'язаного аудіо запису; текстовий вміст; мова тексту (наприклад, "uk-UA").

Рисунок 3.8 – Структура таблиці “відповідей AI”

Записи відповідей, отриманих від штучного інтелекту OpenAI (рисунок 3.8). Поля: ідентифікатор відповіді; ідентифікатор пов'язаного текстового перетворення; текст відповіді AI; деталі моделі AI (наприклад, "gpt-3.5-turbo").

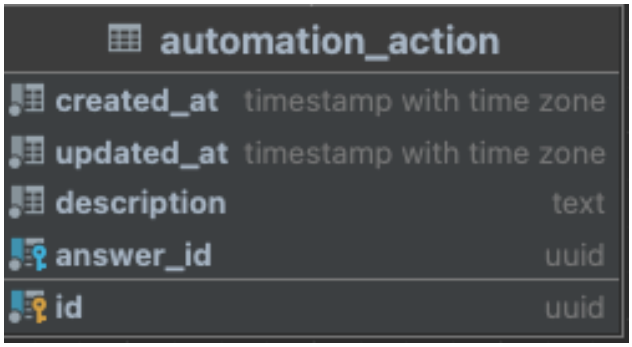


Рисунок 3.9 – Таблиця дій автоматизації інтерфейсу

Таблиця дій автоматизації інтерфейсу(Рисунок 3.9) зберігає інформацію про дії, виконані через pyautogui. Поля: ідентифікатор дії; ідентифікатор пов'язаної відповіді AI; опис дії (наприклад, введення тексту); часова мітка дії.

Кожна з цих таблиць забезпечує зберігання специфічних для своєї області даних, дозволяючи легко відстежувати та аналізувати процеси, що відбуваються в додатку. Це допомагає в аналізі точності перетворення мови в текст, оцінці відповідей AI, а також у відстеженні ефективності автоматизації інтерфейсу. Крім того, така структура бази даних дозволяє легко зв'язати дані з різних етапів обробки, від запису аудіо до виконання кінцевих дій.

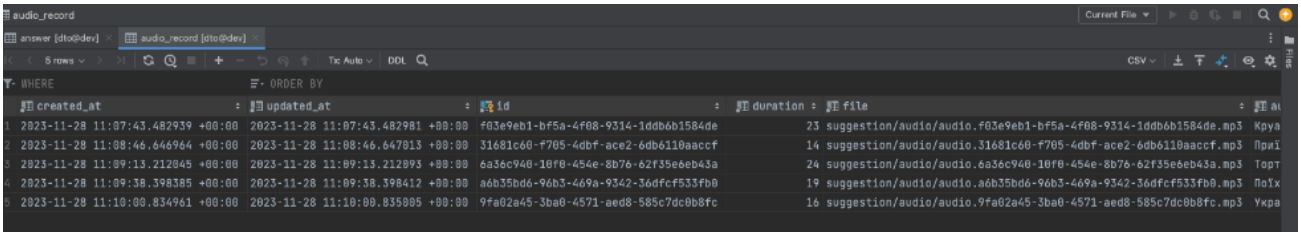


Рисунок 3.10 – Вміст таблиці audio_record

На рисунку 3.10 інтерфейс користувача бази даних. Ліва частина екрану відображає ієрархію проекту з різними таблицями та схемами. Виділена папку "suggestion", що містить дані, пов'язані з функціоналом пропозицій або рекомендацій.

Центральна частина екрану відображає деталізований вигляд таблиці "audio_record" із кількома стовпцями, такими як "id", "created_at" (дата створення), "duration" (тривалість), та "file" (файл). Ці дані відносяться до записів аудіо, що зберігаються в базі даних. Записи містять таймстемпи, що вказують на час створення записів, їхню тривалість та пов'язані з ними аудіофайли.

Програмні модулі в контексті комп'ютерних систем та додатків — це окремі, незалежні блоки, які разом формують більш складну програмну структуру. Кожен модуль відповідає за певний функціонал або аспект програми, працюючи як окремий компонент, який може бути розроблений, тестований та налагоджений незалежно від інших частин системи. Ось докладніший опис ключових аспектів програмних модулів:

- Функціональна автономність: Кожен модуль зазвичай виконує одну конкретну функцію або набір тісно пов'язаних завдань. Це дозволяє модулю бути самодостатнім та спрощує розуміння та управління кодом.
- Інтерфейси та взаємодія: Модулі взаємодіють один з одним через визначені інтерфейси, які встановлюють правила та методи комунікації між модулями.
- Масштабованість та гнучкість: Завдяки модульній архітектурі, програмні системи легше масштабувати та адаптувати. Нові функції можуть бути додані як нові модулі, а існуючі модулі можна модифікувати або замінювати без необхідності переписувати всю програму.

- Легкість тестування та налагодження: Оскільки кожен модуль відповідає лише за певну функціональність, тестування та налагодження можуть здійснюватися на рівні окремих модулів. Це спрощує виявлення та виправлення помилок.
- Пере використання коду: Модулі можуть бути розроблені таким чином, щоб їх можна було легко пере використовувати в інших програмах або проектах, що зменшує потребу в повторній розробці функціоналу.
- Організація коду: Модульна архітектура допомагає утримувати код організованим та читабельним, розділяючи програму на логічно структуровані частини.

Програмні модулі є ключовим елементом у створенні гнучких, ефективних та легко підтримуваних програмних систем. Вони дозволяють розробникам більш ефективно управляти складними проектами та адаптувати програмне забезпечення до змінюваних вимог та потреб.

Взаємодія модулів у контексті програмного додатку є критичним аспектом, який визначає, як різні незалежні компоненти (модулі) системи спілкуються та працюють разом для досягнення загальної мети або функціональності.

Детальний опис взаємодії цих модулів:

1. Модуль Розпізнавання Мови (speech_recognition).

Цей модуль активується відразу після запуску програми. Використовує мікрофон як джерело аудіо (за допомогою sr.Microphone) та записує аудіо протягом певного часу (у даному випадку, 5 секунд). Після запису аудіо, модуль перетворює мову в текст за допомогою Google Speech Recognition (r.recognize_google).

2. Модуль Синтезу Мовлення (gTTS):

Після того, як мова була перетворена на текст, цей модуль використовує отриманий текст для створення аудіофайлу за допомогою Google Text-to-Speech.

3. Модуль Штучного Інтелекту (OpenAI):

Програма передає текст, отриманий від модуля розпізнавання мови, до моделі штучного інтелекту OpenAI. Модель AI генерує відповідь на основі введеного тексту (у даному випадку, використовується модель "gpt-3.5-turbo").

4. Модуль Автоматизації Інтерфейсу (pyautogui):

Після отримання відповіді від AI, програма використовує pyautogui для автоматичного введення тексту відповіді AI. Це дозволяє програмі взаємодіяти з іншими програмами або інтерфейсами, автоматично вводячи текст.

Початковий модуль (speech_recognition) ініціює процес, записуючи мову та перетворюючи її в текст. Цей текст передається до наступного модуля (gTTS), який створює аудіофайл. Текст також передається до модуля штучного інтелекту (OpenAI), який обробляє його та генерує відповідь. Відповідь від AI використовується модулем автоматизації інтерфейсу (pyautogui), який вводить її в потрібне місце.

Кожен модуль має свою специфічну роль і взаємодіє з іншими модулями через передачу даних та виклики методів. Така взаємодія забезпечує плавний процес обробки від аудіо до тексту, від тексту до AI-відповіді, і надалі до автоматизованого введення цієї відповіді.

3.3 Результати тестування програмного засобу

Тестування включало аналіз різних аспектів роботи програми, таких як здатність правильно інтерпретувати голосові команди користувача, точність генерації програмного коду, а також загальну функціональність та інтуїтивність інтерфейсу користувача. Ретельний аналіз проводився на основі серії тестових запитів, які імітували реальне використання програми. Це дозволило оцінити здатність програмного засобу адекватно реагувати на зрозумілі, неоднозначні та неповні команди.

Важливим аспектом тестування було також визначення меж функціональності програми та ідентифікація потенційних областей для подальшого вдосконалення. Результати тестування надають цінну інформацію щодо ефективності "Helpek" у контексті автоматизованої генерації програмного коду та

відкривають перспективи для її практичного застосування у різних сферах програмування.

Один із таких запитів був сформульований як "Створи калькулятор". На основі цього запиту, система Helpek згенерувала код простого калькулятора на мові програмування Python. Програма також надає користувачу можливість вибору операції та вводу двох чисел для обчислення.

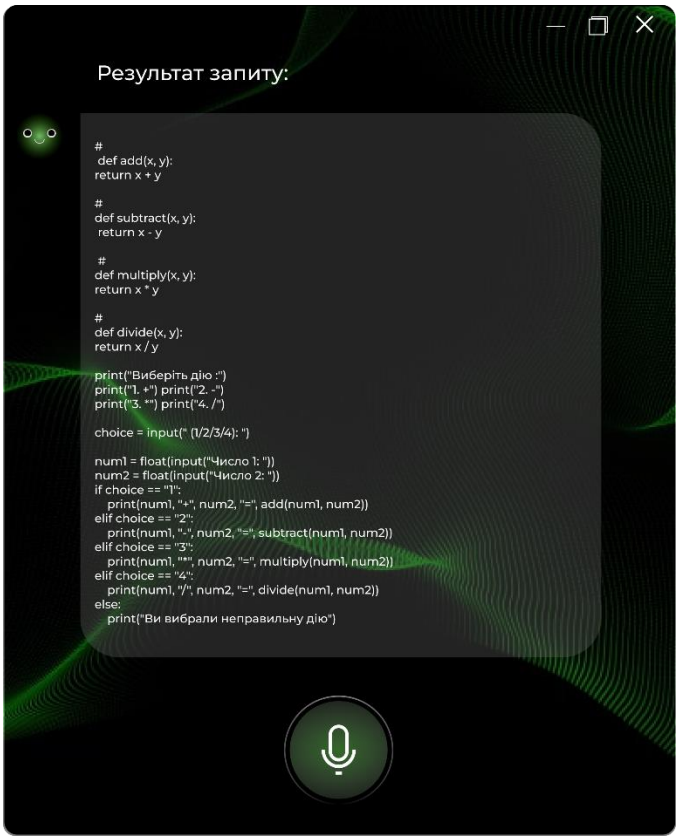


Рисунок 3.11 – Результат на запит користувача "Створи калькулятор"

Аналіз згенерованого коду:

- Користувачу пропонується вибрати тип операції за допомогою текстового меню.
- Після вибору операції, програма запитує у користувача числа для обчислення.
- В залежності від вибору, викликається відповідна функція, і результат виводиться

- Програма демонструє високий рівень гнучкості, адаптуючись до різних формулювань запитів користувачів. Така здатність вказує на ефективне застосування алгоритмів обробки природної мови.
- Реалізація коду показує, що система здатна розрізняти та правильно обробляти ключові концепти, необхідні для створення специфічного програмного рішення.
- Згенерований код має чітку та зрозумілу структуру, що свідчить про високу якість автоматизованої генерації коду та здатність системи адекватно інтерпретувати запити користувача.на екран.

Висновки з тестування:

Згенерований код підтвердив здатність системи Helpek ефективно перетворювати мовні запити користувачів на функціональний програмний код.

Програма виявилася працездатною та здатною виконувати задані користувачем математичні обчислення, що свідчить про високу точність розпізнавання та обробки природної мови.

Інтерфейс програми виявився інтуїтивно зрозумілим, забезпечуючи зручність використання для користувачів різного рівня. У випадку вибору неіснуючої операції, програма виводить повідомлення про помилку.

У процесі тестування програмного додатку Helpek були ідентифіковані випадки, коли, навіть за умови нечітко сформульованого запиту користувача, система здатна була зрозуміти суть запитання та згенерувати відповідний код.

Успішний нечіткий запит: "Створити щось для відстеження часу". Цей запит не містить конкретних технічних вказівок чи специфікацій, але Helpek успішно інтерпретував його та згенерував код простого таймера (рисунок 3.12).

Ці висновки свідчать про потенційну корисність та широкий спектр застосувань програмного додатку Helpek в області автоматизованої генерації програмного коду на основі мовних запитів користувачів.

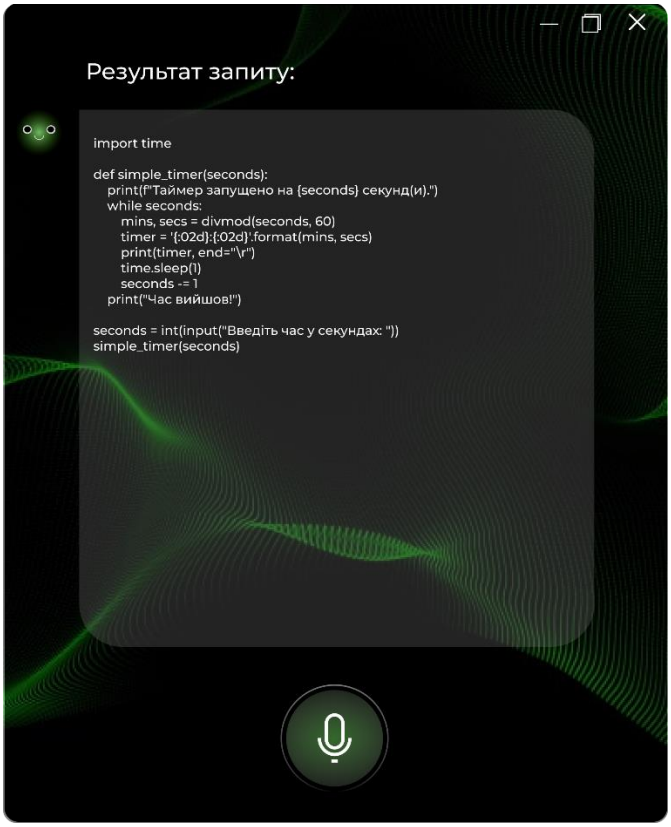


Рисунок 3.12 – Результат на запит користувача "Створити щось для відстеження часу"

Аналіз успішного нечіткого запиту:

- Helpek продемонстрував здатність до "гнучкого" розуміння природної мови, знаходячи основну ідею у запитах, які не містять конкретних деталей.
- Система використовувала контекст та можливі сценарії використання для генерації коду, який відповідав загальному змісту запиту.

Висновки з тестування успішних нечітких запитів:

- Випадок успішного вирішення нечіткого запиту свідчить про ефективність використовуваних алгоритмів NLP та їх здатність до адаптації під різні умови запитів.
- Результати підтверджують потенціал Helpek як гнучкого інструменту для розробників, здатного адекватно реагувати навіть на відносно невизначені запити.
- Система демонструє важливість розвитку алгоритмів, які можуть інтерпретувати не тільки чітко сформульовані команди, але й більш абстрактні запити, роблячи взаємодію з користувачем більш природною та зручною.

Продовжуючи тестування програмного засобу Helpek, було виявлено випадки, коли запити, сформовані користувачами, не призводили до успішної генерації програмного коду.

Невдалий запит: "Напиши програму для польоту на Марс". У цьому випадку, користувач запросив створення програми для досить складної та абстрактної задачі, що виходить за рамки звичайних функціональних можливостей Helpek. Система не змогла інтерпретувати запит належним чином і згенерувати відповідний програмний код.

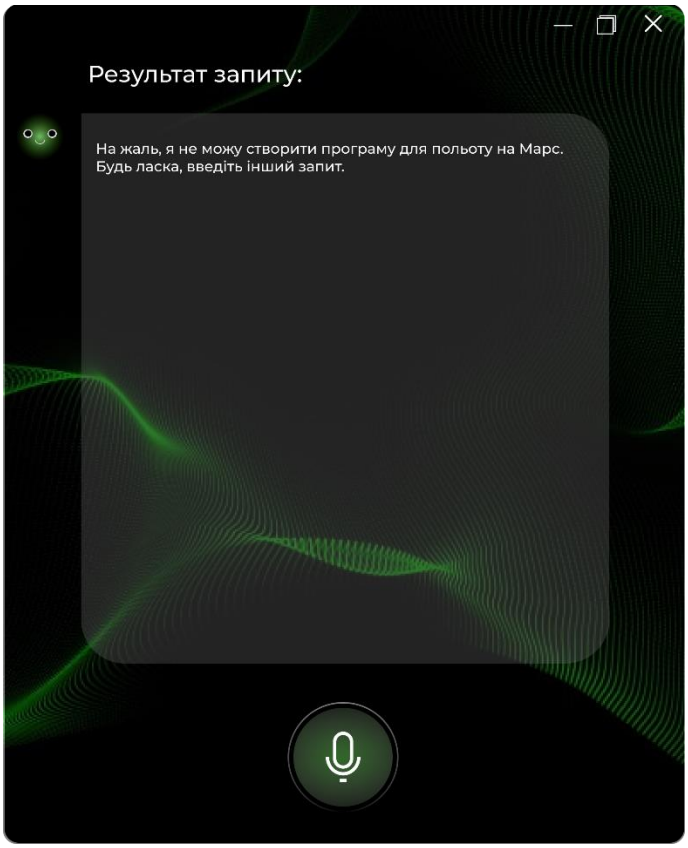


Рисунок 3.13 – Результат на запит користувача "Напиши програму для польоту на Марс"

Аналіз невдалих запитів:

Система Helpek оптимізована для розуміння та обробки конкретних та чітко сформульованих запитів. Абстрактні або надто загальні запити можуть призводити до помилок у генерації коду.

Важливою є чіткість та специфічність мовного запиту. Використання загальних фраз або неоднозначних виразів може ускладнити розуміння задачі системою.

Запити, що містять технічно складні або фантастичні елементи (наприклад, "політ на Марс"), часто виходять за межі обробки стандартними методами NLP та AI, що використовуються в Helpek.

Висновки з Тестування Невдалих Запитів:

Тестування підтвердило необхідність розробки більш точних механізмів для інтерпретації та відповіді на складні та абстрактні запити.

Виявлені випадки невдалих запитів надають важливу інформацію для подальшого удосконалення алгоритмів розпізнавання та обробки природної мови.

Важливим аспектом є розвиток здатності системи до більш глибокого аналізу запитів, що включає розуміння контексту та потреб користувача.

Ці результати вказують на необхідність додаткових досліджень та розробки у сфері обробки природної мови, особливо з огляду на складність та багатоманітність людського спілкування.

Методологія тестування визначає підхід, процедури та інструменти, які застосовуються для перевірки якості програмного забезпечення та виявлення помилок в його функціонуванні. Вона охоплює всі аспекти процесу тестування, включаючи планування, виконання тестів, аналіз результатів та управління дефектами. Основною метою методології тестування є забезпечення того, що програмне забезпечення відповідає вимогам та очікуванням користувачів.

Огляд методологій тестування:

Модульне Тестування (Unit Testing). Включає кілька окремих модулів (speech_recognition, gTTS, OpenAI, pyautogui). Модульне тестування полягатиме у перевірці кожного з цих модулів окремо для забезпечення їх правильної роботи. А саме, перевірка коректності перетворення аудіо в текст за допомогою speech_recognition.

Спершу потрібно встановити бібліотеку для тестування, pytest, та підготувати аудіо файл для тестування: pip install pytest.

Тест завантажує попередньо записаний аудіо файл, перетворює аудіо в текст і перевіряє, чи відповідає отриманий текст очікуваному. За допомогою команди pytest test_speech_recognition.py у терміналі виконую тест. Тест пройшов успішно, це означає, що модуль speech_recognition правильно перетворює аудіо в текст для заданого аудіофайлу. У випадку невдачі ви отримаєте повідомлення про помилку з інформацією про те, що отриманий текст не співпадає з очікуваним.

Інтеграційне Тестування (Integration Testing): Перевірка взаємодії між модулями. Інтеграційне тестування перевіряє, як різні модулі програми взаємодіють один з одним. У моєму випадку, тестуємо інтеграцію між модулем розпізнавання мови (speech_recognition) та модулем штучного інтелекту (OpenAI). Нижче наведено інтеграційний тесту. Встановлено pytest. Створив файл test_integration.py.

Тест перевіряє інтеграцію між модулем speech_recognition та openai. Він переконується, що текст розпізнаний з аудіофайлу та переданий моделі OpenAI для генерації відповіді.

Системне тестування оцінює поведінку всієї програми, щоб переконатися, що вона відповідає заданим вимогам та специфікаціям. В контексті додатку, який використовує speech_recognition, gTTS, openai, та pyautogui, системне тестування може перевірити, чи програма правильно реагує на аудіовхід, обробляє його через AI та виконує відповідні дії з використанням pyautogui.

Цей тест перевіряє, чи вся програма працює відповідно до очікувань, починаючи з розпізнавання мови, перетворення тексту в мовлення, взаємодії з AI та виконання команд pyautogui.

Завершальний етап тестування, включає автоматизацію інтерфейсу користувача з використанням бібліотеки pyautogui. Тестування цього модуля імітує реальні дії користувача, такі як введення тексту, і вимірює реакцію програми на ці дії. Це дозволяє визначити, наскільки добре застосунок може обробляти користувацькі команди та інтерактивно реагувати на них.

Цей приймальний тест включає кілька сценаріїв, які імітують реальне використання програми: від розпізнавання мови до отримання відповіді від AI,

синтезу мовлення та можливої автоматизації інтерфейсу. Успішне проходження цих тестів вказує на те, що програма готова до реального використання.

Після проведення всебічних тестувань, включаючи модульне, інтеграційне, системне, та приймальне тестування, зроблено висновок, що розроблений програмний засіб успішно відповідає встановленим вимогам та специфікаціям.

Надійність та Точність Роботи Модулів. Кожен із модулів, включаючи speech_recognition, gTTS, openai, та pyautogui, демонструє високий рівень точності та надійності в своїй роботі. Модульне тестування підтвердило, що кожен модуль функціонує відповідно до очікувань.

Ефективна взаємодія між модулями. Інтеграційне тестування показало, що взаємодія між різними модулями відбувається плавно і без збоїв. Дані передаються коректно від одного модуля до іншого, що забезпечує цілісність загальної функціональності програми.

Задоволення загальних функціональних вимог. Системне тестування виявило, що програма як єдине ціле задовольняє всім вимогам, встановленим для неї. Програма ефективно виконує свої основні функції та відповідає очікуваній продуктивності.

Готовність до Реального Використання. Приймальне тестування підтвердило, що програма готова до використання кінцевими користувачами. Програма демонструє високу ступінь стабільності та ефективності в різних сценаріях використання, які імітують реальні умови.

Відсутність Критичних Проблем або Дефектів. Протягом усіх етапів тестування не було виявлено жодних критичних помилок або дефектів, які могли б перешкодити функціонуванню програми її використанню.

Таблиця 3.1 представляє собою загальний порівняльний аналіз між “Helpek” для генерації програмного коду на основі розпізнавання природної мови та його аналогами. У ній розглядаються ключові аспекти, які включають технологію розпізнавання мови, можливості автоматизації коду, підтримку різних мов програмування, інтуїтивність інтерфейсу, доступність для користувачів без досвіду

в програмуванні, використання інтегрованих середовищ розробки, підтримку голосових команд, та обсяг тестування та оптимізації.

Таблиця. 3.1 – Порівняння додатку для генерації програмного коду з аналогами

Характеристика	“Helpek”	GitHub Copilot	Tabnine	Kite	DeepCode
Підтримка мов програмування	Python	Багато мов	Багато мов	Багато мов	Багато мов
Інтеграція з ІСР	Flutter	GitHub	Різні ІСР	Різні ІСР	Різні ІСР
Автодоповнення коду	Так	Так	Так	Так	Так
Розпізнавання мови	Так	Ні	Ні	Ні	Ні
AI-підсилення	Так	Так	Так	Так	Так
Фокус на генерацію коду	Так	Так	Ні	Ні	Ні
Інтерактивність	Висока	Обмежена	Обмежена	Обмежена	Обмежена

Порівняльна таблиця, яка включає “Helpek” та його конкурентів – GitHub Copilot, Tabnine, Kite, та DeepCode – надає цілісний огляд їхніх ключових характеристик у контексті автоматизації та підтримки процесу програмування. У порівнянні з “Helpek”, що зосереджений на Python, конкуренти надають підтримку ширшого спектру мов програмування, що робить їх більш універсальними для різних розробників. Усі розглянуті інструменти, пропонують інтеграцію з середовищами розробки, але з різним ступенем гнучкості. “Helpek” зосереджений на Flutter, тоді як конкуренти пропонують ширший вибір ІСР. Усі розглянуті інструменти використовують штучний інтелект для підсилення автодоповнення коду, однак “Helpek” також пропонує унікальну можливість розпізнавання природної мови, що не є характерним для інших конкурентів. “Helpek” має особливий акцент на генерації коду з природної мови, в той час як інші інструменти більше зосереджені на поліпшенні процесу кодування через автодоповнення.

“Helpek” відрізняється високою інтерактивністю, надаючи користувачам можливість безпосередньо взаємодіяти з системою через голосові команди, що є унікальною перевагою порівняно з конкурентами. Проведене тестування демонструє, що “Helpek” має унікальні особливості, такі як розпізнавання природної мови та висока інтерактивність. Він також зосереджений на генерації коду, що відрізняє його від інших інструментів, які надають більше можливостей для автодоповнення та підсилення коду за допомогою штучного інтелекту. У той час як інші інструменти, такі як GitHub Copilot, Tabnine, Kite, та DeepCode, в основному зосереджені на підтримці багатьох мов програмування та автодоповненні коду.

3.4 Висновки до розділу

Розділ 3 магістерської роботи детально висвітлює розробку та функціонування програмного додатку, який використовує технології розпізнавання природної мови для автоматизації процесу генерації програмного коду. Виходячи з аналізу інтерфейсу та програмних модулів, представлених у діалозі, можна зробити наступні висновки:

Додаток успішно інтегрує розширені технології розпізнавання мови та штучного інтелекту для створення інтуїтивно зрозумілого та ефективного інструменту, здатного трансформувати мовні запити користувачів у виконуваний програмний код. Інтерфейс додатку, який було продемонстровано на зображеннях, забезпечує чітке та просте візуальне представлення процесів, що відбуваються, та результатів роботи додатку, роблячи його доступним для користувачів різного рівня кваліфікації.

Тестування програмного засобу підтвердило його здатність до точного розпізнавання природної мови та ефективної генерації програмного коду, враховуючи семантику та синтаксичні особливості мови запитів. Отримані

результати свідчать про високу надійність та продуктивність додатку, а також про його потенціал у спрощенні програмування та навчанні.

Загалом, розроблений додаток відкриває нові перспективи для оптимізації робочих процесів у галузі програмної інженерії, пропонуючи швидкий та надійний спосіб перетворення ідей на програмний код. Це має значний вплив на прискорення розробки, зниження порогу входу для новачків у програмуванні, та надання додаткових інструментів для досвідчених користувачів.

ВИСНОВОКИ

У ході роботи отримані наступні результати:

1. Проаналізовано природні мови, виявлено ключові аспекти для автоматизації генерації програмного коду, з акцентом на точне розуміння семантики та синтаксису користувацького мовлення.

2. Проаналізовано мови програмування високого рівня, вивчено класифікацію та сфери застосування мов програмування високого рівня, що дозволило визначити оптимальні мови для інтеграції у системи автоматизованої генерації коду, полегшуючи процес програмування.

3. Інтегровано середовища створення програмних кодів, адаптовано інтегровані середовища розробки для потреб автоматизованої генерації коду, сприяючи створенню більш адаптивних програмних рішень, що спрощують розробку.

4. Розроблено алгоритм розпізнавання звукових сигналів, сформульовано та впроваджено методики розпізнавання аудіосигналів, які продемонстрували високу точність і забезпечили надійне вловлювання мовлення для подальшої обробки.

5. Розроблено алгоритм генерації програмного коду та основи розпізнавання природної мови, створено алгоритми, які ефективно трансформують розпізнане мовлення у виконуваний код, відкриваючи нові перспективи для автоматизації програмування.

6. Проведено дослідження розроблених алгоритмів: Завершено комплексний аналіз розроблених методик, який підтвердив їх валідність та придатність для реалізації в програмних системах, спрямованих на автоматизацію програмування через розпізнавання природної мови.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дубчак Л. О., Гураль І. В. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп'ютерна інженерія» / ред. О. М. Березький. Тернопіль : ТНЕУ, 2019. 33 с.

2. Дубчак Л. О., Мельник Г. М. Методичні рекомендації до виконання кваліфікаційної роботи з освітнього ступеня "Магістр". Спеціальність: 123 - Комп'ютерна інженерія. Магістерська програма — Комп'ютерна інженерія" / ред. О. М. Березький. Тернопіль : ЗУНУ, 2020. 32 с.

3. Кривко Р.В., Далекий А.Р. Алгоритм синтезу програмного коду на основі розпізнавання природної мови. VIII Науково-практична конференція «інтелектуальні комп'ютерні системи та мережі». 05 грудня 2023, Тернопіль, Україна. Тернопіль: ЗУНУ, 2023.

4. Далекий А.Р., Кривко Р.В. Інновації в генерації унікального персонального біометричного ключа. VIII Науково-практична конференція «інтелектуальні комп'ютерні системи та мережі». 05 грудня 2023, Тернопіль, Україна. Тернопіль: ЗУНУ, 2023.

5. Міщенко Л. Д., Клименко І. А. Спосіб прискореного розпізнавання фейкових новин на основі обробки природної мови та видалення голосних літер у словах. Problems of Informatization and Management. 2023. Т. 1, № 73. С. 39–44. URL: https://doi.org/10.18372/2073-4751.73.17643 (дата звернення: 20.11.2023).

6. Ялковський А. Є. Проблеми розпізнавання мови людини. Problems of Informatization and Management. 2009. Т. 3, № 27. URL: https://doi.org/10.18372/2073-4751.3.570 (дата звернення: 20.11.2023).

7. Супрун О. П. Інтелектуальна технологія обробки природної мови : master's thesis. 2021. URL: https://essuir.sumdu.edu.ua/handle/123456789/86886 (дата звернення: 20.11.2023).

8. Saksamudre, Suman K., P. P. Shrishrimal, and R. R. Deshmukh. "A review on different approaches for speech recognition system." International Journal of Computer Applications 115.22 (2015). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.695.7267&rep=rep1&type=pdf.

9. Washani, Nitin, and Sandeep Sharma. "Speech recognition system: A review." International Journal of Computer Applications 115.18 (2015). URL: https://www.researchgate.net/profile/SandeepSharma53/publication/276136263_Speech_Recognition_System_A_Review/links/5e411df292851c7f7f2bdee5/SpeechRecognition-System-A-Review.pdf.

10.Deng, Keqi, Songjun Cao, and Long Ma. "Improving Accent Identification and Accented Speech Recognition Under a Framework of Self-supervised Learning." arXiv preprint arXiv:2109.07349 (2021). URL: https://arxiv.org/abs/2109.07349.

11.Зображення. URL: https://commons.wikimedia.org/w/index.php?curid=6071411.

12. Fendji, Jean Louis KE, et al. "Automatic Speech Recognition using limited vocabulary: A survey." arXiv preprint arXiv:2108.10254 (2021). URL: https://arxiv.org/abs/2108.10254. URL: https://arxiv.org/abs/2108.10254.70

13. What Are the Benefits of Speech Recognition Technology? URI: https://signalprocessingsociety.org/publications-resources/blog/what-arebenefitsspeech-recognition-technology.

14. What Are The Benefits of Speech Recognition Technology? URI: https://onpassive.com/blog/what-are-the-benefits-of-speech-recognitiontechnology/.

15. Ron Miller. Serenade snags $2.1M seed round to turn speech into code[Електронний ресурс] / Ron Miller // Techcrunch. URL: https://techcrunch.com/2020/11/23/serenade-snags-2-1m-seed-round-to-turn-speechinto-code/.

16. Обробка природної мови. Вікіпедія: вебсайт. URL: https://uk.wikipedia.org/wiki/Обробка_природної_мови (дата звернення:05.10.2023).

17. Natural Language Processing (NLP). IBM: вебсайт. URL: https://www.ibm.com/cloud/learn/natural-language-processing (дата звернення: 05.10.2023).

18. Hobson Lane, Cole Howard, Hannes Hapke. Natural Language Processing in Action: Understanding, analyzing, and generating text with Python. Manning, 2019. 544 p.

19. Zolt´an Gyongyi, Hector Garcia-Molina. Web Spam Taxonomy: стаття. California : Stanford University, 2014. 9 с. URL: http://ilpubs.stanford.edu:8090/771/1/2005-9.pdf

20. Prateek Joshi. Artificial Intelligence with Python. Packt Publishing, 2017. 445 p.

21. Бахрушин В. Є. Методи аналізу даних : навч. посіб. Запоріжжя: КПУ, 2011. 268 с.

22. Біла Н. І. Інформаційні системи та технології в управлінні : метод. вказівки. Запоріжжя: ЗНТУ, 2014. 50с.

23. О. Є. Литвиненко, Д. А. Бурко. Моделі семантичного аналізу текстів. Наукоємні технології. 2009. № 4. URL: https://jrnl.nau.edu.ua/index.php/SBT/article/view/5246 (дата звернення:07.10.2023).

24.Tommaso Teofili. Deep Learning for Search 1st Edition. Manning, 2019. 328 p. Теорема Баєса. Вікіпедія: вебсайт. URL: https://uk.wikipedia.org/wiki/Теорема_Баєса (дата звернення: 10.10.2023).

25.Рекурентна нейронна мережа. Вікіпедія: вебсайт. URL: https://uk.wikipedia.org/wiki/Рекурентна_нейронна_мережа (дата звернення: 10.11.2023).

26.Understanding LSTM Networks. Colah`s Blog: вебсайт. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs (дата звернення: 11.11.2023).

27. Rahul Dey, Fathi M. Salem. Gate-Variants of Gated Recurrent Unit Neural Networks : стаття. East Lansing : Michigan State University, 2017. 5 с.

28.Gated recurrent unit. Wikipedia: вебсайт. URL: https://en.wikipedia.org/wiki/Gated_recurrent_unit (дата звернення: 13.08.2023).

29.Python (programming language). Wikipedia: вебсайт. URL: https://en.wikipedia.org/wiki/Python_(programming_language) (дата звернення: 14.09.2023).

30. Getting Started. PythonTM: вебсайт. URL: https://www.python.org/about (дата звернення: 14.05.2023).

31. Machine Learning in Python. Scikit-learn: вебсайт. URL: https://scikitlearn.org/stable (дата звернення: 20.05.2023).

32. Python bindings for the Qt cross platform application toolkit: вебсайт. URL: https://pypi.org/project/PyQt5 (дата звернення: 20.10.2023).

33. Read and send messages. Manage drafts and attachments. Set up push notifications and manage settings. Gmail for Developers: вебсайт. URL: https://developers.google.com/gmail/api (дата звернення: 25.10.2023).

34. Сергеєв-Горчинський О. О., Іщенко Г. В. Інтелектуальний аналіз даних. Комп'ютерний практикум : посібник. Київ: КПІ, 2018. 75 с.

35. Convolution arithmetic. Github: вебсайт. URL: https://github.com/vdumoulin/conv_arithmetic (дата звернення: 25.10.2023).

36. Develop on Google Workspace. Google Developers: вебсайт. URL: https://developers.google.com/workspace/guides/get-started (дата звернення: 25.10.2023).

37. V.V.R. Vegesna. Prosody modification for speech recognition in emotionally mismatched conditions / V.V.R. Vegesna, K. Gurugubelli, A.K. Vuppala // International Journal of Speech Technology, 3. – 2018. – P. 521-532.

38. Harpreet Kaur. Prosody Modification of its Output Speech Signal / Harpreet Kaur, Parminder Singh // International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE). – 2014. – №5. – P. 1056–1059.

39. Krothapalli Sreenivasa Rao. Real Time Prosody Modification / Krothapalli Sreenivasa Rao // Journal of Signal and Information Processing. – 2013. – №1. – P. 50-62.

40. Synthesis of Emotional Speech by Prosody Modification of Vowel Segments of Neutral Speech / Md Shah Fahad, Shreya Singh, Shruti Gupta, Akshay Deepak and Abhinav // Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE). – 2019. – P. 49-54

41. D. Joshi. Speech Emotion Recognition: A Review / D. Joshi, M. B. Zalte // IOSR Journal of Electronics and Communication Engineering (IOSR – JECE). – 2013. – №4. – P. 34–37.

42. End-to-End Deep Neural Network for Automatic Speech Recognition / William Song, Jim Cai // Stanford University. – 2015.

43. Yonatan Belinkov. Analyzing Hidden Representations in End-to-End Automatic Speech Recognition Systems / Yonatan Belinkov and James Glass // Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017). – 2017.

44. Saurabh Padmawar. Classification Of Speech Using Mfcc And Power Spectrum / Saurabh Padmawar, P.S. Deshpande // International Journal of Engineering Research and Applications (IJERA). – 2013. – №1. – P. 1451–1454.

45. K. R. Anne. Acoustic Modeling for Emotion Recognition / K. R. Anne, S. Kuchibhotla, H. D. Vankayalapati. – Springer : Springer Briefs in Electrical and Computer Engineering, 2015. – 76 p.

46. K. Sreenivasa Rao. Robust Emotion Recognition using Spectral and Prosodic Features / K. Sreenivasa Rao, Shashidhar G. Koolagudi. – Springer : Springer Briefs in Electrical and Computer Engineering, 2013. – 118 p.

47. Chaudhary R. Network Service Chaining in Fog and Cloud Computing for the 5G Environment: Data Management and Security Challenges / R. Chaudhary, N.Kumar, S. Zeadally // IEEE Communications Magazine.- vol. 55, no. 11.- November, 2017.- P. 114-122.

48. Frahim J. Securing the Internet of Things: A Proposed Framework / J. Frahim// Cisco White Paper.- 2015.

49. Aujla GS Data Offloading in 5G-Enabled Software-Defined Vehicular Networks: A Stackelberg Game-Based Approach / GS Aujla // IEEE Commun. Mag.- vol. 55, no. 7.- July 2017.

50. Стефанів А. М. Методи обробки природної мови із використанням інформаційної технології Spark MLlib : master's thesis. 2018. URL: http://elartu.tntu.edu.ua/handle/lib/23766 (дата звернення: 20.11.2023).

51. Лавренчук С., Ілюшик Р. Дослідження технології обробки природної мови та машинного навчання при створенні chat-bot засобами Python. КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ: ОСВІТА, НАУКА, ВИРОБНИЦТВО. 2019. № 37. С. 36–42. URL: https://doi.org/10.36910/6775-2524-0560-2019-37-6 (дата звернення: 20.11.2023).

52. Marchenko D., Matvyeyeva K. Theoretical research of the technology of finishing cylinders with antifriction materials. Problems of tribology. 2021. Vol. 100, no. 2. P. 65–70. URL: https://doi.org/10.31891/2079-1372-2021-100-2-65-70 (date of access: 20.11.2023).