Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих теїнологій
(повна назва)

Кафедра Компьютерно-інтегрованих технологій, автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розробка інтелектуальної голосової системи підтримки прийняття рішень у виробничому приміщені
(тема)

Виконав: Студент 4 курсу,
групи АКТАКІТ-20-2
Терещук Даніїл Олександрович
(прізвище, ініціали)

Спеціальність 151 Автоматизація та Компьютерно-інтегровані технології
(код і повна назва спеціальності)

Тип програми Освітньо-професійна

Осітня програма АКІТ
(повна назва освітньої програми)

Керівник доц.каф.КІТАР Максимова С.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Невлюдов І.Ш.
(підпис) (призвище, ініціали)

2024

Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки

Рівень вищої освіти перший (бакалаврський)

Спеціальність 151 «Автоматизація та комп'ютерно-інтегровані технології»
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Автоматизація та комп'ютерно-інтегровані технології

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«____»_____20____р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Терещуку Даніїлу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка інтелектуальної системи підтримки прийняття рішень у виробничому приміщені

затверджена наказом університету від 03.06.2024 р. № 544 Ст

2. *Термін подання студентом роботи до екзаменаційної комісії* 18.06.2024 р.

3. Вихідні дані до роботи

Призначення і цілі розробки: система призначена для зменшення часу ідентифікації помилок та зупинок роботи на підприємствах

3.2 Середовище розробки: Python

3.3 Дані мають передаватися по локальній мережі.

3.4 Сумісність з ОС Windows, Android, iOS

3.5 Голос ассистента повинен бути чітким для користувача.

4. Перелік питань, що потрібно опрацювати в роботі

4.1 Аналіз сучасного стану автоматизації технічних процесів.

4.2 Аналіз існуючих комерційних рішень для систем голосового управління та голосових повідомлень.

4.3 Розробка програмного забезпечення.

4.4 Реалізація модульної частини.

4.5 Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)

_____
_____
_____
_____
_____

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	*Аналіз технічного завдання*	18.05.2024	*виконано*
2	*Аналіз сучасного стану автоматизації технічних процесів*	19.05.2024	*виконано*
3	*Аналіз існуючих комерційних рішень для систем голосового управління та голосових повідомлень*	21.05.2024	*виконано*
4	*Розробка програмного забезпечення.*	24.05.2024	*виконано*
5	*Створення програмного забезпечення*	27.05.2024	*виконано*
6	*Реалізація модульної частини.*	02.06.2024	*виконано*
7	*Охорона праці*	04.06.2024	*виконано*
8	*Оформлення пояснювальної записки*	17.06.2024	*виконано*
9	*Подання роботи на перевірку на плагіат*	08.06.2024	
10	*Подання роботи на рецензію*	10.06.2024	
11	*Подання роботи на підпис зав. кафедри*	12.06.2024	
12	*Подання роботи до ЕК*	18.06.2024	

Дата видачі завдання 03.06.2024 р.

Студент _____ Терещук Д.О.

(підпис)

Керівник роботи _____ доц. Максимова С.С.

(підпис) (посада, прізвище, ініціали)

Я, як студент(ка) ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав(ла) і не одержував(ла) недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на віповідне джерело.

16.06.2024  Терещук Д.О.

РЕФЕРАТ

Пояснювальна записка: 114 с., 2 табл., 111 рис., 2 дод., 5 джерел.

ІДЕНТИФІКАЦІЯ, РОСПІЗНАВАННЯ МОВИ, СИТЕЗ МОВИ, АВТОМАТИЗАЦІЯ, АЛГОРИТМИ ШТУЧНОГО ІНТЕЛЕКТУ, МОДУЛЬНА СИСТЕМА, АНАЛІЗ ДАНИХ, ІННОВАЦІЇ У ВИРОБНИЦТВІ, ВИРОБНИЧІ ІННОВАЦІЇ.

Мета роботи – покращення продуктивності автоматизації на виробництві за рахунок розробки системи моніторингу.

Об'єкт дослідження – процес автоматизації прийняття рішень у виробничому приміщені.

Предмет дослідження – програмне забезпечення для автоматизації прийняття рішень у виробничому середовищі.

Методами дослідження є метод аналізу та метод синтезу.

Проведено аналіз методів і технологій знаходження помилок. Також проаналізовано існуючі комерційні рішення для систем синтезу мовлення, досліджено їх структуру, технічні параметри, функціональні можливості та особливості їх використання, а також виявлено переваги та недоліки цих систем. Проведено вибір компонентів системи, описано її конструктивні та функціональні особливості, наведено її технічні характеристики.

Розроблено програмну частину, що забезпечує автоматизовану ініціалізацію, обробку та передачу інформації про ідентифіковану помилку шляхом синтезу мови для подальшого аналізу цих даних.

ABSTRACT

Explanatory note: 114 pages, 2 tables, 111 pictures, 2 app, 5 sources.

IDENTIFICATION, SPEECH RECOGNITION, SPEECH SYNTHESIS, AUTOMATION, ARTIFICIAL INTELLIGENCE ALGORITHMS, MODULAR SYSTEM, DATA ANALYSIS, MANUFACTURING INNOVATION, INDUSTRIAL INNOVATION.

The purpose of the work is to improve productivity in production automation by developing a monitoring system.

The object of research is the process of decision-making automation in a production facility.

The subject of research is software for automating decision-making in a production environment.

The research methods include the method of analysis and the method of synthesis.

An analysis of methods and technologies for error detection has been conducted. Existing commercial solutions for speech synthesis systems were also analyzed, their structure, technical parameters, functional capabilities, and features of use were investigated, as well as the advantages and disadvantages of these systems were identified. The system components were selected, and their structural and functional features were described, along with their technical characteristics.

A software part has been developed that ensures the automated initialization, processing, and transmission of information about identified errors through speech synthesis for further analysis of this data.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення;

ІСППР – інтелектуальна система підтримки прийняття рішень;

ШІ – штучний інтелект;

МН – машинне навчання;

TTS – “Text-to-Speech” або “текст у мову”;

JSON – JavaScript Object Notation.

ВСТУП

У світі, де технології постійно еволюціонують і обсяги даних зростають з кожним днем, автоматизація технічних процесів відіграє вирішальну роль у підвищенні ефективності і оптимізації ресурсів у всіх галузях промисловості. Серед інноваційних технологій, що пропонують значні переваги, інтелектуальні системи підтримки прийняття рішень вирізняються здатністю аналізувати великі обсяги інформації, пропонувати обґрунтовані рішення та адаптуватися до змін у реальному часі. Основною метою цієї роботи є розробка такої системи, яка б використовувала штучний інтелект для поліпшення процесів прийняття рішень на виробництві.

Програмне забезпечення, яке розробляється у рамках даної дипломної роботи, призначене для використання у складних виробничих умовах, де потрібно швидко обробляти дані та реагувати на зміни. Система включатиме модулі для збору даних, їх аналізу, модулі прогнозування та інтерфейси для користувачів, що дозволять їм ефективно взаємодіяти з технологією. Алгоритми штучного інтелекту, розроблені в рамках проекту, будуть використовуватися для ідентифікації оптимальних рішень, дозволяючи системі пропонувати стратегічні варіанти дій на основі глибокого аналізу наявних даних.

Значення розробки такої системи важко переоцінити, оскільки вона сприятиме підвищенню технічної та економічної ефективності підприємств. Інтеграція інтелектуальних систем підтримки прийняття рішень дозволить не тільки скоротити час на аналіз та виконання процесів, але й забезпечити вищу точність і надійність управління ресурсами. Таким чином, впровадження цієї системи допоможе підприємствам залишатися конкурентоспроможними на ринку, адаптуватися до швидких змін і максимально використовувати свої можливості в умовах глобалізації.

Крім технічної ефективності, важливість інтелектуальної системи підтримки прийняття рішень також полягає в її здатності підвищувати якість прийнятих рішень. Через інтеграцію різноманітних джерел даних та їх

комплексний аналіз, ІСППР допомагає уникнути суб'єктивності в рішеннях, забезпечуючи об'єктивність і всебічність у вирішенні проблем. Це знижує витрати, пов'язані з помилковими рішеннями, і забезпечує вищу загальну ефективність процесів у компанії.

Розробка інтелектуальної системи підтримки прийняття рішень на основі штучного інтелекту є важливим кроком для сучасних виробничих підприємств. Вона дозволяє підвищити ефективність, знизити витрати, забезпечити високу якість рішень та сприяти сталому розвитку. Використання сучасних технологій та інструментів забезпечує надійність та розширюваність системи, що є критично важливим в умовах швидких змін та високої конкуренції на ринку. Успішна реалізація даного проекту стане значним внеском у розвиток інноваційних технологій у промисловості, сприяючи підвищенню конкурентоспроможності підприємств та забезпеченню їх стабільного розвитку в умовах глобалізації.

Мета роботи – покращення продуктивності автоматизації на виробництві за рахунок розробки системи моніторингу.

Об'єкт дослідження – процес автоматизації прийняття рішень у виробничому приміщені.

Предмет дослідження – програмне забезпечення для автоматизації прийняття рішень у виробничому середовищі.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз сучасного стану автоматизації технічних процесів;
– проаналізувати існуючі комерційні рішення для систем голосового управління та голосових повідомлень;
– розробити програмне забезпечення;
– реалізувати модульну частину;
– Ініціалізація підсистеми друку;
– Ініціалізація і запуск головної програми;
– Сформолювати охорону праці.

1 АНАЛІЗ СУЧАСНОГО СТАНУ АВТОМАТИЗАЦІЇ ТЕХНІЧНИХ ПРОЦЕСІВ

1.1 Значення інтелектуальних систем підтримки прийняття рішень

У сучасному світі, де швидкість прийняття рішень та їх точність стають вирішальними факторами успіху підприємства, важливість автоматизації процесів зростає. Інтелектуальні системи підтримки прийняття рішень (ІСППР) відіграють ключову роль у цьому процесі, забезпечуючи компанії інструментами для ефективного аналізу даних та оптимального вирішення проблем. Цей підрозділ досліджує основні переваги та значення впровадження ІСППР у різні аспекти виробничих процесів.

Однією з основних переваг ІСППР є їх здатність підвищувати ефективність та знижувати операційні витрати. Системи здатні швидко обробляти великі обсяги даних, виділяючи значущі тенденції та аномалії, які можуть вплинути на виробничий процес. Це дозволяє ухвалювати рішення на основі актуальної та об'єктивної інформації, значно скорочуючи час на аналіз та підготовку даних. Швидке реагування на виробничі зміни та можливість прогнозування виробничих потреб дозволяють компаніям оптимізувати використання ресурсів, зменшувати відходи і уникати дорогих збоїв у виробництві.

Завдяки ІСППР, підприємства можуть не тільки реагувати на поточні події, а й прогнозувати майбутні зміни. Системи підтримки прийняття рішень використовують передові алгоритми машинного навчання для аналізу історичних даних, що дозволяє їм виявляти патерни і передбачати майбутні тенденції. Такий підхід значно підвищує стратегічну гнучкість та адаптивність підприємств, роблячи їх більш конкурентоспроможними у динамічному бізнес-середовищі.

Крім того, ІСППР допомагають підвищити якість прийнятих рішень. Завдяки інтеграції різноманітних джерел даних та їх комплексному аналізу, системи забезпечують високий рівень об'єктивності. Це відіграє критичну

роль у складних виробничих процесах, де кожне рішення може мати значні фінансові наслідки. Завдяки точному аналізу та зменшенню людського фактору, можливість помилок значно знижується, що забезпечує стабільність і надійність виробничих процесів.

1.2 Інноваційні технології в галузі штучного інтелекту

Останні роки принесли значні інновації у сфері штучного інтелекту (ШІ) та машинного навчання (МН), які революціонізували багато аспектів виробничих процесів. Завдяки їхньому впровадженню в системи підтримки прийняття рішень (ІСППР), підприємства здобувають можливість значно підвищити точність своїх прогнозів та ефективність автоматизації. Цей підрозділ розглядає, як сучасні технології ШІ та МН інтегровані у ІСППР, а також аналізує їх вплив на операційну діяльність компаній.

ШІ та МН дозволяють системам підтримки прийняття рішень глибше аналізувати дані, виявляючи закономірності, які не очевидні для людського ока. Наприклад, алгоритми машинного навчання можуть автоматично адаптуватися до змін у виробничому процесі, прогнозуючи потенційні збої або підвищення попиту на певні продукти. Такий глибокий аналітичний потенціал не тільки підвищує продуктивність, але й дозволяє підприємствам оперативно реагувати на ринкові зміни, підтримуючи високу конкурентоспроможність.

1.2.1 Штучний інтелект: Огляд і застосування

Штучний інтелект (ШІ) – це галузь комп'ютерних наук, яка займається створенням машин, здатних імітувати людські когнітивні функції, такі як навчання, розуміння та рішення задач. Розвиток ШІ останніми роками перетворив його з наукової фантастики в реальність, яка впливає на багато аспектів нашого життя, від повсякденної взаємодії з персональними асистентами до глибоких змін у промисловості, медицині, і транспорті.

Штучний інтелект об'єднує в собі різноманітні технології, включаючи машинне навчання, нейронні мережі, глибоке навчання, обробку природної мови, і робототехніку. Ключовим аспектом ШІ є його здатність до навчання та адаптації: алгоритми машинного навчання можуть аналізувати великі обсяги даних, виявляти закономірності та вдосконалювати свою поведінку без прямого людського втручання.

Машинне навчання ділиться на кілька підтипів, включаючи навчання під наглядом, ненаглядоване навчання та навчання з підкріпленням, кожен з яких має свої особливості та області застосування. Наприклад, навчання під наглядом використовується для класифікації імейлів на спам та не-спам, тоді як навчання з підкріпленням дозволяє розробляти системи, які оптимізують свої дії на основі винагороди або покарань.

1.2.2 Вплив штучного інтелекту на автоматизацію

У контексті автоматизації технічних процесів, штучний інтелект відіграє ключову роль у підвищенні ефективності, точності та зниженні витрат. Використання ШІ дозволяє автоматизувати складні задачі, які традиційно вимагали людського втручання, такі як діагностика обладнання, управління запасами, і навіть цілі виробничі процеси. ШІ може аналізувати дані з сенсорів в реальному часі, передбачати потенційні збої або неефективності і надавати рекомендації для оптимізації.

Автоматизація з використанням ШІ може значно підвищити продуктивність. Машини, здатні аналізувати та оптимізувати свої дії, можуть виконувати рутинні задачі швидше і з меншою кількістю помилок, ніж людські оператори. Це звільняє персонал для зосередження на більш складних та креативних аспектах роботи, що в кінцевому підсумку веде до інновацій та росту підприємства.

Штучний інтелект забезпечує можливість аналізувати великі обсяги даних з небувалою швидкістю та точністю. В автоматизації це дозволяє системам не тільки виконувати задані інструкції, але й адаптуватися до змін у

виробничому середовищі без безпосереднього людського втручання. Використання алгоритмів машинного навчання дозволяє обладнанню "вчитися" з попередніх досвідів, оптимізуючи свою продуктивність за рахунок зменшення помилок і збільшення продуктивності.

Впровадження штучного інтелекту у сферу автоматизації несе з собою і певні виклики. Передусім, це питання конфіденційності та безпеки даних. Крім того, зростає залежність від технологій, що може впливати на стабільність робочих процесів у випадку технічних збоїв. Однак, ці виклики можна перетворити на можливості для подальшого розвитку та вдосконалення систем, що робить ШІ не лише актуальним, але й перспективним напрямком у сучасній автоматизації.

1.2.3 Основи машинного навчання

Машинне навчання – це галузь штучного інтелекту, яка забезпечує системам здатність навчатися і вдосконалюватися без прямого програмування. МН використовує статистичні методи для інтерпретації патернів у даних, що дозволяє прогнозувати результати. В машинному навчанні алгоритми підподіляються на кілька типів:

- навчання під наглядом вимагає вхідних даних і відповідних міток, щоб навчити модель, яка зможе робити прогнози або класифікації;
- ненаглядоване навчання використовується для аналізу і кластеризації великих обсягів немічених даних;
- навчання з підкріпленням стимулює модель вибирати оптимальні рішення на основі винагород та покарань.

Реалізація машинного навчання в автоматизації технічних процесів вимагає вирішення ряду викликів, включаючи забезпечення безпеки даних, інтеграцію систем в існуючі ІТ-інфраструктури, а також постійне оновлення моделей для відповідності змінам у зовнішньому середовищі. Тим не менш, перспективи машинного навчання в автоматизації обіцяють значні покращення в продуктивності, ефективності та інноваційності.

Машинне навчання змінює парадигми у багатьох галузях, зокрема у автоматизації технічних процесів. Його впровадження дозволяє створювати інноваційні, ефективні та адаптивні системи, що відіграють ключову роль у формуванні майбутнього промисловості і технологій. Як спеціаліст у цій області, я переконаний у важливості продовження досліджень і розробок у галузі машинного навчання, щоб максимально реалізувати його потенціал для підвищення якості життя та бізнес-ефективності.

1.2.4 ІСППР з використанням ШІ та МН

На практичних прикладах передових підприємств можна побачити, як застосування ІСППР із вбудованими технологіями ШІ та МН сприяє зростанню адаптивності компаній. Наприклад, великі роздрібні мережі використовують ІСППР для оптимізації логістики та управління запасами, що дозволяє їм мінімізувати витрати та підвищити рівень задоволеності клієнтів. Інші галузі, такі як виробництво, застосовують ІСППР для моніторингу виробничих ліній у реальному часі, що допомагає знизити частоту і серйозність збоїв у виробництві.

Аналізуючи ці приклади, стає зрозуміло, що ІСППР з інтегрованими технологіями ШІ та МН не тільки покращують існуючі процеси, але й відкривають нові можливості для розвитку та інновацій. Вони надають підприємствам засоби для значного підвищення їхньої ефективності та адаптивності, що є ключовими факторами успіху у сучасному бізнес-середовищі.

ІСППР з використанням ШІ та МН поєднує в собі розширені алгоритми машинного навчання і техніки штучного інтелекту для оптимізації процесів прийняття рішень. Ці системи використовують нейронні мережі для аналізу великих обсягів даних і ідентифікації патернів, що не можуть бути легко виявлені людиною. Це дозволяє виконувати комплексний аналіз і генерувати високоякісні прогнози та рекомендації, що спираються на глибоке розуміння даних.

ІСППР функціонує на основі декількох ключових принципів:

– збір даних: Система автоматично збирає вхідні дані через інтегровані інтерфейси або безпосередньо від користувачів через голосові команди;

– обробка даних: Дані проходять через модулі попередньої обробки для видалення шуму та нормалізації перед подальшим аналізом;

– аналіз даних: Використовуючи алгоритми МН, система аналізує дані для виявлення корисної інформації, патернів або аномалій;

– генерація рішень: На основі аналізу, система генерує рішення або рекомендації, які можуть бути представлені користувачам у зрозумілій формі;

– зворотний зв'язок: Система використовує зворотний зв'язок для оптимізації своїх майбутніх рішень, вчиться на своїх помилках або успіхах.

1.3 Актуальність ІСППР

"Інтелектуальна голосова система підтримки прийняття рішень" спрямована на підвищення загальної ефективності робочих процесів. Використання голосових команд і відповідей дозволяє персоналу зосередитись на вирішенні складніших завдань, зменшуючи час на рутинні операції. Система може аналізувати надходячі дані, виконувати пошук необхідної інформації та надавати користувачам зрозумілі, точні відповіді, значно знижуючи час на знаходження помилки та збільшуючи швидкість реакції на їх виправлення.

Актуальний приклад того є фірма з виготовлення дерев'яних євро вікон, що має при собі автоматизовану машину "Working Process". Вона виконує функцію сортування та фрезерування деталей. Має при собі до роботи 2 машиніста. Один займається підготовкою даних для роботи: креслення, розрахунок кількості деталей для заказу, розрахунки для роботи машини, тощо. Другий у свою чергу саме роботою машини, слідкувати за тим як вона працює. Для початку роботи загружати дошки далі слідкувати за процесом

оброблення, розвантажити дошки. Іноді під час того як машина працює, можуть траплятися автоматичні зупинки виконання. Що це значить? Машина під час своєї роботи знайшла помилку – це можуть бути:

– падіння дошки;
– зупинка роботи витяжки;
– не вірна маса заготовки для оброблення;
– знос фрези, свердла тощо.

Не стільки проблема підняти дошку чи замінити ніж фрези, скільки знайти, що дійсно сталося у найбільш короткий час. За день, такі зупинки можуть з’являтися до 10 разів, що займатимуть собою від 20 – 40 хвилин робочого часу. Тобто “Working Process” зупиняється.

Для найбільш правдивої актуалізації своїх данних, я звернувся до цієї фірми і вони надали мені данні щодо реальної кількості рутинних помилок, та затрат часу на їх виправлення. Завдяки впровадженню інтелектуальної голосової системи підтримки прийняття рішень, завод має можливість значно скоротити час, необхідний для вирішення зазначених проблем. Система, оснащена можливістю аналізувати надходячі дані та виконувати пошук необхідної інформації, може надавати користувачам зрозумілі та точні відповіді. Така технологія не тільки сприяє швидкому виявленню та діагностиці помилок, але й дозволяє операторам зосередитись на більш складних завданнях, замість того, щоб витрачати час на рутинне вирішення повсякденних проблем.

Для точнішої оцінки потенційної ефективності системи, компанія надала дані про реальну кількість помилок і час, витрачений на їх виправлення. За середніми даними, що були надані, падіння дошки сталося 6 разів на день і на виправлення кожної інцидентної події витрачалося 5 хвилин із яких 3 на її знаходження; проблеми з масою заготовки траплялися 3 рази на день, з часом виправлення 30 хвилин і її знаходженням 15 хвилин; зупинки витяжки відбувалися 4 рази на день, з часом виправлення 18 хвилин на кожен інцидент і 10 хвилин на її знаходження.

За цими данними ми можемо зробити розрахунки, що на знаходження дошки витрачається приблизно 18 хвилин, на проблеми з масою 45 хвилин, зупинка витяжки 40 звилин. Окремо всі ці помилки не роблять багато втрат, проте разом і систематично, це доходить до 500 хвилин за тиждень. І завдяки усуненню, саме часу знаходження, ми можемо у досить великому обсязі зменшити цей час. Якщо брати загальний час коли дошка впала, ми маємо 30 хвилин і у данній ситуації якщо прибрати час на її знаходження маємо ефективність у 60%. У розгляданні часу на правильність заготовки ми маємо 80 хвилин і ефективність виходить 56%. При зупинці витяжки котра може відбуватися до 72 хвилин з ефективністю 55%.

Завдяки інтелектуальній системі, час на усунення проблеми падіння дошки може бути скорочений до 12 хвилин, час на вирішення невірної маси заготовки може бути знижений до 36 хвилин, а час реакції на зупинки витяжки може бути зменшений до 33 хвилин. Загальна потенційна економія часу становить 43 хвилин на день або 215 хвилин на тиждень, що є значним покращенням у 43% ефективності виробничих процесів.

2 ОГЛЯД АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Python як основна мова програмування

Взагалі, мов програмування досить багато. Більш того, час від часу з'являються нові. Тому природним чином виникає питання: чому саме Python? Тут можна видалити кілька пунктів:

– мова програмування Python - це мова високого рівня, досить "молода", але дуже популярна, яка вже зараз широко використовується на практиці і сфера застосування Python постійно розширюється;

– мова Python бурхливо розвивається. Цьому сприяє не тільки досить вдала концепція мови, але також сформоване згуртоване співтовариство розробників і популяризаторів мови;

– синтаксис мови Python мінімалістичний і гнучкий. На цій мові можна складати прості та ефективні програми;

– стандартна бібліотека для цієї мови містить безліч корисних функцій, що значно полегшує процес створення програмних кодів;

– мова Python підтримує кілька парадигм програмування, включаючи структурну, об'єктно-орієнтовану і функціональну;

– мова Python цілком вдалий вибір для першої мови при навчанні програмуванню;

– важливий і той факт, що необхідне програмне забезпечення, включаючи середовища розробки, в основній своїй масі безкоштовні;

– все це дає підстави розглядати Python в якості одного з найбільш перспективних мов програмування, рисунок 2.1.

Рисунок 2.1 – Зображення логотипу Python

Простота та мінімалістичність програм на Python дає змогу розуміти код, ніби читання англійського тексту, хоча і дуже строго. Така псевдокодова природа Python є однією з його найсильніших сторін. Python надає можливість програмісту зосередитися на розв’язуванні задачі, а не на самій мові. Ядро мови без додаткових модулів досить просте і мінімальне – це надає мові простоти та логічності, що відповідно спрощує її розуміння.

2.2 Рішення для конвертації мови у текст

“Інтелектуальна голосова” такий початок назви системи. Тобто однією з основних задач є синтез та розпізнавання мови. Коли з’явилось питання “як саме впровадити синтез мови?” у мене було декілька варіантів: написати власну нейронну мережу, або знайти готові рішення. При аналізі я отримав дуже цікавий досвід. Розробка та навчання нейронної мережі дуже

захоплююча та й витратна по часу робота. Так як це займає аж занадто багато часу та ресурсів, я звернувся до наявних технічних рішень для синтезу мовлення і була обрана вибірка найпопулярніших Text-to Speech систем, а саме:

– AssemblyAI;
– IBM Watson Speech to Text;
– Amazon Transcribe;
– Google Cloud Speech-to-Text.

Розглянемо кожну із них та спробуємо виділити переваги та недоліки для визначення, яке з цих рішень доцільніше використовувати для вирішення задачі синтезу мовлення, рисунок 2.2.

Рисунок 2.2 – Зображення логотипу AssemblyAI

AssemblyAI – Speech to Text API має точну транскрипцію звуку, використовуючи найновіші дослідження в сфері глибокого навчання. Однією з найбільш важливих особливостей AssemblyAI є застосування декількох різних бібліотек для різних акцентів, якості запису та середовища запису, враховуючи 19 кількість фонових шумів. Лише ця функція робить AssemblyAI

вартою для розгляду, оскільки транскрипція звуку з динаміків із незнайомими акцентами може бути надзвичайно складною та трудомісткою.

Основні переваги AssemblyAI:

– досить висока точність і швидкість;
– приємна ціна;
– проста інтеграція API;
– підтримка багатьох мов програмування.

Основні недоліки AssemblyAI:

– немає точності рівня людини.

IBM Watson Speech to Text хмарне рішення, яке використовує алгоритми глибокого навчання для застосування знань про граматику, структуру мови та композицію звукових сигналів для створення налаштовуваного розпізнавання мови для оптимальної транскрипції тексту, рисунок 2.3.

Рисунок 2.3 – Зображення логотипу IBM Watson Speech to Text

Основні переваги IBM Watson Speech to Text:

– проста інтеграція API;
– точна інтерпретація речення та його контексту.

Основні недоліки IBM Watson Speech to Text:

– не фокусується на мові лише однієї людини. Якщо будь-яка мова розпізнається, IBM Watson Speech to Text намагається це перетворити в текст і це дуже впливає на якість розпізнавання;

– невелика кількість підтримуваних мов.

Amazon Transcribe – сервіс автоматичного розпізнавання мовлення (ASR), який полегшує розробникам впровадження Speech-to-Text у програмні застосунки. Використовуючи API Amazon Transcribe, можна аналізувати аудіофайли, що зберігаються в Amazon S3 і отримувати текстові файли транскрибованої мови, рисунок 2.4.

Рисунок 2.4 – Зображення логотипу Amazon Transcribe

Основні переваги Amazon Transcribe:

– підтримка багатьох аудіоформатів;

– можливість виокремити одного розмовника.

Основні недоліки Amazon Transcribe:

– розпізнавання дуже сильно залежить від контексту;

– якщо розмір аудіосигналу, що подається на вхід невеликий, час обробки доволі довгий.

Google Cloud Speech-to-Text застосовує найсучасніші алгоритми глибокого навчання нейронних мереж Google для автоматичного розпізнавання мови, рисунок 2.5.

Рисунок 2.5 – Зображення логотипу Google Cloud Speech-to-Text

Основні переваги Google Cloud Speech-to-Text:

– підтримка 125 мов та варіантів вимови;

– можливість обрати навчену модель;

– проста інтеграція API.

При аналізі основних недоліків не було знайдено суттєвих недоліків.

Базуючись на цьому аналізі ми робимо висновок, що нашим фаворитом стає Google Cloud Speech-to-Text.

2.3 Vosk як рішення для розпізнавання мовлення

Розпізнаванням мовлення – це процес, що дозволяє комп'ютеру ідентифікувати та реагувати на звуки людської мови. Завдання розпізнавання мови вирішується вже давно, але якісного рівня вдалося досягти лише в останні роки. Загалом, з появою віртуальних помічників та технологічним розвитком смартфонів, міні-комп'ютерів, інших мобільних пристроїв, користувачам стало зручніше взаємодіяти з технікою за допомогою голосових команд ніж за допомогою традиційних інтерфейсів.

Vosk model – це відкрита модель мовлення, що дозволяє розпізнавати мовлення в режимі реального часу. Вона заснована на технології глибокого навчання і може бути використана на різних мовах. Vosk model може бути використана для розпізнавання мовлення у різних ситуаціях, включаючи шумні середовища, рисунок 2.6.

Рисунок 2.6 – Зображення логотипу Vosk

Для програмної реалізації Vosk model у голосових асистентах використовуються різноманітні інструменти та бібліотеки програмування. Наприклад, для реалізації голосового асистента з використанням Vosk model

можна використовувати мову програмування Python та бібліотеку PyAudio для запису та обробки аудіо-сигналу. Також для реалізації Vosk model використовується бібліотеку SpeechRecognition, яка надає можливість розпізнавати мовлення за допомогою різних API, включаючи Vosk. Для цього потрібно буде завантажити модель Vosk та додаткові бібліотеки, які забезпечують її роботу.

Для реалізації відповідей голосового асистента на команди користувачів після розпізнавання мовлення за допомогою моделі Vosk, можна використовувати програмні бібліотеки та сервіси, що дозволяють інтегрувати голосового асистента з різними сервісами та програмами, які користується користувач. Крім того, для поліпшення ефективності та точності роботи голосового асистента з використанням моделі Vosk, можна застосовувати алгоритми машинного навчання та штучного інтелекту. Ці алгоритми допоможуть розпізнавати мовлення та адаптуватися до індивідуальних особливостей голосу користувача, забезпечуючи більшу ефективність та точність роботи асистента.

Точність розпізнавання мовлення є критично важливою для ефективної роботи голосових асистентів. Використання Vosk model може допомогти значно поліпшити точність розпізнавання мовлення та забезпечити більш ефективну роботу голосових асистентів. Наприклад, при тестуванні Vosk model на різних мовах та платформах, було показано, що точність розпізнавання мовлення зросла на 5-10%.

У ситуаціях з шумом точність розпізнавання мовлення зазвичай погіршується. Однак, Vosk model може бути використана для покращення точності розпізнавання мовлення в шумному середовищі. Для цього, Vosk model може бути навчена розпізнавати мовлення в різних шумних умовах, наприклад, в транспорті, в умовах з фоновим шумом, та інших. Це допоможе забезпечити більш ефективну роботу голосових асистентів у різних умовах.

Використання Vosk model допомагає підвищити продуктивність голосових асистентів на різних мовах та платформах. Застосування Vosk model

редукує кількість помилок у розпізнаванні мовлення, що забезпечує більш точні й оперативні відповіді від голосового асистента. Причому, використання Vosk model сприяє поліпшенню роботи голосового асистента в шумних середовищах, роблячи його більш функціональним і зручним для користувачів. Крім того, Vosk model підтримує розпізнавання мов на різних мовних варіантах, включаючи менш поширені мови. Це дозволяє розширити можливості голосових асистентів та забезпечити їх доступність для користувачів з різних країн і регіонів.

2.3 Математична модель FuzzyWuzzy для чіткого знаходження відповідності слів

Однією з головних проблем аналізу текстів є велика кількість слів у документі. Якщо кожне з цих слів аналізувати, то час пошуку нових знань різко зросте і навряд чи буде задовольняти вимогам користувачів. У той же час очевидно, що не всі слова в тексті несуть корисну інформацію. Крім того, в силу гнучкості природних мов формально різні слова, наприклад синоніми, які насправді означають однакові поняття. Таким чином, видалення неінформативних слів, а також приведення близьких за змістом слів до єдиної форми значно скорочують час аналізу текстів. Усунення описаних проблем виконується на етапі попередньої обробки тексту. Зазвичай використовують такі прийоми видалення неінформативних слів і підвищення суворості текстів: видалення стоп-слів. Стоп-словами називаються слова, які є допоміжними і несуть мало інформації про зміст документа. Зазвичай заздалегідь складаються списки таких слів, і в процесі попередньої обробки вони видаляються з тексту. Типовим прикладом таких слів є допоміжні слова і артиклі, наприклад: «так як», «крім того», тощо.

Стемінг – морфологічний пошук. Він полягає в перетворенні кожного слова до його нормальної формі. Нормальна форма виключає схиляння слова, множинну форму, особливості усного мовлення. Наприклад, слова

«стиснення» і «стислий» повинні бути перетворені в нормальну форму слова «стискати». Алгоритми морфологічного розбору враховують мовні особливості і внаслідок цього утворюють мовнонезалежний алгоритм. N-грами – це альтернатива морфологічному розбору і видалення стопслів, рисунок 2.7.

Рисунок 2.7 – Зображення логотипу FuzzyWuzzy

N-грами – це частина рядка, що складається з N символів. Наприклад, слово «день» може бути представлено 3-грамою «_Де», «ден», «ень», «нь_», або 4-грамою «_ден», «день», «ень_», де символ підкреслення заміняє попередній або замикає слово пробіл. У порівнянні зі стемінг або видаленням стоп-слів, N-грами менш чутливі до граматичним і типографським помилок. Крім того, N-грами не вимагають лінгвістичного подання слів, що робить

даний прийом більш незалежним від мови. Однак N-грами, дозволяючи зробити текст більш суворим, не вирішують проблему зменшення кількості неінформативних слів. Приведення регістра. Цей прийом полягає в перетворенні всіх символів до верхнього або нижнього регістру. Наприклад, всі слова «текст», «Текст», «ТЕКСТ» наводяться до нижнього регістру «текст». Найбільш ефективно спільне застосування цих методів.

Левенштейн В.Й. розробив алгоритм, який дозволяє оцінити, наскільки подібний один рядок на другий. Алгоритм Левенштейна дозволяє отримати саме чисельну оцінку подібності рядків. Відстань Левенштейна в теорії інформації та комп'ютерної лінгвістики – це міра різниці двох послідовностей символів (рядків) щодо мінімальної кількості операцій вставки, видалення і заміни, необхідних для переведення одного рядка в інший. Основна ідея алгоритму полягає в тому, щоб порахувати мінімальну кількість операцій видалення, вставки і заміни, які потрібно зробити над одним з рядків, щоб отримати другий.

Нехай *S1* і *S2* – два рядки довжиною *M* і *N* відповідно, тоді відстань Левенштейна *d*(*S1*,*S2*) обчислюється за формулою *d*(*Sl, S2*) = *D* (*M, N*), при цьому елементи, за формулою.

$$D(i,j)=\begin{cases}0 & якщо\ i=1, j=1,\\ i & якщо\ i>1, j=1,\\ j & якщо\ i=1, j>1,\\ \min(& якщо\ i>1, j>1,\\ D(i,j-1)+1,\\ D(i-1,j)+1,\\ D(i-1,j-1)+m(S_1[i],S_2[j])),\end{cases}$$

$$\text{де}\quad m(S_1[i],S_2[j])=\begin{cases}0, & якщо\ S_1[i]=S_2[j]\\ 1, & якщо\ S_1[i]\neq S_2[j]\end{cases}$$

(2.1)

Розглянемо формулу (2.1) більш детально. Тут крок по *i* відповідає за видалення (*D*) з першого рядка, по *j* – вставку (*I*) в перший рядок, а крок заобома індексами відповідає за заміну символу (*R*) або відсутність змін (*M*). Очевидно, що відстань між двома порожніми рядками дорівнює нулю. Так само очевидно те, щоб отримати порожній рядок з рядка довжиною *i*, потрібно зробити *i* операцій видалення, а щоб отримати рядок довжиною *j* з порожнього, потрібно провести *j* операцій вставки. У нетривіальному випадку потрібно вибрати мінімальну «вартість» з трьох варіантів. Вартість «вставка/видалення» буде в будь – якому випадку становитиме одну операцію, а от заміна може не знадобитися, якщо символи рівні - тоді крок за обома індексами «безкоштовний».

Очевидно, справедливі наступні твердження:

– $d(S_1, S_2) \geq \big||S_1| - |S_2|\big|$;
– $d(S_1, S_2) \leq max(|S_1|, |S_2|)$;
– $d(S_1, S_2) = 0 \leftrightarrow S_1 = S_2$.

Побудуємо матрицю перетворення Текст в Тост. Спочатку матриця виглядає, як показано в табл. 2.1.

Таблиця 2.1 – Початковий стан матриці

		Т	о	с	т
	0	1	2	3	4
Т	1				
е	2				
к	3				
с	4				
т	5				

Проставлення значень у матриці відбувається занаступною формулою:

$$a(i,j) = \min\big(a(i+l,j)+l; a(i,j-l)+l; a\,(i-l,j-l) + if(S_1[i] = S_2[j], 0, l)\big),$$

$if(S_1[i] = S_2[j], 0, l)$ – повертає 0, якщо літери, які стоять у відповідних позиціях однакові, 1 – якщо відрізняються. Отже, щоб отримати значення елемента, потрібно знати значення його сусідів зверху, знизу і по діагоналі. Для елемента a(1,1) отримаємо:

$a(l,l) = \min(a(0,l)+l, a(1,0)+1, a(0,0) + if(S_1[1] = S_2[1], 0, l))$,

$a(1,1) = \min(2; 2; 0 + if('T' = 'T', 0,1)) = \min(2; 2; 0) = 0$

$a(1,2) = \min\big(a(0,2)+1; a(1,1)+1; a(0,1) + if('T' = 'O', 0, l)\big)$,

...і тд.

В результаті отримуємо матрицю (табл.1.2), значення елемента відстані Левенштейна від S1=Тост до S2=Текст.

Таблиця 2.2 – Заповнена матриця

		Т	о	с	т
	0	1	2	3	4
Т	1	0	1	2	3
е	2	1	1	2	3
к	3	2	2	2	3
с	4	3	3	2	3
т	5	4	4	3	2

Функція Левенштейна відіграє роль фільтра, свідомо відкидає неприйнятні варіанти (у яких значення функції більше деякої заданої константи). З точки зору додатків визначення відстані між словами або текстовими полями по Левенштейну має наступні недоліки: при перестановці місцями слів або частин слів виходять порівняно великі відстані; відстані між абсолютно різними короткими словами виявляються невеликими, в той час як відстань між сильно схожими довгими словами виявляються значними.

2.5 Алгоритм роботи програмного забезпечення

2.5.1 Ініціалізація підсистеми друку

Почнемо з модуля “printer”, який відповідає за функціонування підсистеми друку. Перше, що відбувається в цьому модулі, це виклик функції initialize_printer(). Коли ця функція виконується, програма виводить повідомлення "Printer initialized.", що свідчить про успішну підготовку принтера до роботи. Ця функція викликається для того, щоб переконатися, що всі необхідні компоненти принтера налаштовані і готові приймати команди.

Другою важливою функцією цього модуля є print_command(command), яка безпосередньо відповідає за друк. Коли ця функція викликається з певною командою як параметром, вона виводить повідомлення “Printing command: “ з текстом команди, що сигналізує про те, що принтер обробляє і друкує.

2.5.2 Ініціалізація і запуск головної програми

Перейдемо до головного файлу програми “main”, який є центральною точкою управління всіма модулями та підсистемами. Першим кроком в цьому файлі є імпорт всіх необхідних модулів: kommanden, logs, Fuzz, input_voice, tts, і printer. Кожен з цих модулів відповідає за певний аспект роботи програми і виконує свою специфічну функцію.

Після імпорту модулів викликається функція logs.start_logging(), яка налаштовує систему логування. Логування є важливим інструментом для моніторингу і діагностики роботи програми. Налаштування включає рівень логування DEBUG, що означає, що всі важливі події, включаючи налагоджувальні повідомлення, будуть записані.

Наступним кроком є завантаження команд, що здійснюється викликом функції kommanden.load_commands(). Ця функція відкриває файл commands.json і завантажує його вміст у глобальну змінну commands. Це дозволяє програмі знати, які команди вона може розпізнавати і обробляти під час своєї роботи.

Після цього програма ініціалізує всі підсистеми. Функція Fuzz.initialize_fuzzing() відповідає за ініціалізацію модуля Fuzz, який обробляє команди. Функція input_voice.initialize_input() ініціалізує підсистему голосового вводу, яка буде приймати команди від користувача. Далі функція tts.initialize_tts() ініціалізує систему тексту в мову (TTS), яка буде озвучувати помилки та рішення.

2.5.3 Основний цикл програми

Після ініціалізації всіх підсистем програма входить у нескінченний цикл while True, який буде виконуватися до тих пір, поки користувач не перерве виконання програми. У цьому циклі виконуються наступні дії:

По-перше, функція input_voice.get_input() запитує у користувача голосову команду. Вона очікує, поки користувач не введе текстову команду через мікрофон або клавіатуру і повертає цей текст для подальшої обробки.

Отриманий текст передається до функції kommanden.parse_command(input_text), яка порівнює введений текст з доступними командами. Якщо знайдена відповідна команда, вона повертається для подальшої обробки. Якщо команда не знайдена, повертається None.

Якщо команда знайдена, виконується її обробка. Спочатку функція Fuzz.process_command(command) обробляє команду, виводячи повідомлення "Processing command: " з текстом команди. Потім викликається функція tts.speak_command(command), яка озвучує команду, виводячи повідомлення "Speaking command: " з текстом команди.

2.5.4 Завершення роботи програми

Програма постійно виконує основний цикл, приймаючи нові команди, поки не отримає сигнал про завершення роботи. Якщо користувач натискає Ctrl+C або іншим способом перериває виконання програми, обробляється

виключення KeyboardInterrupt. В цьому випадку програма виводить повідомлення "Program terminated by user." і завершує свою роботу.

3 РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ ГОЛОСОВОЇ СИСТЕМИ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ

На початку свого шляху розробки інтелектуальної системи підтримки прийняття рішень, я спідкнувся з проблемами, які виникають при написанні всього коду в одному файлі. Такий підхід привів мене до заплутаного і важкого розуміння коду, що значно ускладнило процес підтримки та розвитку програми. Наприклад, виправлення помилок може займати день, а іноді й до тижня, оскільки навіть незначні зміни в одній частині коду можуть вплинути на функціонування всієї програми.

Під час пошуку інформації, для оптимізації роботи програми, я пізнав таке рішення як “модульне програмування”. Воно дозволяє розділяти код на логічні частини, кожна з яких відповідає за конкретний функціонал. Модульний підхід значно полегшує процес розробки, оскільки кожен модуль має чітко визначене призначення і незалежний від інших частин програми. Наприклад, модуль для роботи з базою даних може бути відокремлений від модуля для обробки користувацького інтерфейсу, що робить код більш структурованим та зрозумілим.

Одна з ключових переваг модульного підходу полягає у можливості повторного використання коду. Модуль, створений для одного проекту, може бути легко використаний в іншому, що дозволяє заощадити час і зусилля, а також забезпечити узгодженість у використанні одного й того ж коду в різних проектах. Тестування модулів також набагато простіше, ніж тестування великого монолітного файлу коду. Модулі можуть бути протестовані окремо, що дозволяє швидко виявити і виправити помилки.

Модульний підхід також робить програму більш гнучкою, оскільки зміни в одному модулі не впливають на інші модулі. Це дозволяє легко змінювати і розширювати функціонал програми. Наприклад, можна легко замінити один модуль на інший, якщо виникає потреба у зміні способу

виконання певних задач, без необхідності переписувати всю програму. Модулі дозволяють інкапсулювати дані і методи, роблячи їх недоступними ззовні. Це забезпечує кращу безпеку і захист даних, оскільки доступ до них мають тільки ті частини програми, які дійсно потребують цього доступу.

Таким чином, використання модулів при написанні коду для інтелектуальної голосової системи підтримки прийняття рішень принесло багато переваг, включаючи поліпшення організації, підвищення повторного використання коду, зручність у тестуванні, полегшення командної роботи, підтримки та оновлення, підвищення гнучкості. Все це зробило розробку програмного забезпечення більш ефективною і зручною. Тому модульний підхід є став фаворитом у моїй розробці програмного забезпечення.

3.1 Головний модуль

Головний модуль моєї інтелектуальної голосової системи підтримки прийняття рішень виконує роль оркестратора, керуючи взаємодією між усіма іншими компонентами системи. Цей модуль інтегрує введення від користувача, аналіз запитів, виведення відповідей та логування, забезпечуючи плавне та ефективне функціонування системи.

Для початку буде наведено повний код “main” файлу:

```
# main.py

from tts import init_tts_client
from logs import check_logs, get_logs, checknout
import concurrent.futures
import json

# Глобальний клієнт TTS
tts_client = init_tts_client()
```

```
logs = get_logs(url="http://127.0.0.1:5000/logs")

def handle_recognized_text(text):
    # Тут обробляйте розпізнаний текст
    print("Розпізнано:", text)

def callback_function(result):
    result_json = json.loads(result)
    if 'text' in result_json:
        recognized_text = result_json['text']

def listen_and_respond():

    with concurrent.futures.ThreadPoolExecutor() as executor:
        future = executor.submit(check_logs)
        result = future.result()  # Отримуємо результат від потоку

        # Запускаємо checknout, якщо результат check_logs == True
        if result:
            checknout()



if __name__ == "__main__":

    listen_and_respond()
```

Для кращого розуміння цього коду я розгляну кожний його фрагмент окремо.

Імпорти бібліотек і модулів:

– “from tts import init_tts_client” – цей рядок імпортує функцію init_tts_client з модуля tts. Функція init_tts_client призначена для ініціалізації клієнта Text-to-Speech (TTS), який буде використовуватися для перетворення тексту в аудіо. Цей клієнт реалізований за допомогою API, у моєму випадку, “Google Cloud Text-to-Speech”;

– “from logs import check_logs, get_logs, checknout” – цей рядок імпортує три функції з модуля logs;

– “check_logs” – функція для перевірки логів на наявність певних подій або помилок;

– “get_logs” – функція, яка витягує логи з певного джерела, у цьому випадку, з веб-сервера;

– “checknout” – може використовуватися для ініціювання певних дій на основі результатів перевірки логів, таких як надсилання сповіщень або ескалація інцидентів;

– “import concurrent.futures” – імпортує модуль concurrent.futures, який дозволяє виконувати асинхронне програмування за допомогою високорівневого інтерфейсу для асинхронного запуску завдань. Це особливо корисно для реалізації многопоточності без необхідності низькорівневого управління потоками;

– “import json” – імпортує модуль json, який використовується для кодування та декодування даних у форматі JSON. JSON (JavaScript Object Notation) є легким форматом обміну даними, який легко читається та пишеться людиною та легко аналізується та генерується машинами.

Ініціалізація глобальних змінних, рисунок 3.1:

```
tts_client = init_tts_client()
```

Рисунок 3.1 – Змінна тексту у мову

– цей рядок викликає функцію init_tts_client, яка ініціалізує клієнт TTS. Цей клієнт використовується для перетворення тексту в мовлення у

системі. Така ініціалізація включає налаштування API-ключів, конфігураційних параметрів з вибраним TTS сервісом;

– викликає функцію get_logs, передаючи їй URL як параметр. Ця функція отримує дані логів від сервера, розташованого за цією адресою. Використання URL з 127.0.0.1 вказує на локальний сервер, що є типовим для тестування та розробки. Дані, отримані з цього URL, будуть у форматі JSON та містять інформацію: записи подій, помилок тощо, рисунок 3.2;

```
    logs = get_logs(url="http://127.0.0.1:5000/logs")
```

Рисунок 3.2 – Змінна отримання логів

– ця функція призначена для обробки тексту, який був розпізнаний системою. Вона приймає текст як аргумент і виводить його на консоль з попередженням "Розпізнано". Ця проста реалізація використовується для відправлення тексту до інших системних модулів для подальшої обробки та відповіді, рисунок 3.3;

```
    def handle_recognized_text(text):
# Тут обробляйте розпізнаний текст
print("Розпізнано:", text)
```

Рисунок 3.3 – Функція обробки розпізнаного тексту

– ця функція використовується як зворотний виклик (callback) для обробки результатів, отриманих з асинхронних операцій та інших запитів. Вона перетворює рядок JSON у словник Python і перевіряє наявність ключа 'text'. Якщо такий ключ існує, витягує відповідний текст для подальшої обробки, рисунок 3.4;

```
    def callback_function(result):
result_json = json.loads(result)
if 'text' in result_json:
    recognized_text = result_json['text']
```

Рисунок 3.4 – Функція зворотного виклику

– ця функція використовує ThreadPoolExecutor для асинхронного виконання функції check_logs, яка перевіряє стан системних журналів. Використання асинхронності дозволяє системі залишатися відгукованою та ефективно обробляти інші завдання під час очікування відповіді. Якщо check_logs повертає True, активується функція checknout, яка може виконувати подальші дії, такі як відправлення сповіщень, рисунок 3.5.

```
    def listen_and_respond():

with concurrent.futures.ThreadPoolExecutor() as executor:
    future = executor.submit(check_logs)
    result = future.result()  # Отримуємо результат від потоку

    # Запускаємо checknout, якщо результат check_logs == True
    if result:
        checknout()
```

Рисунок 3.5 – Функція прослуховування та відповіді

3.2 Модуль для виконання перетворення тексту у мову

Цей модуль реалізує функціональність синтезу мови (Text-to-Speech, TTS) за допомогою сервісу Google Cloud Text-to-Speech. Код організований у функції для ініціалізації клієнта TTS та для виконання синтезу мовлення з подальшим відтворенням аудіо.

Загальний код “tts” модулю:

```
    # Модель для синтезу мови
from google.cloud import texttospeech
import os
import sounddevice as sd
import numpy as np

#шлях до мого облікового запису гугл клауд
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] =
```

"D:\Projects\IVsSystem\IVS_CODE.json"

```
# Ініціалізація клієнта Text-to-Speech
def init_tts_client():
    return texttospeech.TextToSpeechClient()
def text_to_speech(client, ssml_text):
    # Використання SSML для включення пауз
    # ssml_text = "<speak>" + text.replace(".", ".<break time='1000ms'/>") +
"</speak>"
    synthesis_input = texttospeech.SynthesisInput(ssml=ssml_text)
    # Конфігурація голосу
    voice = texttospeech.VoiceSelectionParams(language_code="uk-UA",
ssml_gender=texttospeech.SsmlVoiceGender.NEUTRAL)
    # Конфігурація аудіо (зі змінною швидкості мовлення)
    audio_config =
texttospeech.AudioConfig(audio_encoding=texttospeech.AudioEncoding.LINEAR
16, speaking_rate=0.8, pitch=1.0)
    # Запит на синтез мовлення
    response = client.synthesize_speech(input=synthesis_input, voice=voice,
audio_config=audio_config)
    # Конвертація аудіо відповіді в формат NumPy
    audio_data = np.frombuffer(response.audio_content, dtype=np.int16)
    # Відтворення аудіо
    sd.play(audio_data, 24000)# 24000 - частота дискретизації
#
    sd.wait()  # Чекаємо завершення відтворення
```

a) Огляд кожної його частини;

1) імпорти та налаштування середовища, рисунок 3.6;

```
from google.cloud import texttospeech
```

Рисунок 3.6 – Імпорт “TTS”

– імпортується модуль texttospeech з пакету google.cloud, який надає доступ до сервісів Google Cloud Text-to-Speech. Він дозволяє мені використовувати різні класи та функції для створення запитів до Google Cloud Text-to-Speech API. Це дуже важливо для розробки додатків, які потребують перетворення тексту в мовлення, забезпечуючи різноманітні голоси, мови та інші параметри мовлення;

– імпортується стандартний модуль Python os, який надає функції для взаємодії з операційною системою. В даному контексті, цей модуль використовується для налаштування змінних середовища, що є критично важливим для конфігурації та автентифікації в Google Cloud сервісах. Це включає доступ до системних шляхів, змінних середовища та інших системних параметрів, рисунок 3.7;

```
import os
```

Рисунок 3.7 – Імпорт “Python OS”

– імпортується бібліотека sounddevice, яка забезпечує простий інтерфейс для відтворення та запису звуку. У даному коді, sounddevice використовується для відтворення аудіо даних, отриманих з сервісу Google Cloud Text-to-Speech. Це дає змогу не тільки синтезувати мовлення, але й відразу ж відтворювати його через аудіосистему пристрою, рисунок 3.8;

```
import sounddevice as sd
```

Рисунок 3.8 – Імпорт “Sounddevice”

– імпортується бібліотека numpy, відома своєю широкою підтримкою операцій з багатовимірними масивами та матрицями, а також з

великим набором математичних функцій. В контексті коду, numpy використовується для обробки аудіо даних. Зокрема, для перетворення бінарних аудіо даних, отриманих від Google Cloud Text-to-Speech, у масиви numpy, що полегшує їх відтворення та подальшу обробку, рисунок 3.9;

```
import numpy as np
```

Рисунок 3.9 - Імпорт “Numpy”

– ця частина налаштовує змінну середовища GOOGLE_APPLICATION_CREDENTIALS, що вказує на шлях до файлу JSON, який містить ключі доступу до Google Cloud. Встановлення цієї змінної є необхідною передумовою для автентифікації в сервісах Google Cloud. Ключі в цьому файлі дозволяють програмі безпечно взаємодіяти з API Google Cloud, зокрема з Text-to-Speech, гарантуючи, що використання API відбувається в рамках встановлених дозволів та квот, рисунок 3.10;

```
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] =
"D:\Projects\IVsSystem\IVS_CODE.json"
```

Рисунок 3.10 – Змінна для налаштування

2) Функції.

Ця функція створює та повертає об'єкт клієнта Google Cloud Text-to-Speech. Вона є вступним кроком для подальшого використання API TTS, дозволяючи програмі надсилати текстові дані для синтезу мовлення, рисунок 3.11.

```
def init_tts_client():
  return texttospeech.TextToSpeechClient()
```

Рисунок 3.11 – Функція клієнта “Google cloud Text-to-Speech”

Функція “text_to_speech”:

```
    def text_to_speech(client, ssml_text):
    # Використання SSML для включення пауз
    # ssml_text = "<speak>" + text.replace(".", ".<break time='1000ms'/>") + "</speak>"
    synthesis_input = texttospeech.SynthesisInput(ssml=ssml_text)
    # Конфігурація голосу
    voice = texttospeech.VoiceSelectionParams(language_code="uk-UA", ssml_gender=texttospeech.SsmlVoiceGender.NEUTRAL)
    # Конфігурація аудіо (зі змінною швидкості мовлення)
    audio_config = texttospeech.AudioConfig(audio_encoding=texttospeech.AudioEncoding.LINEAR16, speaking_rate=0.8, pitch=1.0)
    # Запит на синтез мовлення
    response = client.synthesize_speech(input=synthesis_input, voice=voice, audio_config=audio_config)
    # Конвертація аудіо відповіді в формат NumPy
    audio_data = np.frombuffer(response.audio_content, dtype=np.int16)
    # Відтворення аудіо
    sd.play(audio_data, 24000)# 24000 - частота дискретизації
#
    sd.wait()   # Чекаємо завершення відтворення
```

Функція text_to_speech є ключовим компонентом системи синтезу мови, що перетворює вхідний текстовий формат SSML (Speech Synthesis Markup Language) на аудіо. Вона використовує сервіси Google Cloud Text-to-Speech для генерації звуку на основі вказаних параметрів. Розберемо детально кожен крок у функції, рисунок 3.12:

```
def text_to_speech(client, ssml_text):
```

Рисунок 3.12 – Визначення функції “text_to_speech”

– визначення функції text_to_speech, яка приймає два аргументи: client (клієнт Google Text-to-Speech) та ssml_text (текст у форматі SSML для синтезу);

– створення об'єкта SynthesisInput, який пакує SSML текст для подальшої обробки сервісом Text-to-Speech. SSML дозволяє вказати різні аспекти мовлення, такі як інтонацію, паузи, тональність, швидкість, що робить виголошення більш природним та розбірливим, рисунок 3.13;

```
synthesis_input = texttospeech.SynthesisInput(ssml=ssml_text)
```

Рисунок 3.13 – Створення об'єкта “SynthesisInput”

– визначення параметрів голосу через клас VoiceSelectionParams. Задається код мови (language_code="uk-UA" для української мови) та гендер голосу (ssml_gender з встановленою на "NEUTRAL"), що дозволяє вибрати більш нейтральний тон голосу, рисунок 3.14;

```
voice = texttospeech.VoiceSelectionParams(language_code="uk-UA",
ssml_gender=texttospeech.SsmlVoiceGender.NEUTRAL)
```

Рисунок 3.14 – Визначення параметрів голосу

– налаштування конфігурації аудіо через AudioConfig, де визначаються технічні параметри аудіовиходу. Вказується тип кодування (LINEAR16), швидкість мовлення (speaking_rate=0.8 що є дещо повільніше за стандартну швидкість), та висота тону (pitch=1.0, стандартний рівень), рисунок 3.15;

```
audio_config = texttospeech.AudioConfig(audio_encoding=texttospeech.AudioEncoding.LINEAR16, speaking_rate=0.8, pitch=1.0)
```

Рисунок 3.15 – Налаштування конфігурації аудіо

– виконання запиту на синтез мови, де client.synthesize_speech викликається з заданими параметрами вводу, голосу, та аудіо конфігурації. Відповідь з сервера містить аудіо контент, який можна відтворити, рисунок 3.15;

```
response = client.synthesize_speech(input=synthesis_input, voice=voice, audio_config=audio_config)
```

Рисунок 3.16 – Виконання запиту на синтез мови

– конвертація аудіо контенту в масив NumPy. np.frombuffer читає буфер, такий як бінарні дані аудіо відповіді, і перетворює їх у масив NumPy типу np.int16. Цей крок необхідний для подальшого відтворення звуку, рисунок 3.17;

```
audio_data = np.frombuffer(response.audio_content, dtype=np.int16)
```

Рисунок 3.17 – Конвертація аудіо контенту в масив “NumPy”

– відтворення аудіо за допомогою бібліотеки sounddevice. Аудіо дата відтворюється на частоті 24000 Гц, що є стандартною частотою для голосового мовлення, забезпечуючи якісне відтворення, рисунок 3.18;

```
sd.play(audio_data, 24000)
```

Рисунок 3.18 – Відтворення аудіо за допомогою бібліотеки sounddevice

– ця команда зупиняє виконання подальшого коду до завершення відтворення аудіо, гарантуючи, що весь аудіо контент буде повністю відтворений перш ніж програма продовжить виконання, рисунок 3.19.

```
sd.wait()
```

Рисунок 3.19 – Зупиняє виконання коду

3.3 Модуль логування

Модуль логування містить кілька функцій для управління логами, обробки тексту заслухань, та інтеграції з Text-to-Speech сервісом для зворотного зв'язку користувачеві. Основні компоненти скрипта включають роботу з HTTP запитами, паралельну обробку, і взаємодію з користувачем через голос.

```
# logs.py


import requests
import time
from threading import Event
from input_voice import va_listen
import Fuzz
from tts import text_to_speech, init_tts_client
import kommanden

# Глобальна подія для сигналізації про завершення
log_thread_finished = Event()

# URL маршруту логів на вашому Flask-сервері
```

```
log_url = "http://127.0.0.1:5000/logs"


def process_recognized_text(text):
    category = Fuzz.process_command(text)
    if category:
        kommanden.execute(category)
    else:
        print("Команда не розпізнана.")

def get_logs(url):
    try:
        response = requests.get(url)
        response.raise_for_status()  # Перевірка на наявність помилок HTTP
        return response.json()  # Повернення логів у форматі JSON
    except requests.RequestException as e:
        print(f"Помилка при запиті: {e}")
        return None

def checknout(): #Функція для обробки останнього логу, прослуховування мікро та виведення інформації щодо вирішення помилки
    va_listen(process_recognized_text)


def check_logs():
    error_detected = False

    # Оновлення логів до тих пір, поки printing не стане False
    while error_detected == False:
```

```
        global log_thread_finished
        tts_client = init_tts_client()
        logs = get_logs(log_url)

        if logs is not None:
            # Перевірка останнього запису в логах
            latest_log = logs[-1]
            printing_status = latest_log.get("printing")
            error_status = latest_log.get("error")
            current_tape_length_status = latest_log.get("length")

           # print(f"Latest log - Printing status: {printing_status},"
            #      f" Error status: {error_status},"
              #   f" length: {current_tape_length_status}")

            if current_tape_length_status == "50":
                # Якщо залишилось 10 відсотків
                text_to_speech(tts_client, f"Залишилось 50%:
{current_tape_length_status}")
                print("Залишилось 50%")

            # Переривання циклу, якщо error має помилку
#
            if logs and any(log.get("printing") is False for log in logs):
                # Зупинка слухання та виведення питання



                response = input("Сталася помилка, чи хочете ви продовжити? (Y/N):
")
                if response.lower() == 'y':
```

```
                if error_status == "Tape out":
                    # Якщо скінчилась стрічка
                    text_to_speech(tts_client, f"Скінчилась стрічка:
{current_tape_length_status}")




                if error_status == "Error":
                    # Якщо є помилка, відправити повідомлення через TTS
                    text_to_speech(tts_client, f"Помилка принтера: {error_status}")

                    error_detected = True
                    return True
            else:
                log_thread_finished.set()
                return False



            # elif response.lower() == 'n':
            #     log_thread_finished.set()

        else:
            print("No logs received.")

        if error_detected == False:
            time.sleep(5)  # Затримка перед наступним запитом, якщо помилка не
була знайдена
```

 # log_thread_finished.set()

Огляд кожної його частини:

– Імпорти та Налаштування Середовища, рисунок 3.20;

```
import requests
```

Рисунок 3.20 – Імпорт бібліотеки “requests”

– імпортує бібліотеку requests, яка є потужною і зручною бібліотекою для відправки HTTP запитів в Python. Вона використовується для легкої інтеграції з веб-API. requests використовується для здійснення запитів до сервера для отримання або відправки даних, таких як логи або стан системи;

– імпортує модуль time, який надає функції для роботи з часом, включаючи функцію sleep, яка зупиняє виконання коду на заданий період часу. Використовується для створення затримок у виконанні програми, у моєму випадку, при періодичних перевірках стану сервера та обробці подій, рисунок 3.21;

```
import time
```

Рисунок 3.21 – Імпорт модулю “time”

– імпортує клас Event з модуля threading, який дозволяє управління подіями в многопоточних програмах. Event використовуєтьсядля сигналізації між потоками, де один потік чекає на подію, а інший потік сигналізує про цю подію, рисунок 3.22;

```
from threading import Event
```

Рисунок 3.22 – Імпорт класу “Event”

– імпортує функцію va_listen з модуля input_voice. Це функція, яка активує розпізнавання мови та слухає вхідні команди голосом, рисунок 3.23;

```
from input_voice import va_listen
```

Рисунок 3.23 – Імпорт функції “va_listen”

– імпортує модуль Fuzz, який містить функції для фаззі-пошуку та обробки нечітких даних. Модуль Fuzz використовується для аналізу тексту, введеного користувачем, і знаходження найкращих відповідностей або команд у випадку, коли введення є невизначеним або нечітким, рисунок 3.24;

```
import Fuzz
```

Рисунок 3.24 – Імпорт модулю “Fuzz”

– імпортує функції text_to_speech та init_tts_client з модуля tts. Ці функції використовуються для ініціації клієнта TTS (Text-to-Speech) та перетворення тексту в мовлення відповідно. init_tts_client створює екземпляр клієнта, який звертається до TTS сервісів, а text_to_speech використовується для генерації мовлення з тексту, що використовується для зворотного зв'язку користувачу або ведення діалогу, рисунок 3.25;

```
from tts import text_to_speech, init_tts_client
```

Рисунок 3.25 – Імпорт функцій “text_to_speech” та “init_tts_client”

– імпортує модуль kommanden, який, містить різні відповіді, команди та скрипти, що можуть бути викликані при знаходжені помилки, рисунок 3.26.

```
import kommanden
```

Рисунок 3.26 – Імпорт модулю “kommanden”

– глобальні змінні, рисунок 3.27;

```
log_thread_finished = Event()
```

Рисунок 3.27 – Змінна для сигналізації

– log_thread_finished = Event(): створює подію, яка використовується для сигналізації про завершення роботи потоку, що обробляє логи;

– log_url: змінна, що зберігає URL, з якого функція get_logs буде збирати дані логів. URL вказує на локальний сервер, де може бути розгорнуто Flask-додаток для тестування або розробки, рисунок 3.28.

```
log_url = "http://127.0.0.1:5000/logs"
```

Рисунок 3.28 – Змінна, що зберігає URL

– функції, рисунок 3.29;

```
def process_recognized_text(text):
    category = Fuzz.process_command(text)
    if category:
```

```
    kommanden.execute(category)
else:
    print("Команда не розпізнана.")
```

Рисунок 3.29 – Функція “process_recognized_text”

Функція process_recognized_text є важливою частиною системи обробки голосового вводу, призначена для інтерпретації тексту, розпізнаного з голосу, і виконання відповідних дій на основі цього тексту. Для подальшої роботи з ним я, детально проаналізую кожен рядок цієї функції, щоб зрозуміти її функціональність та призначення, рисунок 3.30;

```
def process_recognized_text(text):
```

Рисунок 3.30 – Визначає функції

– визначає функцію process_recognized_text, яка приймає один параметр text. Цей текст є рядком, що містить слова, розпізнані системою розпізнавання голосу. Ця функція є містком між модулем розпізнавання голосу та модулями обробки команд, виконуючи роль обробника тексту, що підлягає подальшому аналізу та виконанню. Вона використовується в системах, де голосовий ввід перетворюється на текстові команди, які потім обробляються для виконання певних дій;

– викликає функцію process_command з модуля Fuzz, передаючи їй розпізнаний текст. Модуль Fuzz використовує алгоритми фаззі-логіки та методи для визначення, яка команда (або 'категорія' дій) відповідає введеному тексту. Це дозволяє згладжувати можливі помилки розпізнавання або нечіткості у визначеному тексті. Результатом є ідентифікація категорії дії, яку слід виконати, на основі тексту введеного користувачем. Це критично важливо для систем, де команди мають бути точно класифіковані для коректного виконання, рисунок 3.31;

```
category = Fuzz.process_command(text)
```

Рисунок 3.31 – Викликання функції “process_command”

– блок умови, що перевіряє, чи було успішно ідентифіковано категорію команди. Якщо category не є None або порожнім (що означає успішне визначення команди), викликається функція execute з модуля kommanden, що має виконати визначену дію. Якщо жодної відповідної категорії не знайдено, виводиться повідомлення "Команда не розпізнана", сигналізуючи про неуспіх у визначенні дій, що мають бути виконані. Цей механізм критично важливий для забезпечення правильної реакції системи на ввод користувача. Він забезпечує, що тільки валідні та розпізнані команди викликають дії в системі, тоді як нерозпізнані або помилкові команди ігноруються або обробляються відповідно, рисунок 3.32.

```
    if category:
  kommanden.execute(category)
else:
  print("Команда не розпізнана.")
```

Рисунок 3.32 – Блок умови

Наступна функція є ключовою компонентою системи обробки логів, використовуючи HTTP запити для отримання даних логів з веб-сервера. Ця функція має на меті отримати інформацію у форматі JSON, що містить записи подій або статусів системи, рисунок 3.33.

```
    def get_logs(url):
  try:
    response = requests.get(url)
    response.raise_for_status()  # Перевірка на наявність помилок HTTP
```

```
        return response.json()  # Повернення логів у форматі JSON
    except requests.RequestException as e:
        print(f"Помилка при запиті: {e}")
        return None
```

Рисунок 3.33 – Функція системи обробки логів

Детальний огляд цієї функції, рисунок 3.34:

```
    def get_logs(url):
```

Рисунок 3.34 – Визначення функції “get_logs”

– визначення функції get_logs, яка приймає один аргумент - url. Це URL-адреса, з якої функція буде намагатися отримати логи. Ця функція створена для того, щоб відправляти HTTP GET запити до вказаного URL для отримання даних логів. Вона є універсальною та може бути використана для звернення до будь-якого веб-сервера, що підтримує такі запити. Використовується у сценаріях, де потрібно автоматично отримувати дані з веб-сервісів, наприклад, для моніторингу стану апаратних засобів, веб-аплікацій, та інших систем, що ведуть логи;

– використання бібліотеки requests для відправки HTTP GET запиту. Метод get відправляє запит до вказаної URL-адреси та зберігає відповідь у змінній response. response.raise_for_status(): Метод, що викликається для перевірки відповіді на наявність помилкових HTTP статус-кодів (таких як 404, 500 і т.д.). Якщо такі коди зустрічаються, метод генерує виняток, який може бути оброблений в блоку except, рисунок 3.35;

```
    try:
        response = requests.get(url)
        response.raise_for_status()  # Перевірка на наявність помилок HTTP
        return response.json()  # Повернення логів у форматі JSON
```

Рисунок 3.35 - використання бібліотеки “requests”

– return response.json(): перетворює тіло відповіді з формату JSON у Python-словник або список і повертає ці дані. Це дозволяє легко обробляти дані логів, які, як правило, структуровані у форматі JSON. Такий підхід дозволяє надійно обробляти відповіді від веб-серверів, відслідковуючи помилки та забезпечуючи коректну обробку даних;

– блок except перехоплює будь-які винятки, що виникають під час виконання запиту, зокрема помилки з'єднання, відповіді з помилками, часові перевищення та інші типи помилок, які можуть виникнути при взаємодії з веб-сервером. Забезпечує стійкість функції до помилок, дозволяючи програмі продовжувати роботу навіть при невдалих спробах доступу до сервера. Вивід повідомлення про помилку допомагає в діагностиці проблем. Важливий для додатків, які залежать від віддалених даних. Обробка винятків дозволяє розробникам уникати збоїв програми і надати користувачам корисну інформацію про проблеми з доступом до даних, рисунок 3.36.

```
    except requests.RequestException as e:
print(f"Помилка при запиті: {e}")
return None
```

Рисунок 3.36 – Блок “except”

Далі йде функція check_logs що слугує для перевірки статусу системи через логи, які періодично оновлюються і відслідковуються. Цей процес дозволяє ідентифікувати і реагувати на потенційні проблеми або зміни стану системи.

Функція “check_logs”:

```
        def check_logs():
    error_detected = False

    # Оновлення логів до тих пір, поки printing не стане False
    while error_detected == False:

        global log_thread_finished
        tts_client = init_tts_client()
        logs = get_logs(log_url)

        if logs is not None:
            # Перевірка останнього запису в логах
            latest_log = logs[-1]
            printing_status = latest_log.get("printing")
            error_status = latest_log.get("error")
            current_tape_length_status = latest_log.get("length")

           # print(f"Latest log - Printing status: {printing_status},"
            #      f" Error status: {error_status},"
              #   f" length: {current_tape_length_status}")

            if current_tape_length_status == "50":
                # Якщо залишилось 10 відсотків
                text_to_speech(tts_client, f"Залишилось 50%:
{current_tape_length_status}")
                print("Залишилось 50%")

            # Переривання циклу, якщо error має помилку
#
            if logs and any(log.get("printing") is False for log in logs):
```

```
            # Зупинка слухання та виведення питання


            response = input("Сталася помилка, чи хочете ви продовжити? (Y/N):
")
            if response.lower() == 'y':

                if error_status == "Tape out":
                    # Якщо скінчилась стрічка
                    text_to_speech(tts_client, f"Скінчилась стрічка:
{current_tape_length_status}")




                if error_status == "Error":
                    # Якщо є помилка, відправити повідомлення через TTS
                    text_to_speech(tts_client, f"Помилка принтера: {error_status}")

                error_detected = True
                return True
            else:
                log_thread_finished.set()
                return False
```

Детально розглянемо кожен елемент цієї функції, рисунок 3.37:

```
    def check_logs():
error_detected = False
```

Рисунок 3.37 – Визначення функції “check_logs”

– визначення функції check_logs. Встановлення локальної змінної error_detected у значення False ініціює логіку перевірки без початкової наявності помилок;

– цикл while продовжується, доки помилка не буде виявлена. Це основна структура циклу, що дозволяє функції постійно перевіряти логи до виявлення помилки, рисунок 3.38;

```
    while error_detected == False:
```

Рисунок 3.38 – Цикл “while”

– встановлення глобальної змінної log_thread_finished, що може використовуватись для контролю стану потоку. Ініціалізація клієнта для Text-to-Speech і отримання логів з вказаної URL-адреси. Ці кроки забезпечують необхідні ресурси та дані для подальшої обробки, рисунок 3.39;

```
    global log_thread_finished
tts_client = init_tts_client()
logs = get_logs(log_url)
```

Рисунок 3.39 – Встановлення глобальної змінної “log_thread_finished”

– якщо логи успішно отримані (не None), функція аналізує останній запис у логах. З цього запису витягується інформація про статус друку, наявність помилок та стан стрічки (якщо відомо), рисунок 3.40;

```
    if logs is not None:
# Перевірка останнього запису в логах
latest_log = logs[-1]
printing_status = latest_log.get("printing")
error_status = latest_log.get("error")
current_tape_length_status = latest_log.get("length")
```

Рисунок 3.40 – Аналіз останнього запису у логах

– якщо стан стрічки показує, що залишилось 50%, система за допомогою TTS інформує користувача про це стан. Це використовується як попередження для можливого втручання або дій, рисунок 3.41.

```
    if current_tape_length_status == "50":
# Якщо залишилось 10 відсотків
text_to_speech(tts_client, f"Залишилось 50%: {current_tape_length_status}")
print("Залишилось 50%")
```

Рисунок 3.41 – Інформування користувача

Функція перевірки наявність помилок:

```
    if logs and any(log.get("printing") is False for log in logs):
  response = input("Сталася помилка, чи хочете ви продовжити? (Y/N): ")
  if response.lower() == 'y':
      if error_status == "Tape out":
          text_to_speech(tts_client, f"Скінчилась стрічка:
{current_tape_length_status}")
      if error_status == "Error":
          text_to_speech(tts_client, f"Помилка принтера: {error_status}")
      error_detected = True
      return True
```

```
    else:
        log_thread_finished.set()
        return False
```

– функція перевіряє наявність помилок, зокрема, перевіряючи статус друку в кожному запису логів. Якщо помилка виявлена, користувачеві надається можливість вибору продовження роботи. Відповідь користувача обробляється, і, в залежності від вибору, можуть бути вжиті різні заходи (наприклад, інформування про специфічні помилки через TTS). Якщо користувач вирішує не продовжувати, потік завершується.

3.4 Модуль виконання помилок

Цей модуль є частиною системи, що використовує модуль Text-to-Speech для комунікації з користувачем у разі виявлення помилок у логах системи.

```
        from tts import text_to_speech, init_tts_client
import requests


log_url = "http://127.0.0.1:5000/logs"

def get_logs(url):
    try:
        response = requests.get(url)
        response.raise_for_status()  # Перевірка на наявність помилок HTTP
        return response.json()  # Повернення логів у форматі JSON
    except requests.RequestException as e:
        print(f"Помилка при запиті: {e}")
```

```
        return None

def execute(category):
    global log_thread_finished
    tts_client = init_tts_client()

    logs = get_logs(log_url)

    # Перевірка останнього запису в логах
    latest_log = logs[-1]
    error_status = latest_log.get("error")
    current_tape_length_status = latest_log.get("length")

    if category == "Помилка":

        if current_tape_length_status == "0":
            # Якщо скінчилась стрічка
            text_to_speech(tts_client, f"Треба замінити стрічку:
{current_tape_length_status}")

        if error_status == "Error":
            # Якщо є помилка, відправити повідомлення через TTS
            text_to_speech(tts_client, f"Помилка принтера. {error_status} Ось три
найчастіші проблеми:  "
                                    f"1) Проблеми з підключенням та мережею: Перевірте
мережеві кабелі та обладнання, перезавантажте мережеві пристрої,
перенастроюйте принтер у мережі."
                                    f"2) Засмічення або знос обладнання: Проведіть чистку
принтера, зокрема друкуючої головки, замініть зношені частини,
використовуйте рекомендовані витратні матеріали."
```

f"3) Збій у програмному забезпеченні: Перезавантажте принтер, встановіть доступні оновлення ПЗ, за потреби перевстановіть ПЗ або зверніться до техпідтримки.")

Основна функція execute відповідає за обробку категорій помилок та надає зворотний зв'язок користувачу з використанням голосових команд. Розглянемо кожен частину коду більш детально.

from tts import text_to_speech, init_tts_client: імпортує функції text_to_speech та init_tts_client з модуля tts. Це основні інструменти для ініціалізації клієнта Text-to-Speech та виконання синтезу мовлення, рисунок 3.42.

```
    from tts import text_to_speech, init_tts_client
import requests



log_url = "http://127.0.0.1:5000/logs"
```

Рисунок 3.42 – Імпорти та Глобальні Змінні

import requests: імпорт бібліотеки requests, яка використовується для здійснення HTTP запитів. У цьому контексті вона слугує для отримання логів з сервера.

log_url: змінна, яка зберігає URL веб-сервера, з якого функція get_logs буде збирати логи. Вказує на локальний сервер для розробки або тестування.

Огляд функцій;

– функція execute, рисунок 3.43;

```
    def execute(category):
  global log_thread_finished
```

```
    tts_client = init_tts_client()
    logs = get_logs(log_url)
    latest_log = logs[-1]
    error_status = latest_log.get("error")
    current_tape_length_status = latest_log.get("length")
    if category == "Помилка":
        if current_tape_length_status == "0":
            text_to_speech(tts_client, f"Треба замінити стрічку:
{current_tape_length_status}")
        if error_status == "Error":
            text_to_speech(tts_client, f"Помилка принтера. {error_status} Ось три
найчастіші проблеми:  "
                                f"1) Проблеми з підключенням та мережею: Перевірте
мережеві кабелі та обладнання, перезавантажте мережеві пристрої,
перенастроюйте принтер у мережі."
                                f"2) Засмічення або знос обладнання: Проведіть чистку
принтера, зокрема друкуючої головки, замініть зношені частини,
використовуйте рекомендовані витратні матеріали."
```

Рисунок 3.43 – Функція “execute”

– функція execute призначена для реагування на специфічні категорії помилок, ідентифіковані в системних логах, з використанням Text-to-Speech (TTS) технології для сповіщення користувачів. Ця функція інтегрує декілька модулів для обробки логів, ініціалізації TTS клієнта, та генерації голосових повідомлень на основі статусу помилок, рисунок 3.44;

```
        def execute(category):
    global log_thread_finished
    tts_client = init_tts_client()
```

Рисунок 3.44 – Оголошення функції

– def execute(category): – оголошення функції з одним параметром category, який визначає тип помилки або події, на яку потрібно реагувати;

– global log_thread_finished – вказівка на використання глобальної змінної log_thread_finished, яка контролює та сигналізує про завершення поточного потоку обробки логів;

– logs = get_logs(log_url) – виклик функції get_logs з URL, що містить місце зберігання логів, для отримання актуальних даних логів у форматі JSON, рисунок 3.45;

```
        logs = get_logs(log_url)

# Перевірка останнього запису в логах
latest_log = logs[-1]
error_status = latest_log.get("error")
current_tape_length_status = latest_log.get("length")
```

Рисунок 3.45 – Виклик функції “get_logs”

– latest_log = logs[-1] – отримання останнього запису з логів. Припускається, що логи впорядковані за часом, та останній запис містить найсвіжішу інформацію;

– error_status = latest_log.get("error") – витягування інформації про наявність помилки з останнього запису;

– text_to_speech(tts_client, f"Помилка принтера. {error_status} Ось три найчастіші проблеми:... – виведення голосового повідомлення з описом потенційних причин помилки та рекомендаціями щодо їх вирішення, включаючи перевірку мережі, чистку обладнання, оновлення програмного забезпечення, рисунок 3.46.

```
        if category == "Помилка":

    if current_tape_length_status == "0":
        # Якщо скінчилась стрічка
        text_to_speech(tts_client, f"Треба замінити стрічку:
{current_tape_length_status}")
    if error_status == "Error":
        text_to_speech(tts_client, f"Помилка принтера. {error_status} Ось три
найчастіші проблеми:  "
                               f"1) Проблеми з підключенням та мережею: Перевірте
мережеві кабелі та обладнання, перезавантажте мережеві пристрої,
перенастроюйте принтер у мережі."
                               f"2) Засмічення або знос обладнання: Проведіть чистку
принтера, зокрема друкуючої головки, замініть зношені частини,
використовуйте рекомендовані витратні матеріали."
                               f"3) Збій у програмному забезпеченні: Перезавантажте
принтер, встановіть доступні оновлення ПЗ, за потреби перевстановіть ПЗ
або зверніться до техпідтримки.")
```

Рисунок 3.46 – Виведення голосового повідомлення

3.5 Модуль симуляції роботи принтера

Цей код є прикладом серверного застосунку, написаного на Flask, який контролює віртуальний принтер. Сервер відслідковує стан принтера, веде логи подій, та забезпечує веб-інтерфейс для керування друкуванням.

```
        from flask import Flask, jsonify, request, render_template
from flask_cors import CORS
import time
import threading
#

#   curl http://127.0.0.1:5000/start_printing

#   curl http://127.0.0.1:5000/status

#   curl http://127.0.0.1:5000/stop_printing

# curl http://127.0.0.1:5000/logs



app = Flask(__name__)
CORS(app)
# Список для логів
logs = []

# Початкові параметри принтера
printer_state = {
    "total_tape_length": 200, # загальна довжина стрічки в метрах
    "length": 200, # Актуальна довжина стрічки
    "printing_speed": 40, # Швидкість друку у метрах про хвилину
    "printing": False, # Стан принтера
    "error": None # Помилка, якщо є
}
```

```
print_timer = None

last_sent_update = 0  # Змінна для відстеження останнього відправленого оновлення

def print_job():
    global print_timer, last_sent_update

    while printer_state["printing"]:
        printer_state["length"] -= printer_state["printing_speed"]
        current_usage_percentage = (1 - printer_state["length"] / printer_state["total_tape_length"]) * 100

        # Перевірка чи використано більше 24 метрів стрічки  (настройка значень при яких зупиняється принтер)
        if printer_state["total_tape_length"] - printer_state["length"] > 200:
            printer_state["printing"] = False
            printer_state["error"] = "Error"

        # Якщо закінчилась стрічка то ми отримуемо помилку
        if printer_state["length"] <= 0:
            printer_state["length"] = 0
            printer_state["printing"] = False
            printer_state["error"] = "Tape out"
            break

        # Інформація о стрічці кожні 5 секунд
        if print_timer is None:
```

```
            print_timer = threading.Timer(5, print_tape_info)
            print_timer.start()

        time.sleep(5)

# Функция для додавання логів
def add_log():
    while True:
        log_entry = {
            "time": time.ctime(),
            "length": printer_state["length"],
            "printing": printer_state["printing"],
            "error": printer_state["error"]
        }
        logs.clear()  # Очистка попередніх логів
        logs.append(log_entry)
        time.sleep(5)

# Запуск потока для регулярного додавання логів
log_thread = threading.Thread(target=add_log)
log_thread.start()
def print_tape_info():
    global print_timer
    used_length = printer_state["total_tape_length"] - printer_state["length"]
    log_entry = {
        "time": time.ctime(),
        "message": f"Використано {used_length} метрів стрічки",
        "length": printer_state["length"],
        "printing": printer_state["printing"]
    }
```

```
    logs.append(log_entry)
    print_timer = None


@app.route('/start_printing')
def start_printing():
    if not printer_state["printing"] and printer_state["length"] > 0:
        printer_state["printing"] = True
        printer_state["error"] = None
        threading.Thread(target=print_job).start()
        return jsonify({"message": "Printing started"})
    else:
        return jsonify({"error": "Cannot start printing"})

@app.route('/stop_printing')
def stop_printing():
    printer_state["printing"] = False
    return jsonify({"message": "Printing stopped"})

@app.route('/status')
def status():
    return jsonify({
        "length": printer_state["length"],
        "printing": printer_state["printing"],
        "error": printer_state["error"]
    })

@app.route('/logs')
def get_logs():
    # Повертаєм останні 20 записів лога
```

```
    return jsonify(logs)

@app.route('/')
def index():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

Детальний огляд модулю “printer”:

Імпорти та глобальні змінні, рисунок 3.47:

```
from flask import Flask, jsonify, request, render_template
```

Рисунок 3.47 – Імпорт класу “Flask”

– Flask: клас, що використовується для створення екземпляра веб-додатка. Flask є мікро-фреймворком для розробки веб-додатків, що надає засоби для обробки запитів, налаштування маршрутів та інші веб-функціональності;

– jsonify: функція, яка використовується для перетворення словарів Python в JSON-відповіді. Це дозволяє легко відправляти структуровані дані через HTTP у форматі, який легко інтегрувати та використовувати на клієнтській стороні;

– request: об'єкт, що використовується для отримання даних запиту від клієнта. Він може містити параметри, JSON тіла, заголовки та інші дані, що надсилаються з веб-клієнта;

– render_template: функція, що дозволяє згенерувати HTML відповідь на основі шаблонів Jinja2. Вона використовує HTML шаблони та змінні Python для динамічного генерування веб-сторінок.

CORS: модуль, що дозволяє налаштувати політику CORS (Cross-Origin Resource Sharing) для Flask додатка. CORS необхідний для безпечного та контрольованого доступу до ресурсів веб-додатка з доменів, відмінних від домену, на якому знаходиться сам додаток. Це дуже важливо для API, які використовуються сторонніми веб-додатками, рисунок 3.48.

```
from flask_cors import CORS
```

Рисунок 3.48 – Імпорт модулю “Cors”

threading: модуль, що використовується для багатопоточності в Python. Він дозволяє додатку виконувати кілька задач одночасно, що є корисним для операцій, які можуть бути довготривалими або які потребують асинхронного виконання. Наприклад, можна запустити фоновий потік, який обробляє дані або виконує інтенсивні до ресурсів завдання без блокування основного потоку веб-сервера, рисунок 3.49.

```
import threading
```

Рисунок 3.49 – Імпорт модулю “Threading”

Це виклик конструктора класу Flask, який створює новий екземпляр веб-додатку. Аргумент __name__ вказує, що інстанція додатку буде використовувати поточний модуль як точку початку для визначення шляхів, статичних файлів, шаблонів та інших ресурсів, рисунок 3.50.

```
app = Flask(__name__)
```

Рисунок 3.50 – Виклик конструктора класу Flask

Це виклик функції CORS з модуля flask_cors, який призначений для додавання заголовків CORS до всіх відповідей, що генеруються додатком Flask. CORS (Cross-Origin Resource Sharing) дозволяє або обмежує запити до веб-ресурсів на домені з сторінок від інших доменів, що забезпечує додатковий рівень безпеки, рисунок 3.51.

```
CORS(app)
```

Рисунок 3.51 – Виклик функції CORS

Цей рядок створює пустий список logs, який використовується для зберігання записів логів, рисунок 3.52.

```
logs = []
```

Рисунок 3.52 – Створення пустого списку

– total_tape_length: 200 – позначає загальну довжину стрічки принтера в метрах;

– length: 200 – це поточна довжина невикористаної стрічки в метрах;

– printing_speed: 40 – швидкість друку, яка вказує, скільки метрів стрічки принтер витрачає за хвилину;

– printing: False – булевий індикатор стану принтера, де False означає, що принтер не друкує;

– error: None – цей параметр використовується для збереження інформації про будь-які помилки, які можуть виникнути під час роботи принтера, рисунок 3.53;

```
    printer_state = {
  "total_tape_length": 200,
  "length": 200,
  "printing_speed": 40,
  "printing": False,
  "error": None
}
```

Рисунок 3.53 – Словник

Функції.

Функція print_job виконує контроль над процесом друку в віртуальному принтері, зменшуючи кількість доступної стрічки на основі заданої швидкості друку, моніторить використання стрічки і зупиняє друк у випадках перевищення ліміту або закінчення стрічки. Розглянемо детально кожну частину і її функціональність у загальній логіки функції, рисунок 3.54;

```
    def print_job():
  global print_timer, last_sent_update
```

Рисунок 3.54 – Визначення функції “print_job”

Визначення функції print_job, яка не приймає параметрів ззовні. Оголошує print_timer та last_sent_update як глобальні змінні, щоб зміни в цих змінних впливали на глобальний стан і були видимі в інших частинах програми.

Цикл виконується, поки стан принтера printing є True, тобто друк активний. Це основний цикл, що контролює процес друку, рисунок 3.54.

```
    while printer_state["printing"]:
```

Рисунок 3.55 – Виконання циклу

З кожним проходом циклу зменшується кількість стрічки, доступної для друку. Кількість, на яку зменшується стрічка, визначається швидкістю друку (printing_speed), що вказує, скільки метрів стрічки витрачається за одну хвилину, рисунок 3.56.

```
        printer_state["length"] -= printer_state["printing_speed"]
```

Рисунок 3.56 – Зменшення кількості стрічки

Розрахунок відсотка використаної стрічки від загальної довжини стрічки. Це дозволяє моніторити ступінь використання ресурсу, рисунок 3.57.

```
        current_usage_percentage = (1 - printer_state["length"] /
printer_state["total_tape_length"]) * 100
```

Рисунок 3.57 – Розрахунок відсотка використаної стрічки

Якщо використано більше ніж 200 метрів стрічки (в цьому випадку це помилка у вихідному коді, оскільки стрічка не може бути довшою за її загальну довжину), принтер зупиняється, і реєструється помилка "Error", риснок 3.58.

```
        if printer_state["total_tape_length"] - printer_state["length"] > 200:
    printer_state["printing"] = False
    printer_state["error"] = "Error"
```

Рисунок 3.58 – Умова для керування

Якщо стрічка закінчилася (довжина стрічки <= 0), функція зупиняє друк, встановлює довжину стрічки на 0, вказує помилку "Tape out", і виходить з циклу, рисунок 3.59.

```
        if printer_state["length"] <= 0:
    printer_state["length"] = 0
    printer_state["printing"] = False
    printer_state["error"] = "Tape out"
    break
```

Рисунок 3.59 – Умова для керування

Якщо print_timer не активовано, ініціалізується новий таймер, що викликає функцію print_tape_info кожні 5 секунд. Це дозволяє регулярно оновлювати інформацію про стан стрічки, рисунок 3.60.

```
        if print_timer is None:
    print_timer = threading.Timer(5, print_tape_info)
    print_timer.start()
```

Рисунок 3.60 – Умова ініціалізації

Встановлення затримки в 5 секунд між ітераціями циклу, що зменшує навантаження на процесор та імітує реальний часовий інтервал між проходами стрічки через принтер, рисунок 3.61.

```
        time.sleep(5)
```

Рисунок 3.61 – Встановлення затримки

Наступною функцією є add_log і вона призначена для безперервного додавання логів стану принтера в список logs. Ця функція є частиною системи моніторингу стану принтера в рамках веб-додатку, створеного за допомогою Flask.

Детальний аналіз цієї функції, рисунок 3.62:

```
    def add_log():
```

Рисунок 3.62 – Визначення функції “add_log”

Визначення функції add_log, яка не приймає жодних аргументів. Ця функція призначена для виконання в окремому потоці і відповідає за регулярне додавання записів логів до загального списку.

Використання нескінченного циклу while True забезпечує постійне виконання логіки внутрішньої функції, поки програма активна або доки цикл не буде перерваний зовнішніми умовами або командами, рисунок 3.63.

```
    while True:
```

Рисунок 3.63 – Використання нескінченного циклу “while”

Створення словника log_entry, який зберігає актуальний стан принтера, рисунок 3.64:

```
    log_entry = {
  "time": time.ctime(),
  "length": printer_state["length"],
  "printing": printer_state["printing"],
  "error": printer_state["error"]
}
```

Рисунок 3.64 – Створення словника “log_entry”

- "time": time.ctime(): запис часу створення логу в читабельній формі, що використовується для відстеження часу виникнення події;
- "length": printer_state["length"]: запис поточної довжини стрічки в принтері;

– "printing": printer_state["printing"]: запис поточного стану принтера, чи активний він (друкує);

– "error": printer_state["error"]: запис поточного стану помилки, якщо така є;

– logs.clear(): очищення списку logs від усіх попередніх записів перед додаванням нового. Це забезпечує, що в списку завжди зберігається тільки останній стан принтера;

– logs.append(log_entry): додавання новоствореного запису log_entry до списку logs, зберігаючи поточну інформацію про стан принтера, рисунок 3.65.

```
        logs.clear()  # Очистка попередніх логів
logs.append(log_entry)
```

Рисунок 3.65 – Очищення і додавання списку “logs”

Встановлення затримки у 5 секунд перед наступною ітерацією циклу. Це регулює частоту оновлення логів і зменшує навантаження на систему, рисунок 3.66.

```
        time.sleep(5)
```

Рисунок 3.66 – Встановлення затримки

Функція print_tape_info призначена для оновлення інформації про використання стрічки в принтері і запису цієї інформації в логи. Ця функція також відіграє роль в ресеті глобального таймера, який використовується для періодичного виклику самої функції. Розглянемо докладно кожну частину функції:

Визначення функції print_tape_info, яка не приймає зовнішніх параметрів. Ця функція призначена для виклику з таймера, що запускається в контексті глобальної функції обробки друку, рисунок 3.67.

```
    def print_tape_info():
```

Рисунок 3.67 – Визначення функції print_tape_info

Ключове слово global використовується для модифікації змінної print_timer, яка оголошена на рівні модуля. Це дозволяє функції встановлювати print_timer в None, що вказує на необхідність реініціалізації таймера при наступному виклику, рисунок 3.68.

```
    global print_timer
```

Рисунок 3.68 – Модифікація змінної

Розрахунок кількості стрічки, що вже використана, шляхом віднімання поточної довжини стрічки (printer_state["length"]) від загальної довжини (printer_state["total_tape_length"]), рисунок 3.69.

```
    used_length = printer_state["total_tape_length"] - printer_state["length"]
```

Рисунок 3.69 – Розрахунок кількості стрічки

Створення словника log_entry, який містить таймстемп (time), повідомлення з кількістю використаної стрічки, поточну довжину стрічки та статус друку, рисунок 3.70.

```
    log_entry = {
  "time": time.ctime(),
  "message": f"Використано {used_length} метрів стрічки",
  "length": printer_state["length"],
  "printing": printer_state["printing"]
}
```

Рисунок 3.70 – Створення словника “log_entry”

Додавання новоствореного запису логу log_entry до глобального списку logs, рисунок 3.71.

```
    logs.append(log_entry)
```

Рисунок 3.71 – Додавання новоствореного запису логу

Встановлення print_timer в None, що показує, що таймер потребує повторної ініціалізації при наступному виклику функції print_tape_info, рисунок 3.72.

```
    print_timer = None
```

Рисунок 3.72 – Встановлення “print_timer” в “None”

Роутинги.

Роутинги використовуються для керування станом принтера та перегляду його логів. Кожен маршрут використовує декоратор @app.route() для визначення кінцевих точок API, що реагують на HTTP запити. Давайте детально розглянемо кожну частину цього коду.

Маршрут /start_printing, рисунок 3.73.

```
    @app.route('/start_printing')
def start_printing():
```

Рисунок 3.73 – визначання маршруту

Декоратор @app.route() визначає маршрут для запуску друку. Функція start_printing() викликається, коли користувач відправляє HTTP запит до цього маршруту.

Умова перевіряє, чи принтер не друкує в даний момент (printer_state["printing"] є False) та чи залишилась достатня кількість стрічки (printer_state["length"] більше нуля), рисунок 3.74.

```
    if not printer_state["printing"] and printer_state["length"] > 0:
```

Рисунок 3.74 – умова перевірки

Якщо умови вище задовільнені, стан принтера змінюється на активний друк, і будь-які помилки ресетуються, рисунок 3.75.

```
    printer_state["printing"] = True
printer_state["error"] = None
```

Рисунок 3.75 – зміна стану принтера

Запускається новий потік, який викликає функцію print_job(). Ця функція керує процесом друку, 3.76.

```
    threading.Thread(target=print_job).start()
```

Рисунок 3.76 – запуск нового потоку

Функція повертає JSON-відповідь із повідомленням про початок друку, рисунок 3.77.

```
    return jsonify({"message": "Printing started"})
```

Рисунок 3.77 – функція повертання JSON-відповіді

Якщо умови для початку друку не виконані, повертається повідомлення про помилку, рисунок 3.78.

```
        else:
    return jsonify({"error": "Cannot start printing"})
```

Рисунок 3.78 – повернення повідомлення про помилку

Маршрут для зупинки друку. Функція stop_printing() викликається, коли користувач відправляє відповідний запит, рисунок 3.79.

```
        @app.route('/stop_printing')
def stop_printing():
```

Рисунок 3.79 – маршрут для зупинки друку

Стан принтера змінюється на неактивний, рисунок 3.80.

```
        printer_state["printing"] = False
```

Рисунок 3.80 – зміна стану принтера

Повертається JSON-відповідь, що повідомляє про зупинку друку, рисунок 3.81.

```
        return jsonify({"message": "Printing stopped"})
```

Рисунок 3.81 – повертання JSON-відповіді

Маршрут /status, рисунок 3.82.

```
        @app.route('/status')
def status():
```

Рисунок 3.82 – маршрут для отримання стану

Маршрут для отримання поточного стану принтера.

Повертає JSON-відповідь з даними про довжину стрічки, стан друку та наявність помилок, рисунок 3.83.

```
        return jsonify({
    "length": printer_state["length"],
    "printing": printer_state["printing"],
    "error": printer_state["error"]
})
```

Рисунок 3.83 – повертання JSON-відповіді

Маршрут /logs, рисунок 3.84.

```
        @app.route('/logs')
def get_logs():
```

Рисунок 3.84 – маршрут для отримання логів

Маршрут для отримання логів принтера.

Повертає JSON-відповідь, що містить усі записи логів. Це забезпечує доступ до історичних даних про роботу принтера, рисунок 3.85.

```
        return jsonify(logs)
```

Рисунок 3.85 – повертання JSON-відповіді

Головна сторінка, рисунок 3.86.

```
@app.route('/')
def index():
```

Рисунок 3.86 – маршрут для головної сторінки

Маршрут для головної сторінки додатка.

Відправляє HTML-сторінку, згенеровану на основі шаблону index.html. Це може включати користувальницький інтерфейс для взаємодії з принтером, рисунок 3.87.

```
    return render_template('index.html')
```

Рисунок 3.87 – відправлення HTML-сторінки

3.6 Модуль розуміння мови

Модуль input_voice використовується для розуміння мови з використанням бібліотеки Vosk, яка є фреймворком для розпізнавання мови. Цей модуль конфігурує аудіо запис через мікрофон і використовує модель Vosk для інтерпретації звуків у текст.

Загальний код модулю:

```
    import sounddevice as sd
import queue
from vosk import Model, KaldiRecognizer
import sys
import time

model = Model("small_model")
samplerate = 16000
device = 1

def va_listen(callback, listen_duration = 3):
```

```
    q = queue.Queue()

    def q_callback(indata, frames, time, status):

        if status:
            print(status, file=sys.stderr)
        q.put(bytes(indata))
#
    with sd.RawInputStream(samplerate=samplerate, blocksize=8000,
device=device, dtype='int16', channels=1, callback=q_callback):
        rec = KaldiRecognizer(model, samplerate)
        print("Запис почався")
        start_time = time.time()  # Записуємо час початку прослуховування
       # global keep_listening
       # keep_listening = True

        while True:
            current_time = time.time()
            if current_time - start_time > listen_duration:
                data = q.get()
                if rec.AcceptWaveform(data):
                    callback(rec.Result())
                    print(rec.Result())
```

Давайте детально розглянемо кожну частину цього коду:

імпорти модулів та бібліотек, рисунок 3.88:

```
import sounddevice as sd
```

Рисунок 3.88 – імпорт модулю “sounddevice”

– модуль sounddevice надає інтерфейс для запису та відтворення звуку на різних платформах за допомогою Python. Цей модуль є обгорткою над бібліотекою PortAudio, яка є крос-платформеною бібліотекою для роботи з аудіо. Використання sounddevice у проекті дозволяє здійснювати захоплення звуку в реальному часі, що є критично важливим для систем, які виконують аналіз аудіо або розпізнавання мови;

– модуль queue в Python надає імплементацію багатопоточної черги, яка використовується для безпечного обміну даними між потоками. Черга в цьому контексті забезпечує FIFO (first-in, first-out) організацію елементів. Черги використовуються для того, щоб уникнути конфліктів даних та забезпечити синхронізацію при обробці аудіо сигналів у реальному часі, рисунок 3.89;

```
import queue
```

Рисунок 3.89 – імпорт модулю “sounddevice”

– Vosk – це бібліотека для розпізнавання мови, яка базується на Kaldi, потужному інструменті для автоматичного розпізнавання мови. Model використовується для завантаження переднавчених моделей мови, а KaldiRecognizer — для розпізнавання мови з аудіо потоку. Застосування Vosk в проекті надає можливість розпізнавати мову в реальному часі, що є ключовою вимогою для багатьох застосунків, пов'язаних з AI та ML, рисунок 3.90;

```
from vosk import Model, KaldiRecognizer
```

Рисунок 3.90 – імпорт бібліотеки “Vosk”

– цей модуль використовується для взаємодії з інтерпретатором Python, зокрема, для роботи з системними змінними та управління потоками виводу, рисунок 3.91.

```
import sys
```

Рисунок 3.91 – імпорт модулю “sounddevice”

Model("small_model"): Завантаження моделі розпізнавання мови Vosk. "small_model" є назвою моделі, яка має бути розміщена в папці проекту, рисунок 3.92.

```
model = Model("small_model")
samplerate = 16000
device = 1
```

Рисунок 3.92 – завантаження моделі розпізнавання мови

samplerate = 16000: Частота дискретизації в Гц, яка використовується для запису звуку.

device = 1: Індекс аудіо пристрою, який використовується для запису.

va_listen(callback, listen_duration = 3): Функція, яка ініціює запис мови з мікрофону і передає результати у вигляді тексту через функцію зворотного виклику (callback), рисунок 3.93.

```
def va_listen(callback, listen_duration = 3):
  q = queue.Queue()
```

Рисунок 3.93 – функція ініціювання запису

queue.Queue(): Створення черги для безпечної передачі аудіо даних з потоку запису до потоку обробки;

– q_callback – це функція, що використовується як зворотній виклик при захопленні звуку через sounddevice. Вона призначена для обробки звукових даних, які надходять від мікрофона, рисунок 3.94;

```
def q_callback(indata, frames, time, status):
```

Рисунок 3.94 – функція зворотнього виклику при захопленні звуку через sounddevice

– indata: масив з даними звуку, які були захоплені. Це "сирі" дані в форматі, який вибрано при налаштуванні потоку в sounddevice;

– frames: кількість кадрів у блоку даних, який був захоплений;

– time: часові мітки, які можуть бути використані для визначення моменту захоплення кожного кадру;

– status: статус поточного захоплення, може містити помилки або інші важливі повідомлення стосовно захоплення аудіо;

– q.put(bytes(indata)): додавання аудіо даних у чергу для подальшого аналізу.

Ця умова перевіряє, чи існують які-небудь повідомлення про статус або помилки, що виникли під час захоплення звуку. Якщо такі існують, вони виводяться в стандартний потік помилок, рисунок 3.95.

```
    if status:
print(status, file=sys.stderr)
```

Рисунок 3.95 – умова перевірки

file=sys.stderr: напрямок виводу помилок в стандартний потік помилок дозволяє легко відсортувати ці повідомлення від звичайного виводу програми.

- sd.RawInputStream(...): створення потоку для запису аудіо з мікрофону з заданою частотою дискретизації, розміром блоку, пристроєм,

типом даних, кількістю каналів та функцією зворотного виклику, рисунок 3.96;

```
        with sd.RawInputStream(samplerate=samplerate, blocksize=8000,
device=device, dtype='int16', channels=1, callback=q_callback):
    rec = KaldiRecognizer(model, samplerate)
    print("Запис почався")
    start_time = time.time()
```

Рисунок 3.96 – створення потоку для запису аудіо

- samplerate=samplerate: визначає частоту дискретизації аудіо, в більшості випадків для мови це 16000 Гц;
- blocksize=8000: розмір блоку даних, який буде оброблятися за один раз. Це значення може впливати на затримку та продуктивність обробки аудіо;
- device=device: визначає аудіо пристрій, який буде використовуватися для захоплення звуку. Це може бути індексом конкретного мікрофону або аудіо інтерфейсу;
- dtype='int16': формат даних, який використовується для зберігання аудіосигналів. int16 є стандартним вибором, оскільки він забезпечує хороший баланс між розміром даних і якістю звуку;
- channels=1: використання одного аудіоканалу (моно) замість стерео, що є достатнім для більшості завдань розпізнавання мови;
- callback=q_callback: функція зворотного виклику, яка буде викликатися sounddevice кожного разу, коли новий блок даних готовий до обробки. Ця функція відповідає за перенаправлення аудіоданих до черги для подальшої обробки;
- KaldiRecognizer(model, samplerate): ініціалізація розпізнавача Vosk для обробки аудіо потоку;

- model: завантажена модель мови, яка визначає, яким чином вхідні аудіодані будуть аналізуватися та інтерпретуватися;
- samplerate: частота дискретизації, з якою модель найефективніше працює, має відповідати вхідній частоті дискретизації в аудіопотоці;
- print("Запис почався"), start_time = time.time(): виведення повідомлення про початок запису і фіксація часу старту;
- Цикл перевіряє час запису і припиняє запис, коли досягнута вказана тривалість (listen_duration), рисунок 3.97;

```
    while True:
current_time = time.time()
if current_time - start_time > listen_duration:
    data = q.get()
    if rec.AcceptWaveform(data):
      callback(rec.Result())
      print(rec.Result())
```

Рисунок 3.97 – цикл перевірки часу запису

- rec.AcceptWaveform(data): Відправлення даних у розпізнавач і перевірка, чи було розпізнано аудіо як повне речення;
- callback(rec.Result()), print(rec.Result()): Виклик функції зворотного виклику з результатом розпізнавання і виведення результату в консоль.

3.7 Модуль нечіткого порівняння текстів

Цей Модуль створено для реалізації системи розпізнавання команд на основі вхідного тексту з використанням бібліотеки fuzzywuzzy, яка дозволяє визначати схожість між рядками. Система аналізує введену команду і вибирає

найбільш відповідну категорію на основі схожості введеного тексту до попередньо визначених варіантів команд у словнику.

```
        from fuzzywuzzy import process

# Словник можливих запитань

komtegory = {

   "Помилка": [
      "Що таке", "Що сталося", "У чому проблема", "Чому стоїмо",
"Розповідай", "Що будемо робити", "Що знову"
                ]
}
def process_command(command_text):

   best_match = None
   highest_score = 0

   for category, command_variants in komtegory.items():
      match, score = process.extractOne(command_text, command_variants)
      if score > highest_score:
         highest_score = score
         best_match = category

   return best_match if highest_score >= 60 else None
```

Детальний аналіз кожної частину коду, рисунок 3.98.

```
from fuzzywuzzy import process
```

Рисунок 3.98 – імпорт модулю “process”

Імпортується модуль process з бібліотеки fuzzywuzzy. Ця бібліотека використовується для порівняння текстів і визначення ступеню їх схожості, що використовує алгоритми, засновані на розрахунках Левенштейна.

Словник komtegory містить категорії і відповідні їм фрази-варіанти, які можуть бути використані для класифікації вхідних команд. Ключ "Помилка" асоційований зі списком фраз, що можуть вказувати на запитання про помилки або проблеми, рисунок 3.99.

```
komtegory = {

  "Помилка": [
    "Що таке", "Що сталося", "У чому проблема", "Чому стоїмо",
"Розповідай", "Що будемо робити", "Що знову"
            ]
}
```

Рисунок 3.99 – словник “komtegory”

Функція process_command призначена для обробки тексту команди, яку вводить користувач, і визначення відповідної категорії на основі визначених шаблонів, рисунок 3.100.

```
def process_command(command_text):
```

Рисунок 3.100 – функція “process_command”

Змінні best_match та highest_score використовуються для збереження найкращої відповідності та найвищого бала схожості, відповідно. Ці змінні оновлюються під час ітерації по словнику команд, рисунок 3.101.

```
    best_match = None
highest_score = 0
```

Рисунок 3.101 – змінні “best_match” та “highest_score”

Цикл перебирає кожну категорію і її варіанти команд в словнику. Функція extractOne з fuzzywuzzy.process використовується для знаходження найкращого збігу між введеною командою та варіантами команд у категорії, рисунок 3.102.

```
    for category, command_variants in komtegory.items():
  match, score = process.extractOne(command_text, command_variants)
```

Рисунок 3.102 – цикл перебору кожної категорії

Якщо знайдений бал схожості вищий за поточний найвищий бал, оновлюються змінні highest_score і best_match для відображення цього більш точного збігу, рисунок 3.103.

```
    if score > highest_score:
  highest_score = score
  best_match = category
```

Рисунок 3.103 – оновлення змінних “highest_score” і “best_match”

Функція повертає категорію найкращого збігу, якщо бал схожості перевищує поріг у 60. Якщо бал нижче порогу, повертається None, що свідчить про відсутність достатньої відповідності, рисунок 3.104.

```
return best_match if highest_score >= 60 else None
```

Рисунок 3.104 – функція повертання категорії найкращого збігу

3.8 Охорона праці

Розробка інтелектуальної голосової системи підтримки прийняття рішень є складним і багатофазовим процесом, який вимагає високого рівня безпеки та відповідності стандартам охорони праці. Особлива увага приділяється безпеці працівників при налаштуванні системи та зниження ризиків пов'язаних із фізичними факторами. Проведемо розрахунки для оцінки ризиків та необхідних заходів безпеки. При розробці ми повинні враховувати фізичні фактори, такі як шум, вібрації та електромагнітне випромінювання, також можуть впливати на здоров'я працівників. Необхідно мінімізувати вплив цих факторів шляхом застосування засобів захисту, таких як звукопоглинальні матеріали, та обмеження часу роботи з обладнанням, що генерує вібрації або випромінювання. Щоб розрахувати рівень шуму використовуємо формулу. Еквівалентний рівень шуму в децибелах дорівнює десяти помноженому на логарифм десятковий відношення інтенсивності звуку до порогової інтенсивності звуку

Електрична безпека є критичним аспектом при розробці інтелектуальних систем, оскільки будь-яке порушення може призвести до серйозних наслідків. Всі електричні пристрої повинні бути заземлені, а працівники повинні бути ознайомлені з правилами роботи з електрообладнанням. Регулярні перевірки стану електропроводки та пристроїв також є обов'язковими.

Основними заходами захисту від електричних небезпек є заземлення обладнання та використання автоматичних вимикачів. Розраховуємо таким чином, опір заземлення визначається як добуток питомого опору ґрунту і довжини заземлювача, поділений на площу перетину заземлювача.

Автоматичні вимикачі захищають електричні кола від перевантажень і коротких замикань. Розраховуємо за формулою, струм дорівнює відношенню потужності навантаження до добутку напруги та косинуса кута зсуву фаз.

Забезпечення охорони праці при розробці інтелектуальних голосових систем підтримки прийняття рішень вимагає комплексного підходу, який включає як організаційні, так і технічні заходи. Врахування фізичних факторів, а також захист від електричних небезпек є критичними для створення безпечного та ефективного робочого середовища. Впровадження цих заходів дозволить не лише зберегти здоров'я і безпеку працівників, а й підвищити продуктивність та якість виконуваних робіт.

Висновки

Мета дослідження полягала у покращенні продуктивності автоматизації на виробництві за рахунок розробки системи моніторингу. Завдання включали аналіз існуючих технологій, розробку програмного забезпечення та інтеграцію системи з виробничими процесами. Використовуючи методи машинного навчання та обробки мови, система була націлена на автоматизацію процесів розпізнавання команд і виведення потрібної інформації для операторів. Проект використовує передові технології, включаючи алгоритми машинного навчання для оптимізації процесів обробки мови і штучний інтелект для адаптації системи до специфіки заводських умов. Інтеграція з існуючими IT-інфраструктурами дозволяла забезпечити високу ефективність і швидкість обробки запитів.

Архітектура системи була спроектована таким чином, щоб забезпечити масштабованість та високу доступність. Компонентна структура дозволяла легко інтегрувати нові модулі та оновлення без зупинки системи. Розробка велася на Python з використанням фреймворків для обробки мови та машинного навчання, що забезпечило гнучкість та високу швидкість розробки. Використання відкритих бібліотек дозволило скоротити витрати та спростити підтримку програмного продукту. Система демонструє значні переваги у покращенні швидкості та точності процесів прийняття рішень на виробництві. Вона забезпечує операторам миттєвий доступ до необхідної інформації, що знижує час на обробку аварійних ситуацій. Система виявилася ефективнішою в порівнянні з традиційними методами управління, завдяки автоматизації рутинних завдань та здатності швидко адаптуватися до змін у виробничих процесах.

Розглянутий приклад фірми з виготовлення дерев’яних євро вікон демонструє практичну користь від впровадження інтелектуальної голосової системи. Машина “Working Process”, яка виконує сортування та фрезерування

деталей, часто стикається з різними проблемами, що призводить до автоматичних зупинок. Завдяки впровадженню інтелектуальної системи, завод може значно скоротити час, необхідний для вирішення зазначених проблем.

На основі наданих даних, система може знизити час на усунення проблем падіння дошки, невірної маси заготовки та зупинок витяжки. Наприклад, час на усунення проблеми падіння дошки може бути скорочений з 30 хвилин до 12 хвилин, що дає ефективність у 60%. Час на вирішення проблеми з масою заготовки може бути знижений з 80 хвилин до 36 хвилин, що забезпечує ефективність у 56%. Час на усунення проблеми з витяжкою може бути скорочений з 72 хвилин до 33 хвилин, що підвищує ефективність на 55%. Загальна потенційна економія часу становить 43 хвилини на день або 215 хвилин на тиждень, що є значним покращенням у 43% ефективності виробничих процесів. Це не лише зменшує витрати, пов'язані з простоєм обладнання, але й дозволяє операторам зосередитись на більш важливих завданнях, підвищуючи загальну продуктивність підприємства.

Застосування інтелектуальних систем підтримки прийняття рішень сприяє підвищенню конкурентоспроможності підприємства на ринку. Оптимізація виробничих процесів, зниження витрат, підвищення безпеки та продуктивності дозволяють підприємству забезпечувати високу якість продукції та швидко реагувати на зміни в ринкових умовах. Це є важливим фактором для успішного розвитку та зростання в умовах глобалізації.

Інтелектуальна система здатна аналізувати надходячі дані, виконувати пошук необхідної інформації та надавати користувачам зрозумілі та точні відповіді. Це значно знижує час на знаходження помилок та збільшує швидкість реакції на їх виправлення. Наприклад, у випадку з автоматизованою машиною “Working Process”, система може швидко ідентифікувати причину зупинки, надаючи оператору необхідну інформацію для швидкого усунення проблеми.

Таким чином, інтелектуальні системи підтримки прийняття рішень, зокрема голосові системи, є потужним інструментом для підвищення ефективності виробничих процесів. Вони дозволяють значно скоротити час на виконання рутинних операцій, забезпечують швидкий і точний аналіз даних, підвищують безпеку на виробництві та знижують витрати. Завдяки цим системам підприємства можуть залишатися конкурентоспроможними на ринку, адаптуватися до швидких змін і максимально використовувати свої можливості в умовах глобалізації. Успішна реалізація таких проектів сприятиме розвитку інноваційних технологій у промисловості та забезпеченню сталого розвитку підприємств.

ПЕРЕЛІК ПОСИЛАНЬ

1. Методичні вказівки з підготовки та захисту кваліфікаційної роботи здобувачами другого (магістерського) рівня вищої освіти спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології, освітньо-професійних програм: «Автоматизоване управління технологічними процесами», «Комп'ютерно-інтегровані технологічні процеси і виробництва», «Комп'ютеризовані та робототехнічні системи» / Упоряд. І. Ш. Невлюдов, Р. В. Артюх, В. В. Безкоровайний, Н. П. Демська, В. В. Євсєєв, О. І. Филипенко, О. М. Цимбал. – Харків: ХНУРЕ, 2021. – 55 с.
2. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки, структура та правила оформлення. – Введ. 2015-06-22. – К. Держстандарт України, 2017 – 29 с.
3. Невлюдов, І.Ш. Дипломне проектування для студентів усіх форм навчання спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології»: Навч. посібник / І.Ш. Невлюдов, А. О. Андрусевич, О. В. Токарєва, Г. В. Пономарьова – Київ-58, пр. Космонавта Комарова, 1, 2018 – 320с.
4. Науменко С. В., Михайловський П. В. ВИКОРИСТАННЯ VOSK MODEL ДЛЯ ПОКРАЩЕННЯ ЕФЕКТИВНОСТІ РОБОТИ ГОЛОСОВИХ АСИСТЕНТІВ //Інформаційні моделюючі технології, системи та комплекси (ІМТСК-2023): IV міжнародна науково-практична конференція. 25-26 травня 2023 р., Черкаси, Україна.–Черкаси: Черкаський національний університет імені Богдана Хмельницького, 2023.–153 с.
5. Худа А. О. Інтелектуальна система розпізнавання мовлення у вивченні іншомовних слів на основі машинного навчання. – 2020.
6. Величко С. В., Кайдан Н. В. Нечітка система обробки текстових даних. – 2022.
7. Костюченко А. О. Основи програмування мовою Python. – 2020.

8. Джулій В. М., Лєнков С. В., Муляр І. В. Метод наближеного пошуку та ідентифікації фізичних осіб. – 2018.