Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра __Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ____другий (магістерський)_____
_____Програмне забезпечення інтелектуального агента як помічника в_____
_____комп'ютерних інтерактивних моделях_____
(тема)

Виконав:
здобувач __2__ року навчання,
групи ___СКСм-23-2____

_____Шатан В.В._____
(прізвище, ініціали)

Спеціальність 123 – Комп'ютерна інженерія
_____
(код і повна назва спеціальності)

Тип програми ___освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)
Освітня програма _____
___Спеціалізовані комп'ютерні системи___
(повна назва освітньої програми)

Керівник ___проф. Кривуля Г.Ф._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ __Чумаченко С.В.__
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп'ютерна інженерія
(код і повна назва)

Тип програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Шатану В'ячеславу Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмне забезпечення інтелектуального агента як помічника в комп'ютерних інтерактивних моделях

затверджена наказом по університету від "08" листопада 2024 р. № 1189 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 24 січня 2025 р.

3. Вихідні дані до роботи Інтерактивні симуляції Phet, Інтелектуальні агенти, OpenAI, PyTorch, PyQt5, Flask, Моделі сегментації зображень, Системи генерації звітів, Selenium

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної області та постановка завдання
2. Аналіз поняття інтерактивності та автоматизації
3. Аналіз технологій для автоматизації та сегментації зображень
4. Розробка програмної реалізації проєкту
5. Тестування розробленої системи

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) 18 слайдів

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	Термін виконання етапів роботи	Примітка
1	Видача теми роботи її узгодження та затвердження	01.09.2024 - 15.09.2024	Виконано
2	Аналіз літератури за напрямом роботи	15.09.2024 - 29.09.2024	Виконано
3	Аналіз існуючих методів вирішення поставленого завдання	29.09.2024 - 14.10.2024	Виконано
4	Проектування системи	15.10.2024 - 26.10.2024	Виконано
5	Програмна реалізація	26.10.2024 – 28.11.2024	Виконано
6	Тестування розробленої системи	28.11.2024 - 06.12.2024	Виконано
7	Оформлення пояснювальної записки	06.12.2024 - 19.12.2024	Виконано
8	Оформлення графічного матеріалу	19.12.2024 - 25.12.2024	Виконано
9	Перевірка виконаного проєкту керівником	12.01.2025	Виконано
10	Захист кваліфікаційної роботи	24.01.2025	

Дата видачі завдання 01 вересня 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Кривуля Г. Ф.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи містить: 86 с., 3 табл., 14 рис., 13 джерел посилання.

ІНТЕЛЕКТУАЛЬНИЙ АГЕНТ, ІНТЕРАКТИВНІ СИМУЛЯЦІЇ, АВТОМАТИЗАЦІЯ ДІЙ, GPT, СЕГМЕНТАЦІЯ ЗОБРАЖЕНЬ, КОМП'ЮТЕРНИЙ ЗІР, ЗВІТИ, ПІДТРИМКА ЛОКАЛІЗАЦІЇ, FLASK, PYQT5, MACHINE LEARNING, PYTORCH, ПРОГРАМНА ІНТЕГРАЦІЯ, ВЕБ-БРАУЗЕР, ОБРОБКА СКРИНШОТІВ.

Метою кваліфікаційної роботи є розробка програмного забезпечення інтелектуального агента, який виконує роль помічника у взаємодії з комп'ютерними інтерактивними моделями, автоматизуючи дії користувача, аналізуючи зображення та створюючи звіти.

У ході кваліфікаційної роботи проведено аналіз предметної області, досліджено існуючі рішення для автоматизації дій у браузері, сегментації зображень та використання GPT для взаємодії з користувачем. Розроблено програмну реалізацію інтелектуального агента, яка включає модулі автоматизації дій, обробки скриншотів за допомогою алгоритмів сегментації, інтеграції GPT, а також створення звітів із можливістю підтримки локалізації. Особливу увагу приділено тестуванню системи, аналізу точності сегментації зображень, коректності автоматизованих дій та зручності використання графічного інтерфейсу.

ABSTRACT

The explanatory note contains 86 pp., 3 tables, 14 figures, 13 sources.

INTELLIGENT AGENT, INTERACTIVE SIMULATIONS, ACTION AUTOMATION, GPT, IMAGE SEGMENTATION, COMPUTER VISION, NATURAL LANGUAGE PROCESSING, REPORTS, LOCALIZATION SUPPORT, FLASK, PYQT5, MACHINE LEARNING, PYTORCH, SOFTWARE INTEGRATION, WEB BROWSER, SCREENSHOT PROCESSING.

The purpose of the qualification work is to develop software for an intelligent agent that acts as an assistant in interacting with computer interactive models by automating user actions, analyzing images, and generating reports.

In the process of completing the certification work, the domain was analyzed, existing solutions for automating actions in a web browser, image segmentation, and GPT integration for user interaction were studied. A software implementation of the intelligent agent was developed, including modules for action automation, screenshot processing using segmentation algorithms, GPT integration, and report generation with localization support. Special attention was paid to testing the system, analyzing the accuracy of image segmentation, correctness of automated actions, and user-friendliness of the graphical interface.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence – штучний інтелект.

CNN – Convolutional Neural Network – згорткова нейронна мережа.

OCR – Optical Character Recognition – оптичне розпізнавання символів.

API – Application Programming Interface – інтерфейс програмування додатків.

UI – User Interface – інтерфейс користувача.

UX – User Experience – досвід користувача.

RL – Reinforcement Learning – Навчання з підкріпленням.

IDE – Integrated Development Environment – інтегрована середа розробки.

HTTP – HyperText Transfer Protocol – протокол передачі гіпертексту.

ML – Machine Learning – машинне навчання.

GPT – Generative Pre-trained Transformer – генеративна попередньо навчена трансформерна модель.

GPU – Graphics Processing Unit – графічний процесор.

HTML – HyperText Markup Language – мова розмітки гіпертексту.

ВСТУП

Інтелектуальні агенти набувають все більшого поширення в сучасних комп’ютерних інтерактивних моделях, забезпечуючи автоматизацію завдань, інтеграцію з користувацькими системами та оптимізацію роботи в різних галузях, таких як освіта, бізнес, наука та індустрія розваг. Сьогодні існує значна кількість розробок, що дозволяють створювати інтелектуальних помічників із використанням сучасних методів штучного інтелекту (AI) та машинного навчання (МН), таких як OpenAI GPT [1], IBM Watson, Google Dialogflow тощо. Ці рішення демонструють значний прогрес у розумінні природної мови, автоматизації рутинних дій та взаємодії з іншими цифровими системами.

Однак, попри досягнутий прогрес, у галузі залишається низка викликів. Зокрема, проблема інтеграції таких агентів у складні комп’ютерні моделі, що потребують багатофункціональної взаємодії з користувачами та аналізу великого обсягу даних у реальному часі, ще не отримала достатньої уваги. Крім того, існують прогалини в стандартизації програмного забезпечення, забезпеченні зручності використання та універсальності таких систем для різних сфер застосування.

Світові тенденції спрямовані на розширення можливостей інтелектуальних агентів за рахунок розвитку технологій обробки природної мови (NLP), аналізу зображень, побудови моделей на основі нейронних мереж та підвищення ефективності інтеграції з різними програмними інтерфейсами (API). Інтелектуальні помічники стають більш адаптивними, здатними до самонавчання та персоналізації відповідно до потреб користувачів.

Актуальність даної роботи зумовлена необхідністю створення програмного забезпечення інтелектуального агента, який виконує функцію помічника в комп’ютерних інтерактивних моделях. Такий агент здатний

автоматизувати виконання складних дій, аналізувати користувацькі запити, інтегруватися з моделями обробки тексту та зображень, а також забезпечувати підвищення продуктивності користувача.

Метою роботи є розробка програмного забезпечення для інтелектуального агента, що інтегрується в комп’ютерні інтерактивні моделі та здатен ефективно вирішувати задачі автоматизації, аналізу даних та взаємодії з користувачами. Основні галузі застосування таких рішень включають освітні платформи, моделювання бізнес-процесів, інтерактивні системи підтримки користувачів та індустрію розваг.

1 ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ ІНТЕЛЕКТУАЛЬНИХ АГЕНТІВ

У розділі розглянуто сучасні підходи до створення інтелектуальних агентів, поняття, класифікацію та ключові характеристики цих систем. Проаналізовано роль інтелектуальних агентів у роботі з інтерактивними моделями, висвітлено основні технології, що використовуються для їх розробки, та окреслено виклики й перспективи їх впровадження.

1.1 Інтелектуальні агенти

Інтелектуальний агент – це автономна програмна або апаратна система, яка здатна сприймати інформацію із зовнішнього середовища, аналізувати її, приймати рішення та виконувати дії, спрямовані на досягнення заданих цілей. Такі агенти працюють на основі алгоритмів штучного інтелекту, що дозволяє їм імітувати людську поведінку у виконанні складних завдань.

Інтелектуальні агенти активно застосовуються у сферах, де необхідна автоматизація процесів, висока швидкість обробки даних та ефективне прийняття рішень. Наприклад, вони використовуються у медичних системах діагностики, фінансових аналітичних платформах, чат-ботах для технічної підтримки, а також у освітніх програмах.

Особливістю таких систем є їх здатність функціонувати в умовах невизначеності, коли вхідні дані можуть змінюватися, а алгоритм не є заздалегідь фіксованим. Це дозволяє агентам бути адаптивними, що значно розширює можливості їхнього використання.

Класифікація інтелектуальних агентів ґрунтується на їхній функціональності, архітектурі та принципах взаємодії з середовищем. Основні типи включають:

– реактивні агенти функціонують на основі простих правил, які визначають поведінку залежно від стану середовища та не зберігаючи

інформацію про попередні дії або події, а їхній алгоритм роботи базується на обробці поточних даних. Перевагою даних реактивних агентів є їхня швидкість та простота реалізації, проте вони не здатні вирішувати складні завдання, що потребують аналізу або навчання;

– когнітивні агенти – це системи, що мають здатність до навчання, що забезпечує адаптацію до нових умов роботи. Вони використовують базу знань та механізми штучного інтелекту для аналізу отриманих даних, прогнозування подій і прийняття рішень. Такі агенти можуть обробляти складні завдання, такі як розпізнавання мови, планування дій або моделювання поведінки користувача;

– багатоагентні системи – комплексні системи, що складаються з декількох взаємодіючих агентів. Кожен агент у системі може виконувати окреме завдання, тоді як загальний результат досягається через координацію їхніх дій. Такий підхід є ефективним для вирішення складних завдань, наприклад, у сфері логістики, де кожен агент відповідає за окремий етап процесу.

Є ряд основних характеристик, які показують ефективність інтелектуальних агентів, і одна з них – це автономність, яка дозволяє їм працювати без постійного контролю з боку людини. Ця характеристика включає здатність самостійно аналізувати середовище, приймати рішення та виконувати завдання, базуючись на отриманій інформації. Наприклад, автономний агент у системі розумного дому може регулювати температуру, освітлення чи інші параметри без втручання користувача.

Не менш важливим показником агента є адаптивність, завдяки чому він може легко змінюватися відповідно до нових умов або вимог середовища. Ця характеристика реалізується через використання методів машинного навчання, аналізу даних та побудови прогнозів. Наприклад, у освітніх програмах агент може адаптувати методику викладання залежно від рівня знань студента.

Здатність до комунікації дозволяє агентам взаємодіяти з іншими агентами, користувачами або зовнішніми системами для обміну інформацією. Ця характеристика є дуже важливою у багатоагентних системах, де ефективність роботи залежить від координації дій між агентами. Наприклад, у системі керування транспортом агенти можуть обмінюватися даними про дорожню ситуацію, щоб оптимізувати маршрути.

Всі наведені характеристики разом забезпечують високу ефективність інтелектуальних агентів, роблячи їх універсальними інструментами для вирішення складних завдань у різних галузях.

1.2 Інтерактивні комп’ютерні моделі

Інтерактивні комп'ютерні моделі є невід'ємною частиною сучасних цифрових технологій. Вони являють собою програмні або апаратно-програмні системи, які дозволяють користувачу взаємодіяти з модельованими процесами, об'єктами або середовищами. На відміну від статичних моделей, що демонструють тільки фіксовану інформацію, інтерактивні забезпечують зворотний зв’язок, що дозволяє користувачам впливати на хід симуляції, змінювати параметри й спостерігати за результатами у реальному часі.

Основною перевагою комп’ютерних інтерактивних моделей є універсальність, що дозволяє застосовувати їх у багатьох сферах людської діяльності. У цих моделях можна відтворювати як абстрактні концепції, так і реальні фізичні явища. Завдяки цьому такі моделі можна вважати потужним інструментом для навчання, тренування, аналізу та прогнозування.

В освітній сфері інтерактивні комп’ютерні моделі сприяють ефективному засвоєнню знань завдяки практичному досвіду. Вони дозволяють відтворювати складні концепції у візуальній та динамічній формі, що полегшує розуміння та сприяє глибшому засвоєнню матеріалу. Наприклад, у фізиці інтерактивні симуляції дозволяють студентам

досліджувати закони динаміки, змінюючи такі параметри, як маса чи швидкість об'єкта, і спостерігати за наслідками цих змін. У хімії подібні моделі дають змогу вивчати реакції між речовинами, не наражаючи учасників на ризик реального експерименту.

У сфері ігрової індустрії інтерактивні моделі являються основою для створення динамічних віртуальних світів. Вони забезпечують взаємодію гравця з об'єктами, персонажами та навколишнім середовищем. Завдяки цьому гравець має можливість впливати на розвиток сюжету чи досягати поставлених цілей. Наприклад, симулятори польотів або автоперегонів використовують інтерактивні моделі для відтворення реалістичної поведінки транспортних засобів у різних умовах.

У бізнесі інтерактивні комп’ютерні моделі сприяють аналізу та оптимізації процесів. Вони використовуються для прогнозування продажів, моделювання ринкових сценаріїв, тестування стратегій і прийняття управлінських рішень. Такі моделі дозволяють проводити «віртуальні експерименти», оцінювати потенційний результат без ризику для реального бізнесу.

Однією з найвідоміших інтерактивних платформ для освітніх цілей є PhET Interactive Simulations [2], яка розроблена Університетом Колорадо. Ця платформа пропонує широкий вибір інтерактивних симуляцій з фізики, хімії, біології, математики та інших наук. Вона має відкритий доступ і використовується в академічних установах та в самостійному навчанні.

Симуляції PhET розроблені з урахуванням потреб різних категорій користувачів: від школярів до студентів університетів. Їхня мета полягає у створенні ефективного навчального середовища, що стимулює активне засвоєння знань через експерименти та спостереження. Наприклад, у симуляції «Коливання маятника» користувач може змінювати різні параметри: довжину маятника, силу гравітації та початкову амплітуду, спостерігаючи за змінами у русі.

PhET інтегрує простий та інтуїтивний інтерфейс, що дозволяє

користувачам легко налаштовувати параметри симуляцій. Однією з ключових особливостей є візуалізація абстрактних концепцій, які складно уявити без відповідних графічних демонстрацій. Наприклад, симуляції з електрики дозволяють «бачити» потік електронів через провідники, що недоступно для спостереження у реальному житті.

Дана платформа підтримує різні мови, включно з українською, що робить її доступною для широкої аудиторії. Ця платформа активно використовується вчителями для створення інтерактивних уроків і лабораторних робіт, а також студентами для самостійного вивчення предметів.

Завдяки своїй універсальності та адаптивності, PhET є прикладом успішної реалізації інтерактивних комп'ютерних моделей у сучасній освіті. Вона демонструє, як технології можуть змінювати підходи до навчання, роблячи його більш ефективним і доступним.

Інтерактивні комп'ютерні моделі забезпечують інтеграцію наукових знань і технологій, створюючи середовище, що підтримує активне навчання, творчість і дослідження. Їхній потенціал у різних сферах є майже необмеженим, оскільки вони продовжують розвиватися та адаптуватися до нових викликів і потреб суспільства.

Інтелектуальні агенти відіграють важливу роль у вдосконаленні взаємодії користувачів з інтерактивними комп'ютерними моделями. Завдяки використанню алгоритмів штучного інтелекту та машинного навчання, такі агенти здатні адаптуватися до потреб користувачів, забезпечувати інтуїтивний інтерфейс і створювати більш ефективне та персоналізоване середовище для навчання, аналізу або розваг.

Основною перевагою інтелектуальних агентів можна відзначити здатність виявляти та розуміти наміри користувача. Вони аналізують дії, виконані у взаємодії з моделлю, і пропонують релевантні рекомендації, допомагають у вирішенні завдань або надають пояснення складних концепцій. В освітніх платформах інтелектуальні агенти можуть допомагати

учням орієнтуватися у складних симуляціях, пояснюючи їм послідовність дій або теоретичні основи експериментів, що вони виконують.

Інтелектуальні агенти також сприяють автоматизації рутинних дій, таких як налаштування параметрів моделей або пошук необхідних матеріалів. У складних системах вони можуть слугувати посередниками між користувачем і технічними елементами моделі, мінімізуючи технічні труднощі, з якими може зіткнутися некваліфікований користувач.

Одним із ключових аспектів є персоналізація взаємодії. Інтелектуальні агенти можуть аналізувати попередню поведінку користувача, зберігати його вподобання та використовувати ці дані для налаштування моделей відповідно до його потреб. В інтерактивних системах для бізнесу агент може наприклад аналізувати попередні сценарії роботи користувача з моделлю і пропонувати найкращі варіанти рішень для оптимізації бізнес-процесів.

Прикладом успішної реалізації є використання інтелектуальних агентів в інтерактивних симуляціях для навчання. На базі платформ, таких як PhET Interactive Simulations, інтелектуальні агенти можуть виконувати функції цифрових наставників та надавати учням інструкції, пояснюють результати симуляції або пропонують варіанти експериментів для поглиблення розуміння предмету. Така інтеграція не лише покращує засвоєння матеріалу та робить навчальний процес цікавішим і більш захоплюючим.

У бізнес-системах інтелектуальні агенти активно застосовуються для аналізу великих обсягів даних, що генеруються інтерактивними моделями. Вони дозволяють виявляти закономірності, прогнозувати тенденції та допомагають приймати стратегічні рішення. У моделюванні ринкових сценаріїв агенти можуть автоматично налаштовувати параметри симуляцій залежно від змін зовнішніх факторів і надавати звіти для керівників.

Ще одним прикладом є використання інтелектуальних агентів у системах охорони здоров’я. У цьому контексті інтерактивні моделі застосовуються для симуляції фізіологічних процесів або проведення віртуальних операцій. Інтелектуальні агенти допомагають медичним

фахівцям аналізувати результати симуляцій, прогнозувати наслідки рішень та оптимізувати план лікування.

У сфері розваг інтелектуальні агенти часто працюють як елементи віртуальних середовищ. Вони взаємодіють із гравцем, адаптуючись до його стилю гри та забезпечуючи високий рівень занурення. А у відеоіграх агенти можуть виконувати ролі персонажів, які навчають гравця або стають його партнерами у виконанні завдань.

Сучасні реалізації інтелектуальних агентів демонструють високий потенціал для подальшого вдосконалення взаємодії з інтерактивними моделями. Завдяки розвитку технологій штучного інтелекту та збільшенню обчислювальних потужностей, такі агенти стають все більш адаптивними, інтуїтивними та ефективними. Їх застосування у різних сферах сприяє покращенню доступності, функціональності та загальної якості інтерактивних систем.

1.3 Огляд технологій, що лежать в основі створення інтелектуальних агентів

Розробка інтелектуальних агентів спирається на досягнення у сфері штучного інтелекту (ШІ) та машинного навчання (МН), які забезпечують здатність агентів до самостійного аналізу, прийняття рішень і адаптації до нових умов. Серед основних технологій, що лежать в основі інтелектуальних агентів, можна виокремити глибоке навчання та навчання з підкріпленням.

Штучний інтелект визначається як здатність комп'ютерних систем виконувати завдання, які зазвичай вимагають людського інтелекту. До таких завдань належать розпізнавання зображень, природна мова, прогнозування та автоматизація процесів [3]. Основною метою ШІ є створення систем, здатних до самостійного прийняття рішень на основі аналізу даних. Існують три основні рівні ШІ:

– штучний вузький інтелект (ANI), який спеціалізується на виконанні

конкретних завдань (наприклад, голосові помічники чи системи рекомендацій);

– штучний загальний інтелект (AGI), що має здатність виконувати будь-які інтелектуальні завдання на рівні людини;

– штучний суперінтелект (ASI), який потенційно перевищує людський інтелект у всіх аспектах.

Наразі більшість інтелектуальних агентів базується на концепціях ANI, використовуючи алгоритми машинного навчання для вирішення прикладних задач.

Машинне навчання є підрозділом штучного інтелекту, що зосереджується на розробці алгоритмів, здатних навчатися на даних і вдосконалювати свої результати без прямого програмування [4]. Алгоритми машинного навчання поділяються на три основні категорії:

– навчання з учителем (supervised learning): модель навчається на розмічених даних, де кожен вхід відповідає певному результату. Прикладом є алгоритми регресії та класифікації;

– навчання без учителя (unsupervised learning): використовується для аналізу невідомих структур у даних, наприклад, кластеризація чи виявлення аномалій;

– навчання з підкріпленням (reinforcement learning): система навчається шляхом взаємодії з середовищем, отримуючи нагороди за правильні дії.

Глибоке навчання є підвидом машинного навчання, що використовує багаторівневі штучні нейронні мережі для моделювання складних взаємозв’язків у даних. Основною структурною одиницею таких моделей є нейрон, який імітує роботу біологічного нейрона.

Глибоке навчання знаходить широке застосування в задачах розпізнавання образів, обробки природної мови та прийняття рішень. Серед ключових архітектур можна виділити згорткові нейронні мережі (CNN), що спеціалізуються на обробці зображень і відео та рекурентні нейронні мережі (RNN) та їх вдосконалення (наприклад, LSTM або GRU), які

використовуються для аналізу часових рядів і текстів.

Особливістю глибокого навчання є потреба у великих обсягах даних та обчислювальних ресурсах. Використання графічних процесорів (GPU) значно підвищило ефективність тренування моделей.

Навчання з підкріпленням (Reinforcement Learning, RL) передбачає створення агентів, які вчаться шляхом проб і помилок у середовищі. Модель агента отримує зворотній зв’язок у вигляді нагород чи штрафів, що допомагає адаптувати поведінку для досягнення поставленої мети.

Основними компонентами RL є:

– агент, який приймає рішення;

– середовище, у якому діє агент;

– функція нагороди, що визначає успішність дій;

– логіка, яка описує стратегію вибору дій.

Сучасні підходи до навчання з підкріпленням включають використання глибоких нейронних мереж для представлення логіки чи функції оцінки. Такі методи отримали назву глибокого навчання з підкріпленням (Deep Reinforcement Learning, DRL).

Поєднання глибокого навчання та навчання з підкріпленням дозволяє створювати інтелектуальних агентів, здатних адаптуватися до складних умов та працювати у реальному часі. Вони здатні аналізувати дані, прогнозувати результати, приймати обґрунтовані рішення та навіть навчатися на основі попереднього досвіду.

Ці технології знайшли застосування у створенні агентів для автоматизації процесів, робототехніки, систем рекомендацій, віртуальних асистентів і симуляційних платформ, таких як PhET. Впровадження передових алгоритмів забезпечує підвищення точності, швидкості та ефективності роботи інтелектуальних агентів, роблячи їх ключовим елементом сучасних інтерактивних систем.

1.4 Проблеми та виклики створення інтелектуальних агенті

Серед ключових проблеми та викликів, що виникають при розробці інтелектуальних агентів можно приділити особливу увагу етичним аспектам, питанням безпеки, складнощам інтеграції із сучасними інформаційними системами, а також проблемам забезпечення високої точності розпізнавання дій користувачів.

Одним із основних викликів є забезпечення етичного використання інтелектуальних агентів. У процесі розробки виникають питання конфіденційності даних, які агенти збирають під час своєї роботи. Наприклад, системи, що працюють з інтерактивними моделями в освітньому середовищі, обробляють інформацію про навчальні досягнення, уподобання та поведінкові характеристики користувачів. Зловживання такими даними може призвести до порушення приватності.

Іншим важливим етичним аспектом є упередженість алгоритмів. Інтелектуальні агенти часто використовують дані з реального світу для навчання, проте ці дані можуть містити соціальні, економічні або культурні упередження. У результаті це може вплинути на прийняття рішень агентом, що є особливо критичним у сферах, пов’язаних із бізнесом або освітою.

Безпека також відіграє важливу роль, особливо у випадках, коли інтелектуальні агенти працюють у відкритих мережах або інтегруються з іншими сервісами. Загроза кібератак, витоків даних або маніпуляції діями агентів може поставити під загрозу стабільність системи та довіру користувачів. Розробники повинні забезпечити надійний захист інформації, використовуючи сучасні методи шифрування та системи автентифікації.

Інтеграція інтелектуальних агентів з існуючими програмними системами є складним технічним завданням. Більшість сучасних інформаційних систем були створені без врахування можливостей автоматизації, які забезпечують агенти, що створює певні бар’єри. Наприклад, у навчальних платформах можуть використовуватися застарілі

стандарти даних або обмежені інтерфейси програмування додатків (API), що ускладнює взаємодію з інтелектуальними агентами.

Ще однією проблемою є різноманітність інструментів, платформ і форматів даних. Агент повинен мати можливість ефективно взаємодіяти з різними мовами програмування, базами даних, системами зберігання інформації та інтерфейсами користувачів. Це вимагає створення адаптивних рішень, які дозволяють легко змінювати конфігурації або додавати нові функції без значних витрат ресурсів.

Також інтеграція часто ускладнюється через різні вимоги до продуктивності, наприклад, швидкість роботи агента або обсяги оброблюваних даних. У таких випадках необхідно оптимізувати архітектуру агентів і забезпечити баланс між якістю роботи та обмеженнями існуючих систем.

Не менш важливим викликом є складність розпізнавання дій і намірів користувачів. Інтерактивні моделі часто передбачають взаємодію в реальному часі, де користувач може використовувати широкий спектр дій: введення тексту, голосові команди, переміщення курсора, натискання кнопок тощо. Інтелектуальному агенту необхідно інтерпретувати ці дії в контексті завдання, що вимагає потужних алгоритмів розпізнавання та аналізу.

Складні ситуації виникають, коли дії користувачів не є однозначними або мають багато можливих інтерпретацій. Наприклад, у навчальних системах користувач може випадково пропустити певний етап завдання або ввести некоректну відповідь. Агент має не лише коректно ідентифікувати цю помилку, але й забезпечити підтримку, яка допоможе користувачеві продовжити роботу.

Для досягнення високої точності інтелектуальні агенти використовують машинне навчання, але навіть найкращі моделі не є безпомилковими. Проблеми виникають, коли система працює з нетиповими даними або стикається з незвичними ситуаціями, не врахованими на етапі навчання. Також важливо враховувати вплив змін у поведінці користувачів

на ефективність системи, що вимагає постійного вдосконалення моделей.

Розробка інтелектуальних агентів стикається з низкою важливих викликів, які впливають на їхню ефективність, безпеку та інтеграцію. Вирішення цих проблем вимагає міждисциплінарного підходу, що включає технічну, етичну та організаційну складові.

1.5 Перспективи розвитку інтелектуальних агентів

Інтелектуальні агенти є одним із ключових напрямів розвитку сучасних технологій і їхнє застосування у майбутньому обіцяє значні зміни в багатьох сферах людської діяльності. Потенційні сценарії використання таких систем стають дедалі ширшими, охоплюючи як повсякденне життя, так і професійні задачі. У майбутньому інтелектуальні агенти можуть перетворитися на багатофункціональних розумних асистентів, здатних допомагати в управлінні побутом, організації робочого часу та оптимізації навчальних процесів. Наприклад, у сфері освіти автоматизовані вчителі, що базуються на принципах персоналізованого навчання, зможуть адаптувати освітні програми під потреби кожного студента, надаючи миттєвий зворотний зв’язок та підтримуючи інтерес до навчання. У бізнесі агенти можуть стати незамінними аналітиками, які збиратимуть, оброблятимуть і інтерпретуватимуть великі обсяги даних, забезпечуючи керівників якісною інформацією для прийняття стратегічних рішень.

Тенденції у вдосконаленні технологій спрямовані на підвищення точності, швидкості обробки інформації та розширення функціональних можливостей агентів. Останнім часом активно розвиваються методи глибокого навчання та навчання з підкріпленням, що дозволяють агентам адаптуватися до складних і динамічних середовищ. Крім того, інтеграція агентів із хмарними технологіями забезпечує доступ до потужних обчислювальних ресурсів, що робить їх більш доступними навіть для невеликих підприємств і освітніх закладів. Удосконалюються також

інтерфейси взаємодії користувачів із агентами, включаючи розвиток природної мови, голосових команд і навіть невербальних способів спілкування.

Вплив інтелектуальних агентів на суспільство та економіку очікується глибоким і багатогранним. З одного боку, такі системи здатні підвищити продуктивність у багатьох галузях, знижуючи витрати на рутинні задачі та скорочуючи час, необхідний для виконання складних операцій. З іншого боку, виникають соціальні виклики, зокрема можливе скорочення робочих місць через автоматизацію певних професій. Одночасно інтелектуальні агенти сприятимуть появі нових сфер зайнятості, пов’язаних із розробкою, навчанням та підтримкою цих систем. У ширшому контексті такі технології можуть вплинути на суспільну структуру, сприяючи зростанню доступності освітніх і медичних послуг, підвищенню рівня життя та створенню умов для більшої соціальної рівності.

Таким чином, розвиток інтелектуальних агентів відкриває нові горизонти для технологічного прогресу, водночас ставлячи перед людством значущі виклики, які потребують усвідомленого і відповідального підходу до їх упровадження та використання.

1.6 Постановка мети та завдань дослідження

Метою кваліфікаційної роботи є створення програмного забезпечення інтелектуального агента, орієнтованого на використання в комп’ютерних інтерактивних моделях, зокрема для автоматизації взаємодії користувача з моделями та підвищення їх функціональних можливостей.

Відповідно до поставленої мети визначено наступні завдання:

– провести огляд сучасних інтерактивних комп’ютерних моделей і дослідити їх можливості для інтеграції з інтелектуальними агентами;

– проаналізувати вибір методів і технологій для розробки програмного забезпечення інтелектуального агента;

– розробити архітектуру програмного забезпечення інтелектуального агента та його основні модулі;

– провести тестування створеного програмного забезпечення для оцінки його ефективності та працездатності;

– проаналізувати результати розробки та підготувати рекомендації щодо подальшого вдосконалення системи.

Виконання поставлених завдань дозволить розробити інтегровану систему, що може використовуватися в освітніх сферах для покращення взаємодії користувачів із комп’ютерними інтерактивними моделями.

2 ПРОЕКТУВАННЯ СИСТЕМИ ІНТЕЛЕКТУАЛЬНОГО АГЕНТА

У другому розділі представлено процес проектування системи інтелектуального агента, яка забезпечує інтеграцію сучасних технологій штучного інтелекту для автоматизації дій користувача, створення інтерактивних інтерфейсів та поліпшення користувацького досвіду. Розділ включає огляд існуючих аналогів, детальний опис архітектури системи та проектування функціональних модулів. Особливу увагу приділено алгоритмам інтеграції інтелектуального агента із користувацьким інтерфейсом.

2.1 Огляд аналогів та існуючих систем інтелектуальних агентів

У сучасному цифровому світі інтелектуальні агенти стали невід'ємною частиною багатьох сфер діяльності, зокрема в автоматизації процесів, підтримці прийняття рішень, навчанні та взаємодії з користувачами. Різні системи інтелектуальних агентів мають свої особливості, функціонал та рівень інтеграції з існуючими інфраструктурами, залежно від сфери їх застосування.

Одним із прикладів є системи голосових помічників, такі як Apple Siri та Google Assistant [5]. Вони забезпечують інтерактивну взаємодію між користувачем та пристроями, дозволяючи виконувати прості дії, наприклад, пошук інформації, керування пристроями «розумного дому» або створення нагадувань. Основний акцент цих систем спрямований на розпізнавання голосу, аналіз запитів та забезпечення відповідей на основі баз знань чи зовнішніх джерел інформації. Втім, їх інтеграція в більш складні інтерфейси, такі як комп’ютерні моделі для навчання чи бізнес-аналітики, є обмеженою через вузьку спеціалізацію.

Ще одним типом агентів є чат-боти, наприклад, ChatGPT від OpenAI, які спеціалізуються на текстовій взаємодії з користувачем. Вони здатні відповідати на складні запитання, допомагати з поясненням концепцій чи навіть писати програмний код. Водночас їх застосування у конкретних інтелектуальних системах вимагає налаштування для конкретної області, оскільки загальні моделі часто не враховують специфічні вимоги інтерактивних комп’ютерних моделей.

Існують також системи автоматизації дій у веб-браузерах, наприклад, UiPath або Automation Anywhere, які дозволяють автоматизувати повторювані процеси на основі сценаріїв, створених користувачем. Вони є корисними для виконання специфічних завдань, таких як введення даних або обробка звітів, але не забезпечують інтуїтивної інтерактивної взаємодії з користувачем.

У сфері інтерактивних комп’ютерних моделей, зокрема навчальних симуляцій, розроблених у середовищі PhET Interactive Simulations, інтелектуальні агенти поки що використовуються обмежено. Основним завданням таких моделей є забезпечення доступного та інтуїтивного інтерфейсу для користувача, а інтеграція інтелектуальних агентів може значно підвищити ефективність взаємодії шляхом створення автоматизованих підказок, аналізу дій користувача та забезпечення персоналізованого досвіду навчання.

Аналіз існуючих аналогів свідчить, що сучасні системи інтелектуальних агентів демонструють високий рівень функціональності в окремих сферах, таких як автоматизація дій або текстова взаємодія, але їх інтеграція з комп’ютерними моделями залишається малодослідженою. Це створює перспективу для розробки спеціалізованих рішень, орієнтованих на взаємодію користувача з інтерактивними середовищами, такими як PhET, з урахуванням їх унікальних вимог і завдань.

2.2 Архітектура системи інтелектуального агента для комп'ютерних інтерактивних моделей

Архітектура системи інтелектуального агента для інтерактивних комп’ютерних моделей розроблена таким чином, щоб забезпечити ефективну обробку запитів користувача, автоматизацію різноманітних процесів, а також інтеграцію з різними функціональними модулями. Система включає в себе кілька взаємопов'язаних компонентів, кожен з яких відповідає за виконання специфічних завдань. Основними складовими архітектури є модуль взаємодії з користувачем через Frontend, модуль автоматизації дій у веб-браузері, модуль створення звітів, модуль для роботи з моделлю сегментації зображень, а також модуль інтеграції з ChatGPT для надання підказок та пояснень. Блок схему з основними модулями зображено на рисунку 2.1.

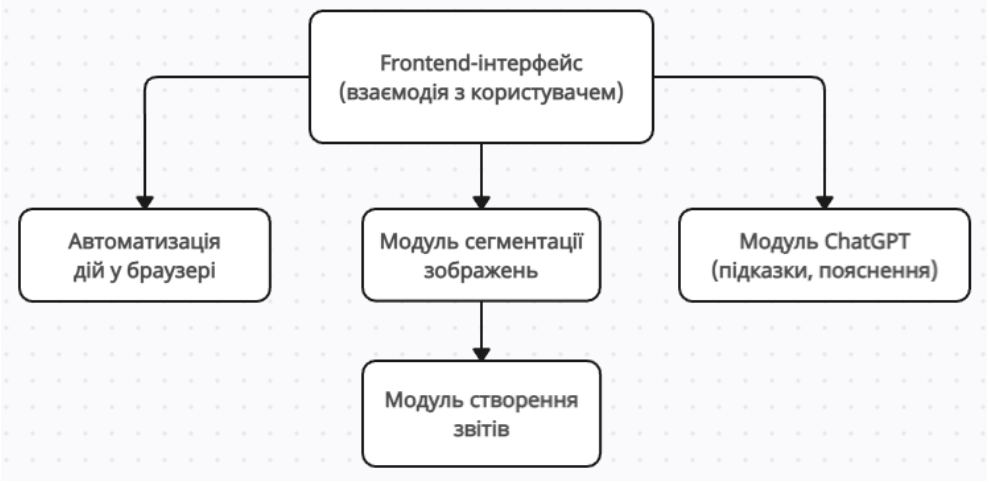

Рисунок 2.1 – Блок-схема основних модулей

Модуль взаємодії через Frontend (інтерфейс користувача) є основним інтерфейсом для користувача, через який відбувається вся взаємодія із системою. Фронтенд-інтерфейс надає користувачу доступ до усіх функцій системи, дозволяючи вводити дані чи переглядати результати роботи інтелектуального агента. Всі операції здійснюються завдяки дії на елементи

інтерфейсу (кнопки, форми), що ініціює відповідні запити до Backend-системи. Крім того, через цей інтерфейс здійснюється комунікація з модулем ChatGPT, який надає користувачеві необхідні підказки та пояснення. Інтерфейс є адаптивним, що дозволяє працювати з різними типами пристроїв від комп'ютерів до мобільних пристроїв.

Наступний модуль автоматизації дій у веб-браузері відповідає за виконання рутинних операцій в інтернет-середовищі. Це може бути автоматичний пошук, заповнення форм, натискання на елементи веб-сторінки чи інші типи взаємодії. Цей модуль бере на себе функції, що дозволяють значно знизити ручну працю користувача, виконуючи стандартні дії без його участі. Після ініціювання користувачем через інтерфейс системи, модуль використовує заздалегідь визначені сценарії для виконання необхідних маніпуляцій на веб-сторінках.

Модуль створення звітів відповідає за генерування структурованих документів на основі результатів обробки даних. Звіти будуть у вигляді результатів експерименту у вигляді зображень, що надають зручний спосіб представлення аналізу. Даний модуль використовує результати, що надаються іншими модулями, такими як модуль сегментації зображень.

Модуль сегментації зображень здійснює обробку скріншотів, отриманих від користувача. Завдання цього модуля виділити важливі елементи зображення. Безпосередньо провести сегментацію для виділення координат об'єктів на зображенні, що дозволяє коректно обрізати або ідентифікувати частини зображення, необхідні для подальшої обробки. В результаті цього модуля зображення стають придатними для включення в звіт.

Останній модуль відповідає за інтеграції з ChatGPT та забезпечує взаємодію користувача з системою через інтерактивний чат. Цей компонент допомагає користувачу отримати пояснення щодо використання системи, а також дає можливість ставити уточнюючі питання та отримувати рекомендації по темі.

Взаємодія між модулями системи побудована на основі чіткої логічної послідовності дій, де кожен модуль виконує свою специфічну функцію в загальному процесі обробки даних. Початковий етап взаємодії починається з фронтенд-модулю, який приймає введені користувачем дані та ініціює подальші дії. Наприклад, якщо користувач бажає створити звіт та зафіксувати певні результати експерименту, після вибору відповідного пункту, помічник автоматично передає запит до модуля для роботи з моделлю сегментації зображень, який обробляє зображення, визначає координати та обрізає його для подальшої роботи. Після обробки зображення, результати передаються до модуля створення звітів, який використовує ці дані для формування фінального звіту. У той самий час, система завдяки модулю для автоматизації дій у веб-браузері може виконати відповідні операції для налаштування симуляції чи використати модуль для отримання підказок та порад, який надає додаткові пояснення, допомагаючи користувачеві зрозуміти або скоригувати свої дії, що робить роботу з системою ще більш інтуїтивною та ефективною.

Система побудована таким чином, щоб кожен з компонентів взаємодіяв з іншими для досягнення загальної мети автоматизації обробки запитів і полегшення задач користувача в межах інтерактивних комп'ютерних моделей.

2.3 Розробка функціональних модулів системи

2.3.1 Модуль автоматизації дій у веб-браузері

Розробка модуля автоматизації дій у веб-браузері є ключовою частиною системи інтелектуального агента. Цей модуль відповідає за виконання автоматизованих дій у браузері, таких як відкриття вкладок, навігація між веб-сторінками, взаємодія з елементами інтерфейсу, перевірка стану браузера та управління його поведінкою.

Серед основних функціональних завдань модуля відзначити запуск і контроль браузера:

– створення інтерфейсу для запуску браузера в автоматизованому режимі;

– здійснення перевірки наявності активних сеансів браузера;

– забезпечення можливості закриття або перезапуску браузера.

Наступне завдання, яке має виконувати даний модуль, це управління вкладками:

– відкриття нових вкладок і закриття існуючих;

– перемикання між відкритими вкладками;

– перевірка активної вкладки та стану сторінки.

Автоматизація взаємодії з веб-елементами:

– пошук елементів на сторінці (кнопки, форми, посилання тощо) за допомогою селекторів (CSS, XPath);

– виконання дій, таких як клік, введення тексту, вибір значень зі списків або чекбоксів;

– виконання скриптів для динамічної обробки елементів.

Не менш важливим завданням для підтримки функціонування системи є обробка помилок і непередбачуваних ситуацій:

– виявлення збоїв у завантаженні сторінки чи непрацездатності елементів;

– запуск механізмів повторного виконання дій або повідомлення про помилки.

Інтеграція з іншими модулями:

– забезпечення обміну даними з іншими модулями системи через API або спільні структури даних;

– передача результатів виконаних дій для використання іншими модулями.

Структура модуля автоматизації буде складатися з двох основних

компонентів, перший це містить клас Browser, який відповідає за управління основними операціями браузера. Основні методи включають:

– ініціалізацію браузера;

– відкриття і закриття вкладок;

– управління вікнами та перевірка стану браузера.

Цей клас слугуватиме базовим шаром для роботи з браузером, використовуючи бібліотеку автоматизації Selenium [6].

А другий компонент буде складатися з класу ActionRecorder, який відповідає за виконання конкретних дій користувача. Основні методи включають:

– навігацію на певні URL-адреси;

– перевірку стану веб-сторінок;

– вибір і виконання дій, таких як клік на кнопки, введення даних або скидання налаштувань.

Інтеграція модуля в систему, модуль автоматизації дій у веб-браузері буде інтегрований із загальною системою через API або спільні структури даних. Він отримуватиме запити від модуля фронтенду або інших компонентів, виконуватиме необхідні дії та повертатиме результати. Цей підхід забезпечує модульність системи та полегшує розширення функціоналу модуля у майбутньому.

2.3.2 Модуль створення звітів

Модуль створення звітів є важливою частиною системи, яка забезпечує генерацію, структурування та збереження результатів експериментів або обробки даних. Його основна мета — спрощення процесу документування та систематизації результатів, що дозволяє користувачам отримувати інформацію в зручному та доступному форматі.

Цей модуль реалізований у вигляді класу ReportHandler, який містить методи для управління різними аспектами створення звітів. Основні функції

класу охоплюють локалізацію даних, додавання та видалення етапів експерименту, а також формування й збереження остаточного звіту.

Під час роботи модуля кожен експеримент або обробка даних структуруються у вигляді етапів, які користувач може додавати або видаляти відповідно до потреб. Клас також дозволяє зберігати отримані дані в форматах, зручних для подальшого використання чи аналізу.

При локалізації даних модуль автоматично визначає ключові елементи, які повинні бути включені до звіту, забезпечуючи високу точність обробки. Наприклад, координати важливих елементів або сегментовані зображення додаються до звіту разом із текстовим описом для повнішого розуміння результатів.

Ключовим аспектом є можливість динамічного створення структури звіту. Користувач може керувати змістом звіту, додаючи чи видаляючи етапи експерименту залежно від їхньої актуальності. Це гнучкість забезпечує адаптивність модуля до різних завдань, які вирішує система.

Модуль надає функціонал для збереження звітів у вигляді файлів із форматами .docx, що дозволяють інтеграцію з іншими системами. Це робить модуль універсальним інструментом для створення документів і обміну результатами з колегами або клієнтами.

Таким чином, модуль створення звітів виконує функцію зв’язувальної ланки між автоматизованими процесами системи та кінцевим користувачем, дозволяючи представити результати у зрозумілому вигляді. Його інтеграція з іншими модулями системи забезпечує комплексний підхід до виконання завдань.

2.3.3 Модуль для роботи з моделлю сегментації зображень

Модуль для роботи з моделлю сегментації зображень забезпечує автоматизацію процесу ідентифікації та виділення ключових областей на зображеннях. Його основне завдання полягає у використанні попередньо

навченої нейронної мережі для прогнозування координат та розмірів цікавих ділянок. Результати роботи модуля використовуються в інших частинах системи, таких як модуль створення звітів. Основою модуля є нейронна мережа, яка реалізована за допомогою бібліотеки PyTorch [7]. Архітектура моделі побудована таким чином, щоб забезпечувати ефективне розпізнавання координат об’єктів на зображеннях. Для цього використовується проста згорткова нейронна мережа, яка включає кілька шарів для обробки зображень, з подальшим їхнім зведенням до координат і розмірів об’єкта.

Функціонал модуля складається з попереднього навчання моделі, оскільки для роботи модуль вимагає заздалегідь навченої моделі, яка може бути створена на основі кастомного набору даних. Навчання буде передбачати:

– підготовку датасету, який складається із зображень і відповідних координат об’єктів;

– використання функцій втрат, таких як MSELoss, для мінімізації розбіжностей між передбаченими та реальними координатами;

– оптимізацію моделі за допомогою алгоритмів, таких як Adam, для забезпечення швидкої та стабільної збіжності.

Після завершення навчання модель зберігається в локальному сховищі, що дозволяє її подальше використання без необхідності повторного навчання. Модуль забезпечує функціонал для завантаження існуючої моделі, що значно пришвидшує інтеграцію в систему.

Та безпосередньо етап використання моделі для прогнозування, буде складатися з основного робочого процесу модуля, що включає завантаження зображень, обробку їх за допомогою моделі та отримання результатів у вигляді координат і розмірів області інтересу та подальшої сегментації. Процес прогнозування:

– здійснюється на одному чи кількох зображеннях;

– передбачає нормалізацію та перетворення вхідних даних у формат, сумісний із моделлю;

– виконує прогноз із подальшим масштабуванням координат до вихідних розмірів зображення.

Результати роботи модуля передаються до інших компонентів системи, таких як модуль створення звітів. Отримані координати використовуються для обрізки зображень завдяки функціям у моделі та в подальшому у звіті візуалізації ключових областей.

Алгоритм роботи можна розбити на такі етапи:

– завантаження навченої моделі з локального сховища;

– передача вхідного зображення або набору зображень через функцію обробки;

– отримання передбачених координат об’єктів;

– масштабування передбачених координат до реальних розмірів зображення;

– передача результатів у вигляді координат і розмірів для подальшого використання;

– сегментація зображення та використання його в інших модулях.

Модуль для роботи з моделлю сегментації забезпечує високу точність прогнозування завдяки використанню сучасних технологій глибокого навчання. Його інтеграція в систему автоматизує завдання виділення ключових областей, зменшуючи ручну роботу користувачів і підвищуючи загальну ефективність.

2.3.4 Модуль інтеграції з ChatGPT

Модуль інтеграції з ChatGPT [8] призначений для автоматизації взаємодії з системою на основі штучного інтелекту для отримання текстових відповідей, пояснень або рекомендацій. Його основна мета — забезпечити підтримку користувача шляхом обробки запитів у вигляді текстових повідомлень і надання зрозумілих відповідей у реальному часі.

Модуль побудовано таким чином, щоб забезпечити двосторонню

комунікацію з моделлю ChatGPT. В основі функціональності лежить обмін текстовими запитами й відповідями через API, що дозволяє інтегрувати модуль у різні частини системи або використовувати його як окремий інструмент для підтримки. Серед основних функцій модулю можна виділити формування текстових запитів, користувач або інші модулі системи формують текстовий запит, який надсилається до моделі ChatGPT. Такі запити можуть містити:

– вказівки для роз'яснення певних процесів;

– питання щодо специфічних функціональних аспектів;

– запити на отримання загальної інформації або підказок;

– обробка відповідей.

Після надсилання запиту модуль отримує текстову відповідь від ChatGPT. Ця відповідь може бути одразу передана користувачу або використана іншими модулями для подальшої обробки. Модуль буде відповідати відповідає за форматування отриманих даних для зручного відображення та за необхідності видалення непотрібної інформації.

Модуль інтеграції дозволяє користувачам взаємодіяти з системою в інтерактивному режимі, ставлячи уточнювальні питання або змінюючи запити, щоб отримати детальнішу відповідь.

Інтеграція дозволяє також враховувати контекст попередніх запитів. Це забезпечує логічний зв’язок між послідовними відповідями, створюючи більш природний і ефективний досвід спілкування.

Алгоритм роботи модуля матиме такі етапи:

– користувач або модуль системи формує текстовий запит;

– запит передається через API до моделі ChatGPT;

– модуль отримує відповідь і проводить її попередню обробку;

– відповідь передається користувачеві або іншому модулю системи для подальшого використання.

Модуль інтеграції з ChatGPT буде виконувати роль універсального консультанта, який допомагає роз'яснювати складні аспекти роботи системи

та надавати інформаційну підтримку під час виконання завдань.

Серед переваг даної інтеграція з ChatGPT можна відмітити забезпечення доступ до великого обсягу знань, що може бути корисним у процесі використання системи. Також замість залучення фахівців технічної підтримки, користувач може отримати допомогу безпосередньо в інтерфейсі системи, та безпосередньо користувач матиме змогу ставити уточнювальні питання та отримувати відповіді в реальному часі.

Модуль може бути використаний як у рамках окремих функціональних процесів системи, так і в інших інтеграціях, де потрібна текстова підтримка або пояснення.

Цей модуль робить систему більш доступною, зрозумілою та зручною для користувачів, особливо в ситуаціях, які потребують додаткових пояснень чи уточнень.

2.3.5 Модуль інтерфейсу користувача

Модуль інтерфейсу користувача (UI) є важливим компонентом системи, який забезпечує взаємодію користувача з усіма функціональними модулями. Основна мета даного модуля надати зручний і інтуїтивно зрозумілий спосіб доступу до функціональності системи, включаючи введення даних, перегляд результатів і управління процесами [9].

Інтерфейс користувача буде складатися з основної форми, яка організована для виконання ключових завдань. Вона включає текстові поля, кнопки та інші інтерактивні елементи, що дозволяють виконувати такі дії, як надсилання запитів до ChatGPT, управління процесами автоматизації в браузері, створення та управління звітами, а також локалізацію інтерфейсу.

Основні функціональні елементи інтерфейсу забезпечують інтерактивну взаємодію користувача з системою через зручну і зрозумілу форму. Центральним елементом є текстове поле для введення інформації, яке дозволяє користувачам надсилати запити до модуля інтеграції з ChatGPT або

вводити інші дані для роботи з функціональними модулями. Це поле підтримує динамічний текстовий ввід, який може бути миттєво оброблений системою.

Для виводу результатів передбачене поле, в якому відображаються відповіді від ChatGPT, статус виконання дій у браузері чи звіт про успішне створення або збереження документа. Поле виводу забезпечує наочне представлення даних, що дозволяє користувачеві швидко оцінити результат виконаної дії.

Крім текстових полів, важливим елементом інтерфейсу є набір кнопок, які виконують різноманітні функції. Зокрема, кнопки для надсилання запитів, очищення текстових полів, управління браузером чи модулем автоматизації дають змогу швидко виконувати типові операції. Кнопки для роботи зі звітами дозволяють додавати або видаляти етапи експерименту, зберігати створений звіт чи видаляти його, а також забезпечують простоту управління даними звітності.

Особливої уваги заслуговує кнопка для локалізації, яка дозволяє перемикати мову інтерфейсу, адаптуючи його до потреб користувача. Це підвищує зручність використання системи для ширшого кола користувачів. Всі елементи інтерфейсу працюють синхронно, забезпечуючи чітку і злагоджену взаємодію між користувачем і функціональними модулями системи.

Процес роботи користувача з інтерфейсом побудований на простій та логічній послідовності дій, що забезпечують ефективну взаємодію з усіма функціональними модулями системи.

Користувач розпочинає роботу, вводячи текстовий запит у відповідне поле вводу, для звернення до ChatGPT та отримання роз’яснень Після введення даних користувач натискає одну з інтерактивних кнопок, відповідну до бажаної дії. Наприклад, при роботі з ChatGPT запит буде надсилатися до відповідного модуля, який миттєво обробляє текст і генерує відповідь. Якщо ж виконується автоматизація у браузері, інтерфейс ініціює

запуск відповідної функції модуля автоматизації, яка здійснює потрібну дію, наприклад, взаємодію із веб-сайтами або налаштування параметрів. У випадку роботи зі звітами користувач може додати новий етап експерименту, зберегти сформований звіт або видалити застарілі дані, просто натиснувши потрібну кнопку.

Результати виконання дій миттєво відображаються у полі виводу або виводяться у вигляді сповіщень. Наприклад, користувач отримує текстову відповідь від ChatGPT, повідомлення про успішне виконання автоматизації в браузері чи підтвердження збереження звіту. Це забезпечує зворотний зв’язок і дозволяє оцінити результат кожної взаємодії.

Якщо виникає потреба повторити дії або почати новий процес, користувач може очистити всі текстові поля за допомогою спеціальної кнопки та вводити нові дані. Така організація алгоритму взаємодії буде забезпечувати гнучкість і простоту у використанні, роблячи систему доступною навіть для недосвідчених користувачів.

Модуль інтерфейсу користувача буде ядром взаємодії системи з користувачем, забезпечуючи зручність і функціональність у використанні різних компонентів системи для досягнення мети.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЄКТУ

У цьому розділі детально описано процес розробки програмного забезпечення, включаючи вибір інструментів, налаштування середовища, реалізацію функціональних модулів, тестування та локалізацію. Обґрунтовано вибір сучасних технологій для забезпечення функціональності та зручності роботи системи, а також демонстрація готового функціоналу для оцінки результатів.

3.1 Вибір інструментів і середовища розробки

Процес вибору інструментів і середовища розробки є важливим етапом створення програмного забезпечення, оскільки правильний вибір технологій безпосередньо впливає на ефективність, масштабованість і зручність підтримки системи. Для реалізації даного проєкту було обрано мову програмування Python та низку спеціалізованих бібліотек і платформ, які найкраще відповідають вимогам і цілям проєкту.

Мова програмування Python є основою розробки через її численні переваги, серед яких виділяються простота та читабельність синтаксису, а також широка екосистема бібліотек, що дозволяє вирішувати різноманітні завдання. Python активно використовується в галузі машинного навчання, аналізу даних та веб-розробки, що робить його універсальним інструментом для реалізації багатофункціональних систем. Крім того, Python має розвинену спільноту, яка надає доступ до великої кількості навчальних матеріалів, документації та готових рішень, що сприяє швидкому розвитку проєкту та зменшує витрати на розробку.

Для створення серверної частини було обрано мікрофреймворк Flask [10]. Цей фреймворк забезпечує високу гнучкість у розробці, дозволяючи створювати масштабовані та продуктивні веб-додатки. Flask надає

можливість легко інтегрувати сторонні модулі, а його мінімалістичний підхід сприяє оптимізації системи. Його використання в рамках цього проєкту дозволяє ефективно організувати взаємодію між користувачем та функціональними модулями системи, включаючи обробку запитів і передачу даних до відповідних компонентів.

Для розробки графічного інтерфейсу користувача було обрано бібліотеку PyQt5 [11]. Цей інструмент забезпечує широкий набір функціональних можливостей для створення професійних графічних інтерфейсів. PyQt5 дозволяє розробляти зручні та інтуїтивно зрозумілі інтерфейси, які відповідають сучасним стандартам UI/UX-дизайну. Крім того, бібліотека має вбудовану підтримку роботи з різними платформами, що дозволяє легко адаптувати систему для запуску на різних операційних системах. Використання PyQt5 гарантує, що інтерфейс буде стабільним, стильним і функціональним.

Для реалізації компонентів, пов’язаних із машинним навчанням, обрано бібліотеку Torch. Torch є одним із провідних інструментів для навчання та розгортання нейронних мереж, що робить його ідеальним вибором для задач сегментації зображень. Torch надає ефективні механізми роботи з великими обсягами даних та оптимізації моделей, а його підтримка апаратного прискорення через GPU дозволяє значно скоротити час навчання. Крім того, бібліотека забезпечує високу точність передбачень, що є критично важливим для забезпечення коректної роботи системи.

Комплексний підхід до вибору інструментів також передбачає врахування їхньої сумісності. Python, Flask, PyQt5 та Torch гармонійно інтегруються, створюючи єдину екосистему для розробки, тестування та підтримки програмного забезпечення. Цей набір інструментів забезпечує високу продуктивність, гнучкість у налаштуванні та масштабуванні, а також простоту підтримки та подальшого розвитку системи. Таким чином, вибір вказаних технологій є обґрунтованим і відповідає меті створення функціонального та надійного програмного продукту.

3.1.1 Flask як серверна платформа

Flask обрано як серверну платформу для цього проєкту завдяки його мінімалістичній архітектурі, гнучкості та здатності ефективно вирішувати задачі, пов’язані з обробкою запитів і організацією взаємодії між користувачем та функціональними модулями системи. Основна роль Flask у даному проєкті — забезпечити зв’язок між фронтенд-компонентами та бекенд-логікою, яка виконує обчислювальні завдання, обробку даних та інтеграцію з зовнішніми сервісами.

Flask забезпечує легкість створення веб-додатків, оскільки побудований на основі WSGI (Web Server Gateway Interface) та використовує шаблон Jinja2 для рендерингу HTML-сторінок. У цьому проєкті він використовується для створення API-інтерфейсів, які слугують посередником між користувачем і внутрішніми модулями системи.

Однією з ключових причин вибору Flask є його модульна структура. Flask надає базовий функціонал, залишаючи можливість розширення через спеціалізовані бібліотеки. У межах проєкту це дозволяє інтегрувати модулі автоматизації, обробки зображень, створення звітів та взаємодії з ChatGPT без значного збільшення складності системи.

Flask підтримує стандартні HTTP-запити (GET, POST, PUT, DELETE), що забезпечує високу гнучкість у роботі з даними. Наприклад, у даному проєкті запити типу POST використовуються для передачі даних, введених користувачем через графічний інтерфейс, до серверної частини. Серверна частина, у свою чергу, викликає відповідні функції, такі як виконання запитів до ChatGPT, сегментація зображень або генерація звітів, і повертає результат у форматі JSON.

Flask реалізує концепцію маршрутизації, яка дозволяє створювати чітко визначені точки доступу до різних функціональних модулів. Кожен маршрут відповідає певній операції, наприклад, обробці текстового запиту,

завантаженню зображення або отриманню статусу виконання завдання.

Інтеграція з інтерфейсом користувача: Користувачі взаємодіють із графічним інтерфейсом, створеним у PyQt5, який формує запити та надсилає їх на сервер Flask. Сервер приймає ці запити, перевіряє їх коректність і пересилає до відповідного модуля для виконання.

Взаємодія з функціональними модулями: Flask виступає посередником, забезпечуючи доступ до таких компонентів, як модель сегментації зображень, модуль автоматизації в браузері та модуль створення звітів. Кожна функція в модулі має свій маршрут, що спрощує управління запитами.

Форматування результатів: Результати виконання обробляються сервером і повертаються користувачеві в структурованому вигляді. Наприклад, це може бути текстова відповідь від ChatGPT, згенерований звіт у форматі PDF або координати сегментації зображення.

Flask має мінімальні вимоги до ресурсів, що робить його оптимальним для проєктів із високим навантаженням і обмеженими обчислювальними ресурсами. Це особливо важливо для систем, які одночасно обробляють запити з кількох модулів. Простота розгортання Flask на різних платформах, включно з локальними серверами, хмарними інфраструктурами або контейнерами Docker, забезпечує додаткову гнучкість у підтримці та масштабуванні системи.

Flask також забезпечує високу безпеку, завдяки можливості використання таких механізмів, як шифрування даних, авторизація та захист від атак типу CSRF. У рамках проєкту ці функції можуть бути використані для захисту даних користувача та забезпечення стабільної роботи системи.

У результаті використання Flask дозволяє створити добре структуровану, ефективну та легку в обслуговуванні серверну частину, що ідеально відповідає вимогам інтерактивної системи з багатофункціональними модулями.

3.1.2 PyQt5 для створення графічного інтерфейсу

PyQt5 обрано для створення графічного інтерфейсу користувача завдяки його потужним можливостям, гнучкості у створенні складних GUI-компонентів та підтримці широкого спектра платформ. Ця бібліотека надає інструменти для створення професійного та інтуїтивно зрозумілого інтерфейсу, який відповідає потребам сучасних інтерактивних систем.

PyQt5 є однією з найпопулярніших бібліотек для створення графічних інтерфейсів у Python. Вона базується на Qt — потужному інструменті для розробки кросплатформених застосунків [12]. Це робить PyQt5 ідеальним вибором для проєктів, які потребують стабільності та адаптивності до різних операційних систем.

У цьому проєкті PyQt5 використовується для створення головного вікна програми, яке забезпечує взаємодію користувача з усіма функціональними модулями. Інтерфейс включає наступні ключові компоненти:

– текстове поле для введення запитів: Користувач вводить текстову інформацію, яка потім передається на сервер Flask для обробки;

– поле для відображення результатів: отримана відповідь або оброблені дані показуються у візуально зручному вигляді;

– кнопки для управління функціоналом: Кнопки надають можливість взаємодії з модулями автоматизації браузера, створення звітів та надсилання запитів до ChatGPT;

– меню налаштувань: Цей компонент дозволяє змінювати параметри системи, наприклад, вибір мови локалізації або активацію певних модулів.

PyQt5 забезпечує багатий набір інструментів для створення як базових, так і складних графічних елементів. У даному проєкті це дозволяє об'єднати різні функціональні можливості в єдиний, зручний для користувача інтерфейс. Додаткові переваги включають:

– кросплатформенність: завдяки підтримці різних операційних систем

застосунок може бути розгорнутий на будь-якій платформі без значних змін у коді;

– підтримка багатопоточності: PyQt5 дозволяє легко інтегрувати багатопоточність, що особливо важливо для роботи з великими обчислювальними задачами, такими як сегментація зображень або генерація звітів, без блокування інтерфейсу;

– інтеграція з іншими бібліотеками Python: PyQt5 чудово поєднується з іншими бібліотеками, такими як Flask, що дозволяє ефективно реалізувати обмін даними між інтерфейсом і бекендом.

PyQt5 використовує сигнал-слот архітектуру, що дозволяє ефективно обробляти події, такі як натискання кнопок або введення тексту. Це дозволяє створити інтерфейс, який швидко реагує на дії користувача та інтегрує відповідні функціональні модулі. Наприклад:

– при натисканні кнопки для автоматизації в браузері надсилається відповідний запит на сервер Flask, який активує функції модуля автоматизації;

– відповідь, отримана від ChatGPT, відображається в текстовому полі через функцію, пов'язану із сигналом завершення обробки запиту.

PyQt5 є оптимальним вибором для створення графічного інтерфейсу в цьому проєкті. Його можливості забезпечують зручну реалізацію всіх необхідних функціональних компонентів, включаючи введення та відображення даних, інтеграцію з іншими модулями та адаптацію до потреб користувачів. Це рішення дозволяє створити сучасний, естетично привабливий та ефективний інтерфейс, що є важливим аспектом успішної взаємодії між системою та користувачем.

3.1.3 Torch для навчання моделей штучного інтелекту

PyTorch був обраний як основний інструмент для розробки та навчання моделей штучного інтелекту завдяки його широким можливостям, зручності

у використанні та високій продуктивності. Цей фреймворк відомий своєю гнучкістю, що дозволяє легко створювати та адаптувати нейронні мережі для різноманітних задач, включаючи сегментацію зображень, класифікацію та прогнозування.

PyTorch є однією з провідних бібліотек для роботи з машинним навчанням і глибоким навчанням. Його архітектура дозволяє працювати з динамічними обчислювальними графами, що робить процес відладки та оптимізації моделей більш простим і інтуїтивно зрозумілим. У контексті даного проєкту, PyTorch забезпечує такі переваги:

– зручність розробки: завдяки простому синтаксису, бібліотека дозволяє швидко створювати складні моделі та інтегрувати їх у загальну систему;

– висока продуктивність: PyTorch підтримує апаратне прискорення через GPU, що значно скорочує час навчання моделі навіть на великих наборах даних;

– активна спільнота: велику кількість прикладів, документації та попередньо навчених моделей можна знайти у відкритому доступі, що сприяє швидкому розвитку проєкту.

В даному проєкті PyTorch використовується для побудови та навчання нейронної мережі, яка виконує сегментацію зображень. Основна мета моделі прогнозування координат для виділення областей, що потребують подальшої обробки. Це досягається через тренування нейронної мережі на основі реальних даних, отриманих із зображень. Процес побудови моделі включає:

– створення архітектури нейронної мережі: в проєкті використовується згорткова нейронна мережа (CNN), яка є ідеальним вибором для аналізу зображень. Архітектура розроблена так, щоб забезпечити баланс між точністю прогнозу та швидкістю обробки;

– обробка та підготовка даних:всі зображення передаються через стадії нормалізації та масштабування, що гарантує стабільність процесу навчання;

– навчання моделі: за допомогою PyTorch модель оптимізується через

мінімізацію функції втрат, використовуючи методи градієнтного спуску. В процесі навчання досягається покращення точності передбачень завдяки ітеративній обробці набору даних.

PyTorch забезпечує легку інтеграцію з іншими компонентами системи. Після завершення процесу навчання модель зберігається у форматі .pth, що дозволяє використовувати її в модулі обробки зображень. Завдяки цьому, модель може працювати в реальному часі, виконуючи прогнозування на основі нових вхідних даних.

Для оптимізації взаємодії з інтерфейсом користувача результати роботи моделі перетворюються в зрозумілі формати, які використовуються в модулі генерації звітів. Це дозволяє забезпечити єдиний потік даних між різними компонентами системи.

PyTorch є незамінним інструментом для реалізації задач, пов’язаних із штучним інтелектом, у цьому проєкті. Його можливості дозволяють створювати продуктивні, адаптивні та інтуїтивно зрозумілі моделі, які забезпечують високий рівень точності та ефективності. Вибір PyTorch забезпечує гнучкість та масштабованість системи, що є важливими аспектами для її подальшого розвитку та інтеграції нових функціональних можливостей.

3.2 Налаштування середовища розробки

Для реалізації програмного забезпечення було обрано IDE Visual Studio Code (VS Code). Цей вибір обумовлений його високою функціональністю, широкою підтримкою технологій, гнучкістю в налаштуванні та відповідністю сучасним вимогам до розробки складних програмних рішень.

Visual Studio Code є потужним інструментом для розробки програмного забезпечення завдяки своїм ключовим перевагам:

– широка підтримка мови Python, завдяки розширенням, таким як Python Extension, забезпечується автоматична перевірка синтаксису, зручна

робота з форматуванням коду та можливість інтеграції з утилітами для тестування;

– інтеграція з інструментами: VS Code дозволяє легко налаштовувати інтеграцію з Git, Docker, а також іншими інструментами, що робить його ідеальним для командної розробки та деплойменту;

– можливості дебагінгу: вбудований відлагоджувач спрощує діагностику помилок, що є критично важливим при розробці багатокомпонентної системи, такої як ця;

– кросплатформеність: підтримка Windows, macOS і Linux дозволяє розробникам працювати у звичному середовищі без необхідності додаткового налаштування;

– оптимізація для великих проєктів: потужна система пошуку та управління файлами спрощує навігацію по великих кодових базах.

Ці переваги роблять VS Code оптимальним вибором для реалізації проєкту, що поєднує серверну розробку, графічний інтерфейс і машинне навчання.

Щоб дотримуватись стандартів Python (PEP 8) [13] і забезпечити читабельність коду, всі імпорти у головному файлі main.py повинні бути структуровані у наступному порядку:

– стандартні бібліотеки Python;

– бібліотеки сторонніх виробників (third-party libraries);

– локальні модулі проєкту.

Відповідно до цієї структури, вигляд імпорту бібліотек та моделей представлено у лістингу 3.1.

Лістинг 3.1 – Імпорт бібліотек та модулей

```
# Стандартні бібліотеки Python
import os
import sys
import json
from datetime import datetime
```

```
# Бібліотеки сторонніх виробників
from flask import Flask, request, jsonify, render_template
from PyQt5.QtWidgets import (
     QApplication, QMainWindow, QTextEdit, QVBoxLayout,
QPushButton, QWidget, QLineEdit, QHBoxLayout
     )
from PyQt5.QtCore import Qt
import requests
from PIL import Image
import torch
from torch.utils.data import DataLoader
from torch import nn, optim
# Локальні модулі проєкту
from backend.browser import Browser
from backend.actions import ActionRecorder
from backend.gpt_integration import GPTAssistant
from backend.report_handler import ReportHandler
from model import SimpleCNN
from dataset import ScreenshotDataset, transform
```

Налаштування середовища розробки є критичним етапом, який забезпечує готовність системи до роботи. У цьому процесі враховані всі аспекти інтеграції компонентів, підготовки залежностей та структурування проєкту. Для забезпечення швидкого та коректного налаштування системи реалізовано функцію install_dependencies(). Вона автоматизує процес встановлення залежностей, перелічених у файлі requirements.txt. Ця функція використовує модуль subprocess для виконання команд терміналу через Python. У лістингу 3.2 наведено код цієї функції.

Лістинг 3.2 – Код функції install_dependencies()

```
def install_dependencies():
    try:
        subprocess.check_call([os.sys.executable, "-m", "pip",
"install", "-r", "requirements.txt"])
        print("All dependencies are installed.")
    except Exception as e:
        print(f"Error installing dependencies: {e}")
        exit(1)
```

При першому запуску програми функція install_dependencies() викликається автоматично. Вона гарантує, що всі необхідні пакети, такі як

Flask, PyQt5, Selenium, OpenAI API та інші, будуть встановлені в актуальних версіях. Завдяки цьому користувачеві або розробнику не потрібно вручну виконувати процес встановлення залежностей.

Проєкт структуровано таким чином, щоб забезпечити логічний поділ функціональності, спрощення навігації та масштабованість. Нижче на рисунку 3.1 представлена структура проєкту з описанням компонентів:

```
├── main.py                # Головний файл, що запускає систему.
├── backend/
│   ├── browser.py         # Логіка взаємодії з сайтом (Selenium).
│   ├── actions.py         # Виконання та запис дій користувача.
│   ├── reports_handle.py  # Генерація звітів.
│   └── gpt_integration.py # Інтеграція з OpenAI API.
├── frontend/
│   ├── index.html         # HTML для користувацького інтерфейсу.
│   ├── style.css          # CSS-стилі для інтерфейсу.
│   └── script.js          # JavaScript для динамічного функціоналу.
├── data/
│   ├── complete/          # Тимчасове збереження готових скриншотів.
│   ├── screenshots/       # Зберігання необроблених скриншотів.
│   ├── training_data/     # Дані для тренування моделей.
│   ├── config.json        # Конфігурація системи.
│   └── links.json         # Налаштування та посилання на сайти.
├── model/
│   ├── dataset.py         # Завантаження даних для навчання моделі.
│   ├── model.py           # Опис нейронної мережі.
│   └── photo_processing.py # Підготовка даних для тренування.
├── models/                # Збережені моделі.
├── reports/               # Готові звіти, створені користувачами.
├── requirements.txt       # Список залежностей.
└── README.md              # Опис проєкту.
```

Рисунок 3.1 – Структура проєкту

Проєкт має чітко визначену структуру, яка забезпечує легкість навігації, модульність і розширюваність. Основним файлом є main.py, який об'єднує всі компоненти системи та відповідає за запуск ключових функцій, таких як серверна частина на Flask, графічний інтерфейс на PyQt5 і модулі для обробки даних. Цей файл виконує роль центральної точки взаємодії між різними частинами проєкту.

У каталозі backend розміщено логіку, яка стосується роботи з даними, браузером і API. Наприклад, файл browser.py містить код для автоматизації дій у браузері за допомогою Selenium, тоді як actions.py реалізує функції запису й відтворення дій користувача. Окремий файл gpt_integration.py відповідає за інтеграцію з OpenAI API, забезпечуючи обробку текстових запитів, а reports_handle.py містить функціонал для створення звітів, що дозволяє автоматизувати аналіз та структурування отриманих даних.

Фронтальна частина системи знаходиться в каталозі frontend, де зберігаються файли HTML, CSS і JavaScript. Вони забезпечують створення користувацького інтерфейсу, який відображається через браузер. HTML-файли формують структуру сторінки, CSS надає їй стиль, а JavaScript забезпечує динамічність і взаємодію з серверною частиною через AJAX-запити.

Дані проєкту організовані у каталозі data. Тут знаходяться підкаталоги для зберігання скріншотів, тренувальних даних для моделей і тимчасових файлів, таких як готові скріншоти. Окрім цього, у каталозі зберігаються файли конфігурації, які містять основні налаштування системи (config.json) і специфічні параметри для роботи з окремими веб-сайтами (links.json).

Каталог model містить усі файли, пов’язані з обробкою та аналізом даних за допомогою машинного навчання. Наприклад, файл dataset.py реалізує завантаження і підготовку даних для навчання нейронної мережі, а model.py містить архітектуру моделі простої згорткової нейронної мережі. Також у цьому каталозі є файл photo_processing.py, який виконує попередню обробку зображень, необхідну для ефективного навчання моделі.

Для зберігання готових моделей і звітів створено відповідні каталоги — models і reports. Вони дозволяють структурувати результати роботи системи, забезпечуючи зручний доступ до збережених даних.

Список залежностей проєкту зберігається у файлі requirements.txt. У ньому перелічено всі необхідні бібліотеки, такі як Flask, PyQt5, Selenium та інші. Це гарантує, що всі компоненти системи будуть сумісними між собою,

а налаштування нового середовища відбуватиметься швидко й без помилок.

Така структура забезпечує не лише логічний поділ функціоналу, але й легкість у підтримці й розширенні проєкту, дозволяючи розробникам фокусуватися на ключових аспектах роботи системи.

3.3 Реалізація програмних модулів

3.3.1 Модуль автоматизації дій у веб-браузері

Модуль автоматизації дій у веб-браузері реалізований для спрощення взаємодії з веб-інтерфейсами через автоматизацію дій, таких як увімкнення або скидання налаштувань. Цей функціонал інтегрований у систему через Flask-додаток, клас для дій із браузером і взаємодію з інтерфейсом користувача. Ключові компоненти модуля включають обробку запитів через Flask, класи для взаємодії з веб-браузером і автоматизацію дій через координати, збережені у конфігураційних файлах.

Flask-додаток приймає запити на виконання автоматизованих дій через два маршрути: /enable-settings для увімкнення налаштувань і /reset-experiment для скидання симуляції. У функції enable_settings здійснюється перевірка активного вікна браузера. Якщо вікно існує, передається URL сторінки разом із затримкою кліків до методу enable_settings об'єкта ActionRecorder. У випадку відсутності активного вікна користувач отримує відповідне повідомлення. Подібним чином, маршрут /reset-experiment скидає налаштування за допомогою методу reset_settings класу ActionRecorder, але також перевіряє наявність активного вікна перед виконанням дії. Фрагмент коду обробника enable_settings наведено у лістингу 3.3.

Лістинг 3.3 – Фрагмент коду обробника enable_settings

```
@app.route("/enable-settings", methods=["POST"])
def enable_settings():
```

```
    """Обробник увімкнення налаштувань."""
    global browser, action_recorder
    data = request.get_json()
    localization = data.get("localization", "ua")
    current_window = browser.get_current_window()
    if current_window:
        browser.switch_to_window(current_window)
    action_recorder.enable_settings(browser.driver.current_url,
config["default_click_delay"])
        if localization == "ua":
            return jsonify({"message": "Налаштування
увімкнено."})
        else:
            return jsonify({"message": "Settings enabled."})
    else:
        browser.handle_window_closure()
        if localization == "ua":
            return jsonify({"message": "Немає активного вікна
для роботи."})
        else:
            return jsonify({"message": "No active window to work
with."})
```

Клас ActionRecorder, описаний у файлі actions.py, виконує основну логіку автоматизації. Він працює з координатами елементів на сторінці, що зберігаються в JSON-файлі (data/links.json). Цей клас підтримує два основних методи: enable_settings для увімкнення налаштувань і reset_settings для скидання. Обидва методи використовують допоміжні функції, такі як click_coordinates, яка симулює натискання на екран за вказаними координатами із заданою затримкою.

Ключова функція класу — find_site_data — шукає дані про сайт у конфігураційному файлі за URL. Це дозволяє адаптувати автоматизацію для різних сайтів, що зберігаються в структурі JSON. Метод enable_settings наведено у лістингу 3.4.

Лістинг 3.4 – Код методу enable_settings

```
def enable_settings(self, url, delay):
    """
    Включає налаштування на сайті.
    :param url: URL поточної сторінки.
```

```
    :param delay: Затримка між кліками.
    """
    site_data = self.find_site_data(url)
    if site_data:
        print(f"Enabling settings for site:
{site_data['name']}")
  self.click_coordinates(site_data.get("settings_button_coords",
[]), delay)
    else:
        print(f"No settings button coordinates found for URL:
{url}")
```

Клас Browser, реалізований у файлі browser.py, є обгорткою для роботи з браузером на основі undetected_chromedriver. Він підтримує такі функції, як відкриття сторінок, переключення між вікнами, збереження скриншотів, і обробка закриття активних вікон. Цей клас тісно інтегрований із ActionRecorder, оскільки забезпечує доступ до поточного URL і управління станом браузера.

У графічному інтерфейсі реалізовані функції для надсилання запитів до Flask-сервера. Наприклад, метод enable_settings у класі FormWindow виконує HTTP-запит до маршруту /enable-settings і відображає результат у текстовому полі. Аналогічно працює функція reset_experiment, яка звертається до /reset-experiment. Код інтеграції наведено у лістингу 3.5.

Лістинг 3.5 – Фрагмент коду інтеграції

```
def enable_settings(self):
    """Увімкнути налаштування."""
    try:
        response = requests.post(
            "http://127.0.0.1:5000/enable-settings",
            json={"localization": self.localization},
        )
        data = response.json()
        self.output.setText(data.get("message", "Помилка."))
    except Exception as e:
        self.output.setText(f"Помилка: {str(e)}")
```

Модуль автоматизації дій у веб-браузері забезпечує плавну інтеграцію між автоматизованими діями, серверною логікою та графічним інтерфейсом.

Завдяки гнучкій архітектурі, він дозволяє адаптуватися до змін у сайтах і забезпечує стабільну роботу через ретельно реалізовані класи і методи.

3.3.2 Модуль для обробки зображень і сегментації

Модуль обробки зображень і сегментації є одним із ключових компонентів системи. Його основна функція — автоматизована обробка скриншотів із використанням глибокого навчання для виділення потрібних областей (сегментів) на зображеннях. Реалізація модуля включає створення нейронної мережі, її навчання на спеціально підготовлених даних і використання для обробки нових зображень.

Стосовно архітектури моделі, у файлі model.py визначено структуру згорткової нейронної мережі SimpleCNN, яка виконує завдання регресії для прогнозування координат об’єктів на зображеннях. Код зображено на лістингу 3.6.

Лістинг 3.6 – Код класу SimpleCNN

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1,
padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1,
padding=1)
        self.fc1 = nn.Linear(32 * 32 * 32, 128)
        self.fc2 = nn.Linear(128, 4)  # Пророцтво x1, y1,
ширина, висота

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = x.view(x.size(0), -1)  # Flatten
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Ця модель складається з двох згорткових шарів (conv1 і conv2) із функцією активації ReLU, пулінгових шарів для зменшення розміру зображень, і двох повнозв'язних шарів (fc1 і fc2), які виконують кінцеве прогнозування координат та розмірів виділеної області.

Модуль завантаження даних для навчання реалізовано у файлі dataset.py, який виконує ключову функцію підготовки вхідних даних. В цьому файлі визначено клас ScreenshotDataset, що відповідає за завантаження зображень із зазначеної директорії та асоційованих з ними координат прямокутників, які зберігаються в назвах файлів. Ця структура забезпечує точне співвідношення між зображенням і мітками, необхідними для навчання моделі.

Дані організовані у форматі, де кожне зображення має ім'я, що включає координати x1, y1, ширину w і висоту h прямокутника, який потрібно виділити. Клас ScreenshotDataset зчитує ці дані та перетворює їх у формат, зрозумілий для нейронної мережі.

Основні функції класу:

– ініціалізація: у методі __init__ задається шлях до папки зі скриншотами та параметри трансформації. Список файлів створюється шляхом фільтрації всіх .png файлів у вказаній директорії;

– розбір імен файлів: метод parse_filename розділяє ім'я файлу, отримуючи числові значення координат x1, y1, ширини w і висоти h. Це дозволяє використовувати ці координати як вихідні значення для навчання;

– завантаження зображень: метод __getitem__ забезпечує доступ до окремих зображень за індексом. Він відкриває файл, перетворює його у формат RGB, застосовує вказані трансформації (за наявності) та повертає пару: зображення і тензор координат;

– обчислення розміру набору даних: метод __len__ повертає кількість зображень у наборі, що дозволяє використовувати цей клас у рамках стандартного пайплайна PyTorch.

Для зменшення розмірів вхідних зображень і покращення ефективності

обробки використовується спеціально створена трансформація, яка масштабує зображення до розміру 128×128 пікселів та перетворює їх у формат тензорів, що є стандартом для PyTorch. Зазначений код трансформації дозволяє нейронній мережі працювати із стандартизованими даними, зменшуючи варіативність у розмірах вхідних даних і спрощуючи процес обчислень. Код реалізації ScreenshotDataset зображено у лістингу 3.7.

Лістинг 3.7 – Код відповідний за підготовку даних

```
import os
from PIL import Image
import torch
from torch.utils.data import Dataset
from torchvision import transforms
class ScreenshotDataset(Dataset):
    def __init__(self, screenshots_dir, transform=None):
        self.screenshots_dir = screenshots_dir
        self.image_files = [
            f for f in os.listdir(screenshots_dir) if
f.endswith('.png')
        ]
        self.transform = transform
    def parse_filename(self, filename):
        _, x1, y1, w, h = filename.replace('.png',
'').split('_')
        return int(x1), int(y1), int(w), int(h)
    def __len__(self):
        return len(self.image_files)
    def __getitem__(self, idx):
        file_path = os.path.join(self.screenshots_dir,
self.image_files[idx])
        image = Image.open(file_path).convert("RGB")
        x1, y1, w, h =
self.parse_filename(self.image_files[idx])
        if self.transform:
            image = self.transform(image)
        return image, torch.tensor([x1, y1, w, h],
dtype=torch.float32)
```

Важливими аспектами реалізації ScreenshotDataset є модульна структура, яка дозволяє легко змінювати параметри завантаження даних, додавати нові трансформації чи налаштовувати обробку конкретних форматів зображень. Ця гнучкість забезпечує сумісність із різними

сценаріями використання моделі, дозволяючи змінювати дані для навчання в залежності від завдань системи.

Процес навчання моделі реалізовано у функції train_model, яка знаходиться в головному файлі main.py. Ця функція відповідає за підготовку даних, ініціалізацію моделі, її навчання та збереження у форматі .pth. Якщо розглядати цей процес детальніше, то на першому етапі завантажується набір даних через клас ScreenshotDataset, який описано у модулі dataset.py. Використовується підготовлена трансформація transform, що масштабує зображення та перетворює їх у тензори. Потім дані передаються до об'єкта DataLoader, який розділяє їх на батчі, забезпечує перемішування даних і готує їх до обробки моделлю.

Модель створюється за допомогою класу SimpleCNN, визначеного у файлі model.py. Ця модель є простою згортковою нейронною мережею, що передбачає координати прямокутника для обрізки зображення. Перед початком навчання модель переводиться на доступний пристрій обчислення (GPU або CPU), що визначається автоматично через torch.device.

У випадку, якщо вже існує попередньо навчена модель, її параметри завантажуються із файлу last_model.pth. Це забезпечує можливість продовження навчання з останньої збереженої точки, що зручно для великих наборів даних чи тривалих навчальних сесій.

Для навчання використовується функція втрат MSELoss (середньоквадратична похибка), яка підходить для задачі регресії. Оптимізація параметрів моделі здійснюється методом Adam, що забезпечує швидку та стабільну конвергенцію.

Навчання відбувається у кілька епох, кількість яких задається параметром epochs. На кожній епосі дані проходять через модель батчами, які обробляються у циклі. В кожній ітерації:

– дані переміщуються на пристрій обчислення (GPU або CPU);

– виконується прямий прохід через модель для отримання прогнозів;

– обчислюється функція втрат між прогнозами та справжніми мітками;

– градієнти обчислюються методом зворотного проходу;

– параметри моделі оновлюються оптимізатором.

Після завершення кожної епохи обчислюється середнє значення втрат, яке використовується для моніторингу прогресу навчання. Код даної описаної функції зображено у лістингу 3.8.

Лістинг 3.8 – Код функції train_model

```
def train_model(epochs=10, batch_size=8, learning_rate=0.001):
    dataset = ScreenshotDataset(data_dir, transform=transform)
    dataloader = DataLoader(dataset, batch_size=batch_size,
shuffle=True)
    model = SimpleCNN()
    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
    model.to(device)
    last_model_path = os.path.join(model_dir, "last_model.pth")
    if os.path.exists(last_model_path):
        model.load_state_dict(torch.load(last_model_path,
map_location=device))
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        for images, targets in dataloader:
            images, targets = images.to(device),
targets.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        avg_loss = running_loss / len(dataloader)
        print(f"Эпоха {epoch + 1}/{epochs}, Потеря:
{avg_loss:.4f}")
    torch.save(model.state_dict(),last_model_path,
_use_new_zipfile_serialization=False)
```

Після завершення навчання модель зберігається у файл last_model.pth. Це забезпечує можливість повторного використання чи відновлення моделі без необхідності повторного навчання. Збереження здійснюється

стандартним методом torch.save. Для обробки зображень за допомогою вже навченої моделі використовується функція process_with_model. Вона виконує такі кроки:

– завантаження моделі: збережена модель завантажується з файлу, переданого як аргумент model_path, і переводиться в режим оцінювання (eval), що дозволяє уникнути обчислення градієнтів під час прогнозування;

– обробка вхідного зображення: вхідне зображення відкривається за допомогою бібліотеки Pillow, перетворюється у формат RGB і піддається трансформації transform. Перетворене зображення додається у вигляді нового виміру, щоб відповідати очікуваному формату моделі;

– прогнозування координат: модель передбачає координати прямокутника на зображенні, що визначають обрізку. Ці координати (x1, y1, ширина, висота) масштабуються та округлюються до цілих значень;

– обрізка та збереження результату: обрізане зображення зберігається в папці, вказаній параметром output_dir. Ім'я результату формується на основі імені вхідного файлу.

Функція process_with_model дозволяє використовувати модель для автоматичної обробки зображень, виділяючи необхідні області з високою точністю. Цей підхід інтегрується в загальну систему, забезпечуючи автоматизацію сегментації зображень. Код якої представлено у лістингу 3.9.

Лістинг 3.9 – Код функції process_with_model

```
def process_with_model(model_path, input_image,
output_dir=complete_dir):
     model = SimpleCNN()
     model.load_state_dict(torch.load(model_path,
map_location=torch.device('cpu')))
     model.eval()
     image = Image.open(input_image).convert("RGB")
     transform_image = transform(image).unsqueeze(0)
     with torch.no_grad():
          output = model(transform_image).squeeze().numpy()
          x1, y1, w, h = map(int, output)
     cropped_image = image.crop((x1, y1, x1 + w, y1 + h))
```

```
output_filename = f"cropped_{os.path.basename(input_image)}"
output_path = os.path.join(output_dir, output_filename)
cropped_image.save(output_path)
```

Цей модуль використовується для обробки зображень у процесі створення звітів. Задля розділення обов'язків навчання моделі виконується в окремому робочому файлі (наприклад, train_model.py), що спрощує підтримку й оновлення системи.

3.3.3 Модуль для генерації звітів

Генерація звітів реалізована через спеціалізований клас ReportHandler, який знаходиться у файлі reporthandler.py. Цей клас відповідає за створення звітів у форматі Word-документів, додає до них текстові описи, зображення, видаляє зайві етапи, зберігає результати з унікальними назвами файлів і забезпечує можливість очищення документів. У головному файлі main.py методи цього класу інтегровані в графічний інтерфейс програми, дозволяючи користувачу взаємодіяти з функціоналом через кнопки. Давайте детально розглянемо, як це працює, із посиланнями на код та поясненням його логіки.

Коли створюється об'єкт класу ReportHandler, він ініціалізує новий порожній документ Word через бібліотеку python-docx. Шлях до файлу звіту задається через параметр file_path, та в цьому випадку все буде зберігатися у папці reports, а локалізація тексту визначається через параметр localization. Це дозволяє програмі створювати звіти як українською, так і англійською мовами. Ініціалізація також включає встановлення лічильника етапів (stage_counter), який використовується для нумерації доданих частин звіту. Код, який відповідає за це, розміщено у лістингу 3.10.

Лістинг 3.10 – Код ініціалізації класу ReportHandler

```
def __init__(self, file_path="reports/report.docx",
localization="ua"):
```

```
    self.file_path = file_path
    self.localization = localization
    self.document = Document()
    self.stage_counter = 0
```

Ключова роль цього конструктора полягає в підготовці об’єкта до роботи, забезпечуючи структуру для додавання даних.

Метод add_stage дозволяє додавати текстовий опис етапу і, за потреби, зображення. Коли викликається цей метод, збільшується лічильник етапів, і до документа додається заголовок, що залежить від вибраної локалізації. Наприклад, якщо локалізація встановлена як ua, заголовок буде виглядати як "Етап 1", "Етап 2" тощо. Після цього додається текст етапу, який передається як аргумент stage_text.

Якщо разом із текстом передано шлях до скріншота, програма перевіряє, чи існує файл за вказаним шляхом. Якщо файл знайдено, зображення додається до документа. В іншому випадку виводиться попередження про відсутність файлу. Реалізацію даного методу наведено у лістингу 3.11.

Лістинг 3.11 – Код методу add_stage

```
def add_stage(self, stage_text, screenshot_path=None):
    self.stage_counter += 1
    stage_title = f"Етап {self.stage_counter}" if
self.localization == "ua" else f"Stage {self.stage_counter}"
    self.document.add_heading(stage_title, level=1)
    self.document.add_paragraph(stage_text)
    if screenshot_path and os.path.exists(screenshot_path):
        self.document.add_picture(screenshot_path,
width=5000000)
    elif screenshot_path:
        warning = f"Скріншот {screenshot_path} не знайдено." if
self.localization == "ua" else f"Screenshot {screenshot_path}
not found."
```

Цей код автоматизує процес формування звіту, додаючи елементи у правильному порядку.

Метод remove_last_stage використовується для видалення останнього доданого етапу, якщо користувач вирішує, що внесені дані більше не потрібні. Процес починається з перевірки наявності хоча б одного етапу в документі. Якщо етапи відсутні, користувач отримує повідомлення про те, що немає чого видаляти.

Якщо етапи є, програма ідентифікує всі елементи документа, що належать до останнього етапу. Ці елементи видаляються зі структури документа. Лічильник етапів зменшується, а користувачу повідомляється про успішне виконання операції. Реалізацію методу зображено у лістингу 3.12.

Лістинг 3.12 – Код методу remove_last_stage

```
def remove_last_stage(self):
    if self.stage_counter == 0:
        print("No stages to remove.")
        return
    body = self.document._element.body
    elements_to_remove = []
    stage_keyword = "Етап" if self.localization == "ua" else
"Stage"
    for child in reversed(body):
        if child.tag in
["{http://schemas.openxmlformats.org/wordprocessingml/2006/main}
p",
"{http://schemas.openxmlformats.org/wordprocessingml/2006/main}t
bl"]:
            elements_to_remove.append(child)
        if child.tag ==
"{http://schemas.openxmlformats.org/wordprocessingml/2006/main}p
" and \
                stage_keyword in (child.text or ""):
            break
    for element in elements_to_remove:
        body.remove(element)
    self.stage_counter -= 1
    print(f"Last stage removed. Remaining stages:
{self.stage_counter}")
```

Метод save_report дозволяє зберегти документ у папці reports. Перед збереженням програма перевіряє, чи містить звіт хоча б один етап. Якщо етапи відсутні, користувач отримує повідомлення про неможливість

збереження. У випадку, коли звіт заповнений, створюється унікальне ім’я файлу з використанням часової мітки, і документ зберігається у форматі .docx. код даної функції зображено у лістингу 3.13.

Лістинг 3.13 – Код методу для збереження файлу

```
def save_report(self):
    if self.stage_counter == 0:
        print("Cannot save: no experiment stages present.")
        return
    reports_dir = "reports"
    if not os.path.exists(reports_dir):
        os.makedirs(reports_dir)
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    file_name = f"report_{timestamp}.docx"
    full_path = os.path.join(reports_dir, file_name)
    self.document.save(full_path)
    print(f"Report saved: {full_path}")
```

У головному модулі main.py функції додавання, видалення, збереження та очищення звіту прив’язані до кнопок графічного інтерфейсу. Наприклад, кнопка "Додати етап" викликає функцію add_stage, яка генерує текст, робить скріншот, обробляє його моделлю і додає результати до звіту.

Аналогічно, кнопки "Видалити етап", "Зберегти звіт" і "Очистити звіт" викликають відповідні методи класу ReportHandler, забезпечуючи зручність у роботі з інтерфейсом.

3.3.4 Модуль інтеграції GPT для роботи з текстом

Модуль інтеграції GPT виконує роль проміжного рівня для взаємодії програми з мовною моделлю OpenAI GPT через API. Ця інтеграція забезпечує можливість відправки запитів до GPT, отримання відповідей, і подальшого відображення цих відповідей у графічному інтерфейсі користувача. У реалізації задіяні три основні компоненти: клас GPTAssistant для роботи з API, функція у Flask-додатку для обробки запитів від клієнта, та

інтеграція цієї функціональності в основному вікні програми з використанням PyQt.

Клас GPTAssistant у файлі gpt_integration.py забезпечує основну логіку взаємодії з GPT. Він приймає API-ключ під час ініціалізації для встановлення зв’язку з OpenAI. Цей клас має метод ask, який приймає текст запиту від користувача та опціонально параметр token, що визначає максимальну кількість токенів у відповіді.

У методі ask формується запит до GPT через клієнт OpenAI. Для цього використовується модель gpt-4o-mini. Вхідні дані передаються у форматі списку повідомлень, де кожне повідомлення містить роль (наприклад, "user") і текст. Отримана відповідь повертається у вигляді тексту, отриманого з першого елемента списку choices. Код класу GPTAssistant наведено у лістингу 3.14.

Лістинг 3.14 – Код класу GPTAssistant

```
class GPTAssistant:
    def __init__(self, api_key):
        self.client = OpenAI(api_key=api_key)
    def ask(self, user_input, token=200):
        completion = self.client.chat.completions.create(
            model="gpt-4o-mini",
            store=True,
            messages=[{"role": "user", "content": user_input}],
            max_tokens=token
        )
        return completion.choices[0].message
```

Основна роль цього класу абстрагувати складнощі роботи з OpenAI API, забезпечуючи простий інтерфейс для виконання запитів. Обробка запитів у Flask-додатку працює наступним чином, а саме Flask-сервер працює як проміжна ланка між клієнтською частиною програми та модулем GPTAssistant. Функція send_gpt приймає POST-запити з текстом, що передається через JSON. Після цього:

– запит отримується із ключа query;

– якщо текст відсутній, сервер повертає помилку зі статусом 400;

– текст передається до методу ask об’єкта GPTAssistant для обробки;

– якщо відповідь отримана успішно, вона повертається у форматі JSON клієнту. Інакше сервер надсилає повідомлення про невдачу.

Фрагмент коду обробника запитів у Flask зображено у лістингу 3.15.

Лістинг 3.15 – Фрагмент коду обробника запитів у Flask

```
@app.route("/send-gpt", methods=["POST"])
def send_gpt():
    """Обработчик запроса к GPT."""
    global gpt_assistant

    try:
        data = request.json
        user_input = data.get("query", "")
        if not user_input:
            return jsonify({"message": "Введіть текст для
запиту."}), 400

        # Обработка через GPT
        response = gpt_assistant.ask(user_input)
        if response:
            return jsonify({"message": response.content})
        else:
            return jsonify({"message": "Не вдалося отримати
відповідь від GPT."}), 500
    except Exception as e:
        print(f"Ошибка обработки запроса: {e}")
        return jsonify({"message": "Виникла помилка на
сервері."}), 500
```

Ця функція забезпечує стабільну взаємодію з GPT навіть у випадку помилок, адже всі виключення обробляються та повертаються відповідні статуси.

У графічному інтерфейсі (файл main.py, клас FormWindow) функція send_gpt_request відповідає за відправлення запиту до Flask-сервера і обробку отриманих відповідей. Коли користувач натискає кнопку, текст із текстового поля зчитується, і виконується HTTP POST-запит до ендпоінту /send-gpt. Сервер повертає JSON із ключем message, який містить відповідь від GPT.

Функція також враховує можливість помилок, таких як відсутність тексту у полі або проблеми з сервером, і відповідно повідомляє користувача. Фрагмент коду для відправки запиту наведено у лістингу 3.16.

Лістинг 3.16 – Фрагмент коду для відправки запиту

```
def send_gpt_request(self):
    """Відправляє запит до GPT."""
    query = self.input_text.text()
    if not query:
        if self.localization == "ua":
            self.output.setText("Будь ласка, введіть текст для
запиту.")
        else:
            self.output.setText("Please enter text for the
query.")
        return
    try:
        # Надсилаємо запит на сервер
        response = requests.post("http://127.0.0.1:5000/send-
gpt", json={"query": query})
        data = response.json()
        message = data.get("message", "Помилка." if
self.localization == "ua" else "Error.")
        # Очищаємо форматування
        sanitized_message = (
            message.replace(r"\(", "")
            .replace(r"\)", "")
            .replace(r"\[", "")
            .replace(r"\]", "")
            .replace(r"\\", "")
            .replace(r"\cdot", "·")
            .strip()
        )
        # Виводимо відповідь
        self.output.setText(sanitized_message)
    except Exception as e:
        self.output.setText(f"Помилка: {str(e)}" if
self.localization == "ua" else f"Error: {str(e)}")
```

Цей код забезпечує зручний інтерфейс для користувача: запит надсилається автоматично після натискання кнопки, а відповідь виводиться у відформатованому вигляді.

Інтеграція GPT виконується через модуль GPTAssistant, який абстрагує

API-запити. Flask-додаток забезпечує посередництво між клієнтською частиною і GPT, а функція в PyQt інтегрує цей функціонал у графічний інтерфейс. Така архітектура дозволяє розділити логіку програми на чіткі компоненти, забезпечуючи масштабованість і гнучкість у роботі.

3.3.5 Модуль інтерфейсу користувача

Модуль інтерфейсу користувача забезпечує зручну взаємодію користувача з додатком через графічний інтерфейс (GUI) та веб-інтерфейс. Цей модуль об'єднує дві ключові складові: десктопне GUI-додаток, реалізоване на основі PyQt5, і фронтенд для веб-додатку з використанням HTML, CSS, JavaScript.

Інтерфейс користувача програми, розроблений за допомогою бібліотеки PyQt5, складається з головного вікна, яке забезпечує функціональність взаємодії користувача із системою. Нижче наведено детальний опис архітектури інтерфейсу, логіки його роботи та його інтеграції з серверною частиною.

Головне вікно програми представлено класом FormWindow, який є підкласом QMainWindow. Воно виконує роль центрального елемента управління, через який користувач взаємодіє з функціоналом програми. Код створення головного вікна наведено у лістингу 3.17.

Лістинг 3.17 – Фрагмент коду створення головного вікна

```
class FormWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.localization = "ua"  # Поточна локалізація
        self.init_ui()  # Ініціалізація графічного інтерфейсу
        self.report_handler = ReportHandler()
        self.screenshots_dir = "data/screenshots"
        os.makedirs(self.screenshots_dir, exist_ok=True)
```

Метод init_ui(), що наведено у лістингу 3.18, відповідає за створення та

налаштування усіх елементів графічного інтерфейсу. У головному вікні присутні текстові поля, кнопки, а також спеціальна кнопка для перемикання локалізації (між українською та англійською мовами).

Лістинг 3.18 – Фрагмент коду методу init_ui

```
def init_ui(self):
    self.setWindowFlags(Qt.Window | Qt.WindowStaysOnTopHint)
    self.setGeometry(1600, 450, 300, 350)
    self.setWindowTitle("Assistant")
    layout = QVBoxLayout()
    self.output = QTextEdit(self)
    self.output.setReadOnly(True)
    self.input_text = QLineEdit(self)
    self.input_text.setPlaceholderText("Введіть текст")
    layout.addWidget(self.output)
    layout.addWidget(self.input_text)
```

Два текстових поля виконують різні функції. Поле output призначене для виведення відповідей системи або повідомлень про помилки, а поле input_text дозволяє вводити запити користувача.

Інтерфейс містить декілька груп кнопок, кожна з яких відповідає за певну функцію. Наприклад, кнопки для надсилання запиту до GPT, очищення вводу, додавання етапів до звіту та інші.

Кожна кнопка підключена до відповідного методу, який обробляє події користувача. Наприклад, кнопка "Запит у GPT" викликає метод send_gpt_request, що надсилає текстовий запит до сервера. Інтерфейс підтримує локалізацію для двох мов: української та англійської. Перемикання локалізації відбувається за допомогою кнопки внизу інтерфейсу. Текст кнопок оновлюється динамічно.

Графічний інтерфейс взаємодіє з сервером Flask для виконання запитів. Наприклад, метод send_gpt_request надсилає текстовий запит на /send-gpt і виводить отриману відповідь у текстове поле output, код наведено у лістингу 3.19.

Лістинг 3.19 – Фрагмент коду створення кнопок

```
def send_gpt_request(self):
    query = self.input_text.text()
    if not query:
        self.output.setText("Введіть текст для запиту.")
        return
    try:
        response   =   requests.post("http://127.0.0.1:5000/send-
gpt", json={"query": query})
        data = response.json()
        sanitized_message = (
            data.get("message", "Помилка.")
            .replace(r"\(", "").replace(r"\)", "")
            .replace(r"\[", "").replace(r"\]", "")
            .replace(r"\\", "").replace(r"\cdot", "·").strip())
        self.output.setText(sanitized_message)
    except Exception as e:
        self.output.setText(f"Помилка: {str(e)}")
```

Цей метод включає обробку помилок і очищення тексту відповіді від небажаних символів.

Графічний інтерфейс на основі PyQt5 — це добре структуроване рішення, яке дозволяє інтегрувати багатофункціональний клієнтський додаток з серверною частиною та моделями машинного навчання. Він підтримує локалізацію, взаємодію з сервером через API, а також обробку користувацьких запитів та зображень.

3.4 Локалізація програмного забезпечення

Реалізація локалізації програмного забезпечення в даному проєкті дозволяє забезпечити зручність використання програми користувачами, які розмовляють різними мовами. У нашій програмі підтримується дві основні локалізації: українська (UA) та англійська (EN). Перемикання локалізації реалізовано через інтерактивний інтерфейс, що надає можливість користувачеві змінювати мову без необхідності перезапуску програми. Цей процес супроводжується динамічним оновленням тексту всіх кнопок і

текстових полів.

Основним елементом, що відповідає за локалізацію, є кнопка self.localization_button. Її натискання змінює значення атрибута self.localization, після чого викликається метод toggle_localization. На початку роботи програми інтерфейс встановлено на українську мову. Початковий вигляд програми до перемикання локалізації можна побачити на рисунку 3.2. На ньому видно, що всі кнопки мають підписи українською мовою.

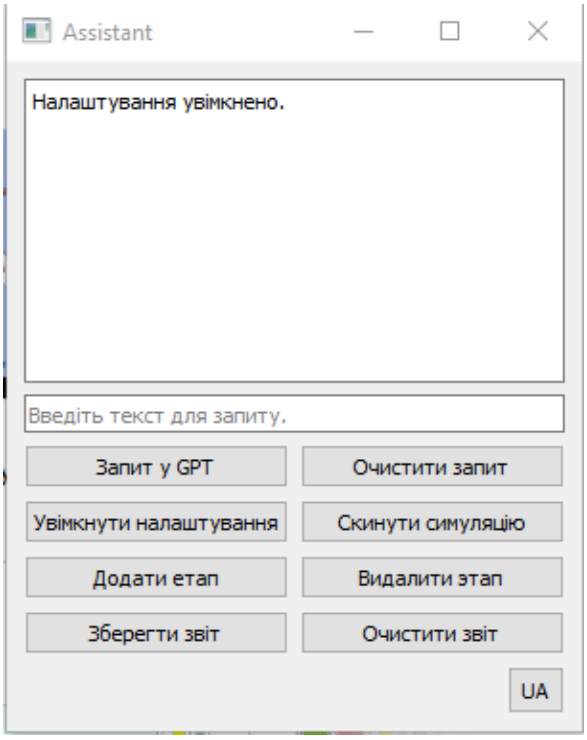

Рисунок 3.2 – Вигляд додатку українською мовою

Метод toggle_localization відповідає за зміну тексту на кнопках та в інших елементах інтерфейсу залежно від вибраної мови, по замовчуванням на українській. Наприклад, при зміні мови на англійську текст кнопок оновлюється за допомогою методу update_button_texts, а текстове поле вводу змінюється на "Enter your query". Код, який реалізує зміну локалізації, наведено у лістингу 3.20.

Лістинг 3.20 – Фрагмент коду створення кнопок

```
def toggle_localization(self):
    if self.localization == "ua":
        self.localization = "en"
        self.localization_button.setText("EN")
        self.update_button_texts([
            "Request to GPT", "Clear request",
            "Enable settings", "Reset experiment",
            "Add stage", "Remove stage",
            "Save report", "Clear report",
        ])
        self.input_text.setPlaceholderText("Enter your query.")
        self.output.setText("")
```

Результат після перемикання локалізації на англійську мову можна побачити на рисунку 3.3. Тут видно, що всі кнопки перекладені на англійську, а текстова підказка у полі вводу також відображається відповідною мовою.

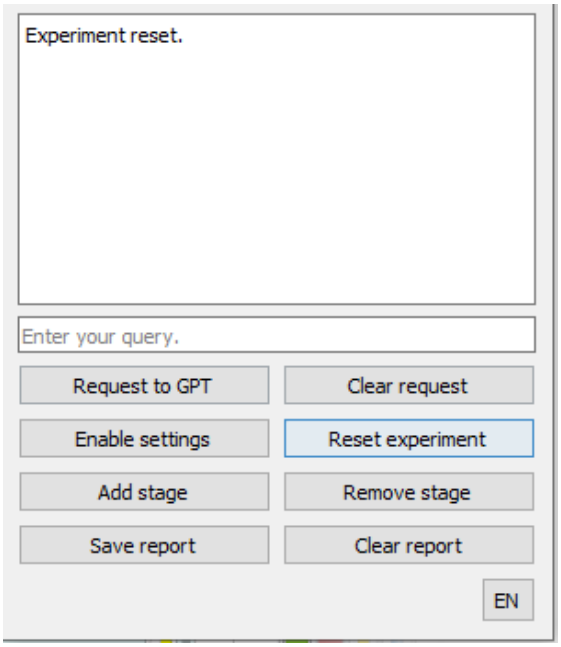

Рисунок 3.3 – Вигляд додатку на англійській мові

Локалізація врахована не лише для тексту інтерфейсу, а й для повідомлень, що відображаються користувачеві. Наприклад, якщо

користувач не введе текст у поле вводу та спробує надіслати запит, у полі output з'явиться повідомлення "Будь ласка, введіть текст для запиту." для української мови або "Please enter text for the query." для англійської. Це забезпечує інтерактивність і гнучкість роботи програми.

Таким чином, реалізація локалізації дозволяє користувачам легко взаємодіяти з програмою на вибраній мові, а динамічне оновлення інтерфейсу гарантує зручність використання без потреби перезапуску. Візуалізація інтерфейсу до та після перемикання локалізації (рисунки 3.2 та 3.3) демонструє, як текст змінюється в реальному часі, підтверджуючи коректну роботу локалізації.

3.5 Тестування системи та аналіз результатів

У цьому пункті представлено результати тестування помічника для роботи з інтерактивними комп’ютерними моделями. Основною метою тестування було перевірити функціональність, стабільність, продуктивність, а також оцінити користувацький досвід. Процес тестування був структурованим і включав планування, підготовку тестового середовища, виконання тестів, аналіз виявлених дефектів та повторне тестування після їх виправлення. Процес тестування мав такі етапи:

– перевірити функціональність системи, щоб забезпечити відповідність розроблених функцій до вимог і гарантувати коректну роботу основних можливостей програми;

– оцінити інтерфейс користувача (UI) за критеріями зручності, швидкості реакції, відповідності принципам дизайну та логічності;

– провести тестування зручності використання для виявлення можливих складнощів у роботі з додатком та перевірити його інтуїтивність.

Функціональне тестування було спрямоване на перевірку всіх ключових функцій програми. Було проведено тестування:

– реалізації запитів до GPT, перевірка відповідності введеному тексту;

– динамічного додавання та видалення етапів експерименту, забезпечення правильного оновлення інтерфейсу;

– збереження та очищення звітів.

Тестування функції надсилання запиту передбачало сценарій введення коректного тексту запиту та перевірку результату у полі виводу. Також були перевірені обробка помилкових сценаріїв, таких як відправка пустого запиту, коли очікувалося повідомлення про необхідність введення тексту.

Було протестовано правильність розташування елементів UI, швидкість реагування кнопок, логіку навігації та відповідність дизайну. Наприклад, перевірено, що при натисканні кнопки "Скинути симуляцію" прогрес очищується та сторінка повертається до початкового виду, а інші елементи інтерфейсу залишаються незмінними. На рисунку 3.4 зображено початковий стан інтерфейсу перед очищенням, а на рисунку 3.5 – після натискання відповідної кнопки.

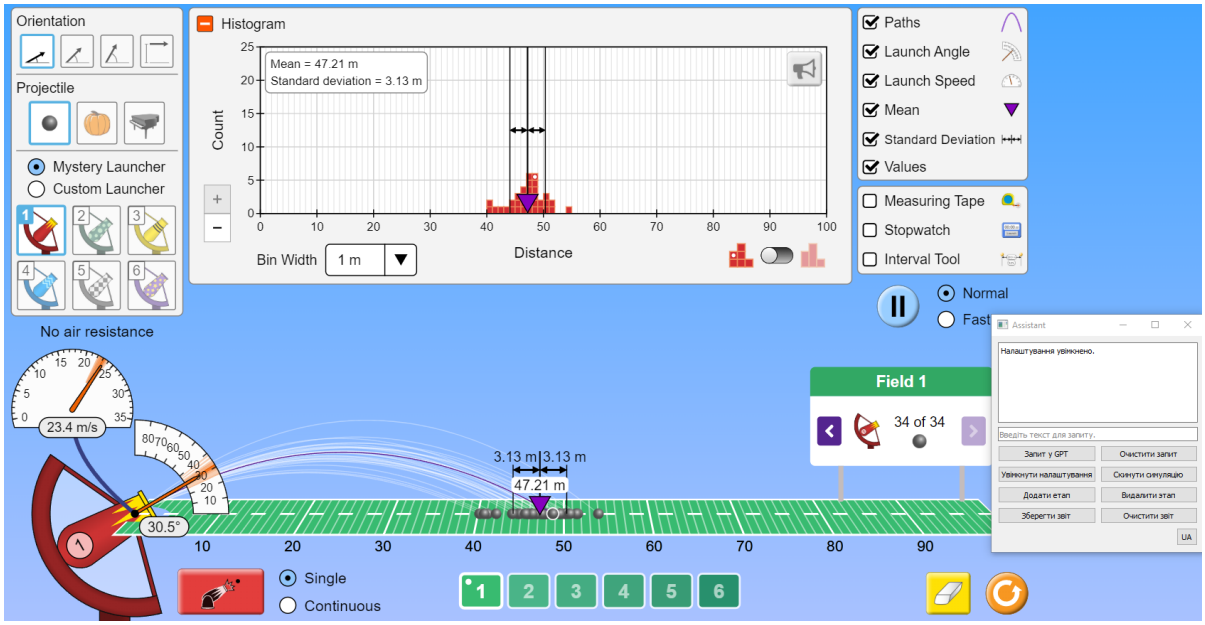

Рисунок 3.4 – Вигляд симуляції під час проведення експерименту

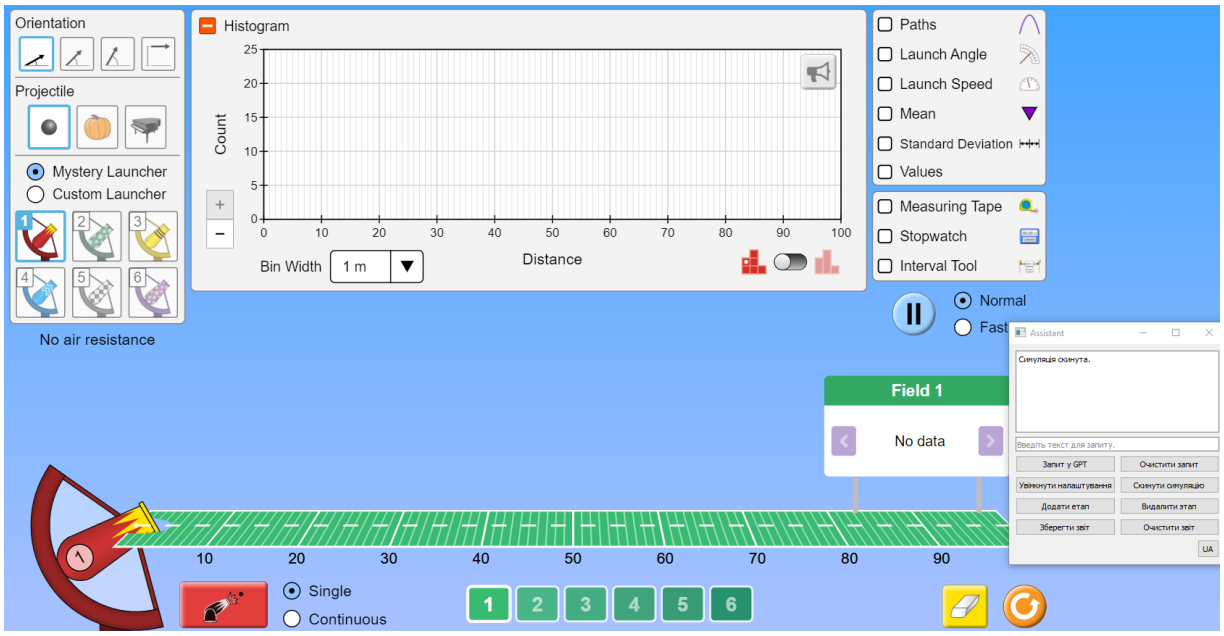

Рисунок 3.5 – Вигляд симуляції після скидання

Для проведення тестування були розроблені тест-кейси, які охоплювали основні сценарії використання програми. На таблицях 3.1 – 3.3 кілька прикладів тест-кейсів, виконаних під час тестування.

Таблиця 3.1 – Тест-кейс Ф1: Перевірка запиту до GPT

Назва тесту:	Ф1: Перевірка запиту до GPT	
Функція:	Введення тексту та надсилання запиту до GPT	
Дія	Очікуваний результат	
Передумова:		
Відкрити та запустити додаток	Додаток відкритий, користувач має до нього доступ	Пройдено
Перейти до вводу запиту	Поле вводу має бути активним	Пройдено
Кроки тесту:		
Ввести текст у поле запиту	Запит користувач відображено та готовий до обробки	Пройдено
Натиснути кнопку "Запит у GPT"	Текст відповіді GPT у полі виводу успішно відображено	Пройдено
Постумова:		
Запустити додаток	Запуск успішний	Пройдено

Перейти до вводу запиту та натиснути кнопку запит	Текст запиту пустий, відображається повідомлення про необхідність введення тексту запиту	Пройдено
Результати тесту		
Тестувальник: Шатан В.В.	Дата прогону тесту: 30.11.2024	Результат тесту (P/F/B): ПРОЙДЕНО (P)
Примітка:		

Таблиця 3.2 – Тест-кейс Ф2: Локалізація інтерфейсу

Назва тесту:	Ф2: Локалізація інтерфейсу	
Функція:	Перемикання локалізації програми	
Дія	Очікуваний результат	
Передумова:		
Відкрити додаток	Додаток відкритий та користувач має доступ	Пройдено
Перевірити локалізацію кнопок	Усі кнопки та надписи на українській мові	Пройдено
Кроки тесту:		
Натиснути кнопку зміни мови	Додаток відобразив результати зміни	Пройдено
Перевірити текст кнопок та підказок у полі вводу	Усі результати та компоненти відображені на відповідній англійській мові	Пройдено
Створити звіт	Звіт повністю на англійській мові, усі компоненти називаються вірно	Пройдено
Постумова:		
Відкрити додаток	Додаток відкритий, користувач має до нього доступ	Пройдено
Створити запит у GPT	Відповідь відображена на необхідній мові	Пройдено
Натиснути кнопку «Очистити запит»	Усі поля очищені, запит та відповідь успішно видалені	Пройдено
Результати тесту		
Тестувальник: Шатан В.В.	Дата прогону тесту: 02.12.2024	Результат тесту (P/F/B): ПРОЙДЕНО (P)
Примітка:		

Таблиця 3.3 – Тест-кейс Ф3: Робота з звітом

Назва тесту:	Ф3: Робота з звітом	
Функція:	Перевірка розділу створення звітів	
Дія	Очікуваний результат	
Передумова:		
Відкрити додаток	Додаток відкритий та користувач має доступ	Пройдено
Перейти до розділу створення звіту	Розділ готовий до роботи	Пройдено
Кроки тесту:		
Ввести дані декілька разів та зберегти результати симуляції до звіту	Відображені повідомлення про успішне додавання кожного етапу експерименту	Пройдено
Натиснути кнопку для видалення етапу	Користувача повідомлено про успішне видалення останнього етапу експерименту	Пройдено
Зберегти звіт	Успішно збережено останній сформований звіт	Пройдено
Перевірити звіт	Усі компоненти звіту відображені вірно та на потрібній мові	Пройдено
Постумова:		
Відкрити додаток	Додаток відкритий, користувач має до нього доступ	Пройдено
Додати 1 етап експерименту	Етап успішно додано	Пройдено
Натиснути кнопку «Очистити звіт»	Усі поля звіту очищено, та звіт готовий до заповнення спочатку	Пройдено
Результати тесту		
Тестувальник: Шатан В.В.	Дата прогону тесту: 04.12.2024	Результат тесту (P/F/B): ПРОЙДЕНО (P)
Примітка:		

Результати тестування підтвердили, що помічник відповідає вимогам і демонструє стабільну роботу. Всі функції працюють коректно, інтерфейс є зручним, а реакція програми швидка. Усі виявлені недоліки були виправлені, і повторне тестування підтвердило їхнє усунення.

3.6 Демонстрація функціоналу системи

У цьому пункті продемонстровано основні можливості програмного забезпечення інтелектуального агента, розробленого для допомоги в комп’ютерних інтерактивних моделях. Головна мета демонстрації наочно показати, як система виконує свої функції, імітуючи дії користувача, обробляючи запити, а також створюючи та керуючи звітом.

Після запуску програми користувач потрапляє на головний екран, який є точкою входу до всіх функцій агента. Інтерфейс представлено у вигляді простого меню, що дозволяє легко здійснювати навігацію між основними розділами. На рисунку 3.6 зображено початковий стан програми після запуску, де видно основні елементи: поле для введення запитів, панель для роботи зі звітом і налаштування симуляції.

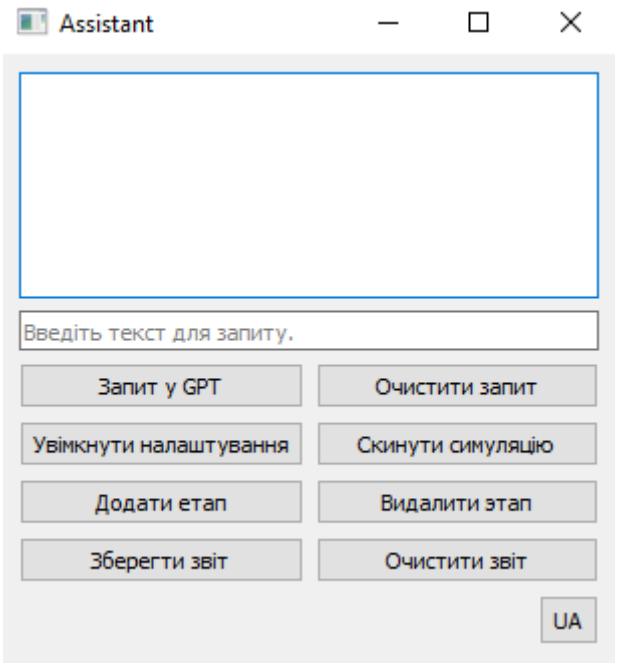

Рисунок 3.6 – Початковий стан помічника

Демонстрація починається з основної функції обробки запитів. Користувач вводить текстовий запит у відповідне поле, після чого програма передає його до моделі GPT. Наприклад, як показано на рисунку 3.7, було

введено запит для пояснення дії закону Гука. Результат відображається одразу в полі виводу у вигляді відповіді від агента.

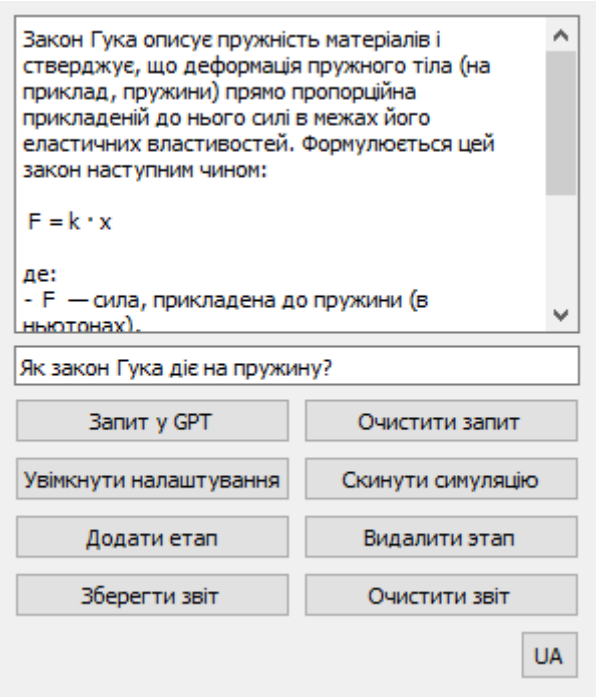

Рисунок 3.7 – Запит до помічника

Наступним аспектом є можливість очищення історії запитів. Після обробки кількох запитів користувач може натиснути кнопку "Очистити запит", що дозволяє видалити всі попередні введення та повернути програму до початкового стану, як було зображено на рисунку 3.6.

Додатково продемонстровано функціонал налаштувань симуляції. Користувач може налаштовувати усі необхідні параметри симуляції , одним натискання кнопки. На рисунку 3.8 показано процес виконання налаштувань.

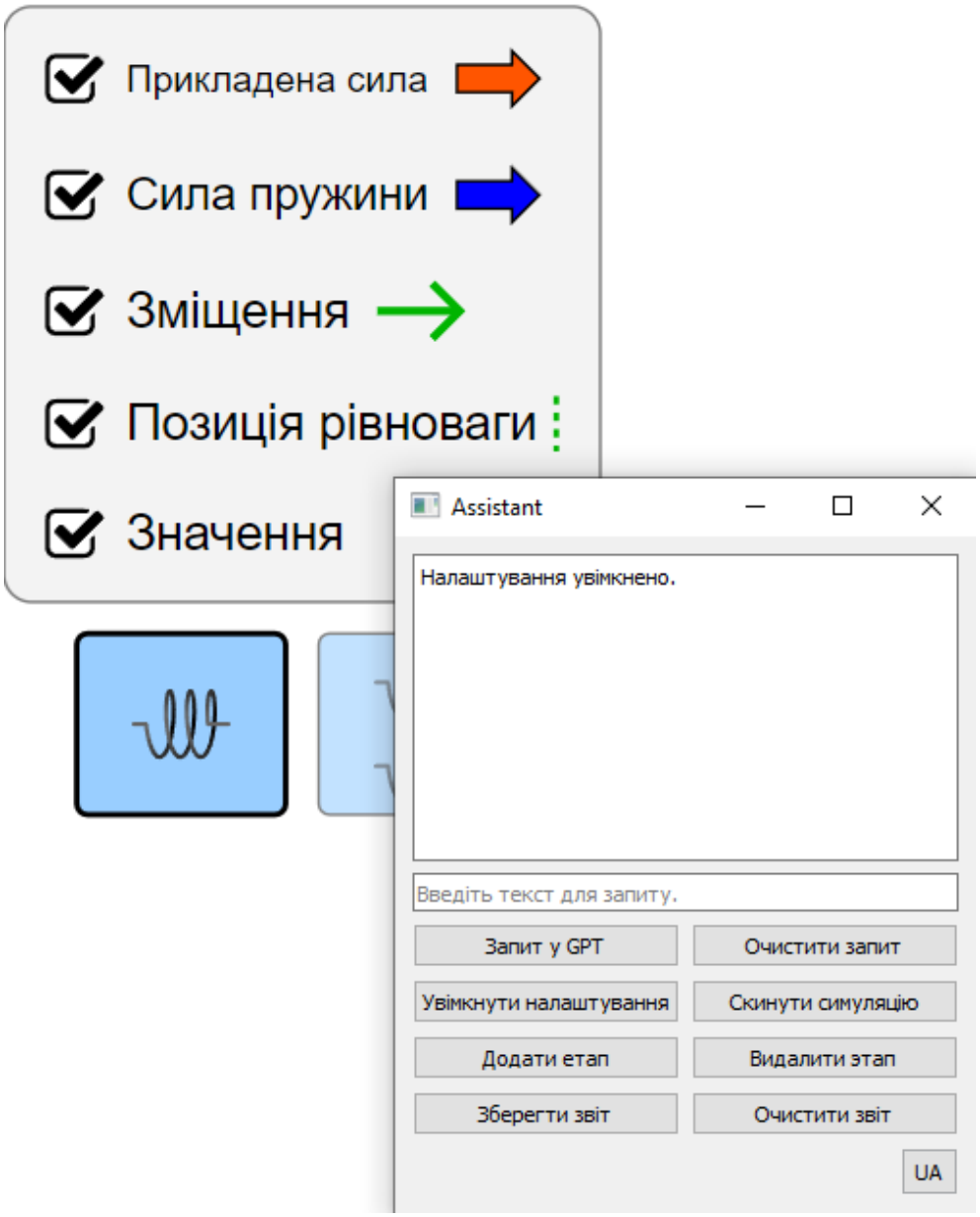

Рисунок 3.8 – Увімкнення налаштувань симуляції

Важливим елементом системи є робота зі звітом. Користувач може створити новий звіт, додавати до нього етапи, видаляти зайві або редагувати існуючі. Як видно на рисунку 3.9, користувач створює новий етап та додає його до звіту. На рисунку 3.10 продемонстровано можливість збереження звіту у файл, що дає змогу відновити його в подальшому.

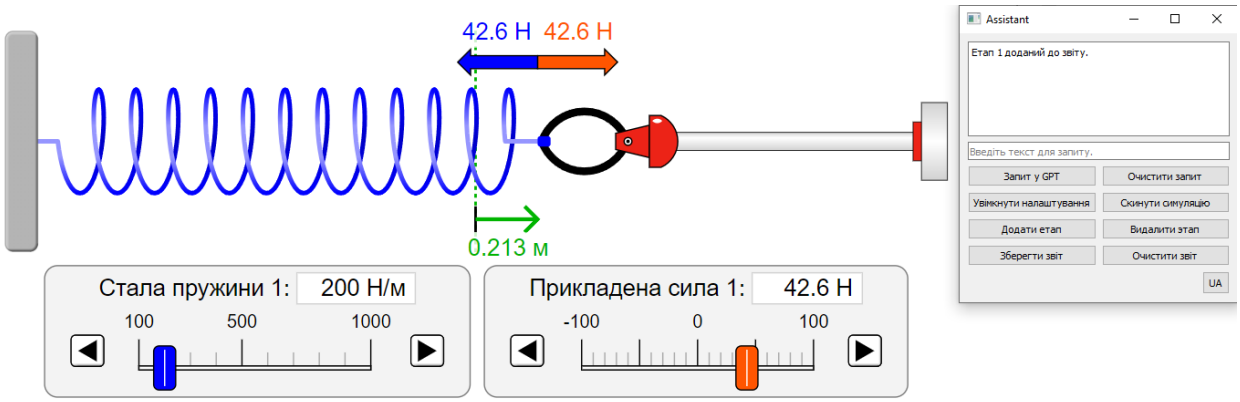

Рисунок 3.9 – Додавання етапу до звіту

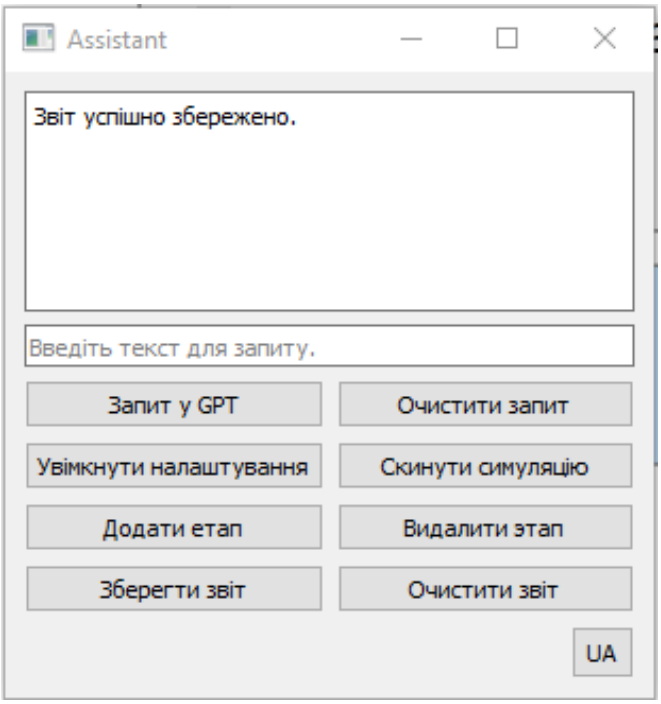

Рисунок 3.10 – Збереження звіту

На рисунку 3.11 продемонстровано згенерований звіт із зазначенням всіх ключових етапів роботи програми. Доступний для подальшої роботи та вдосконалення користувачем.

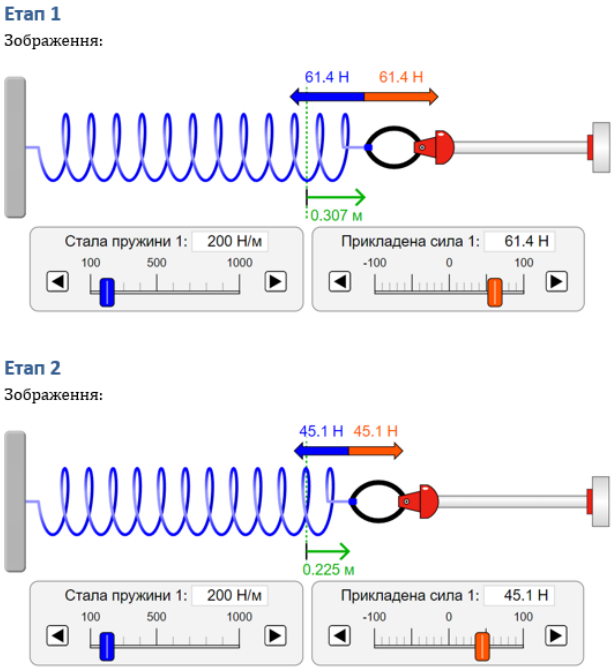

Рисунок 3.11 – Загальний вигляд звіту

В кінці демонстрації було перевірено функцію повного очищення симуляції, яка видаляє всі введення, звіти та налаштування, повертаючи програму до початкового стану. Цей процес показано на рисунку 3.12.

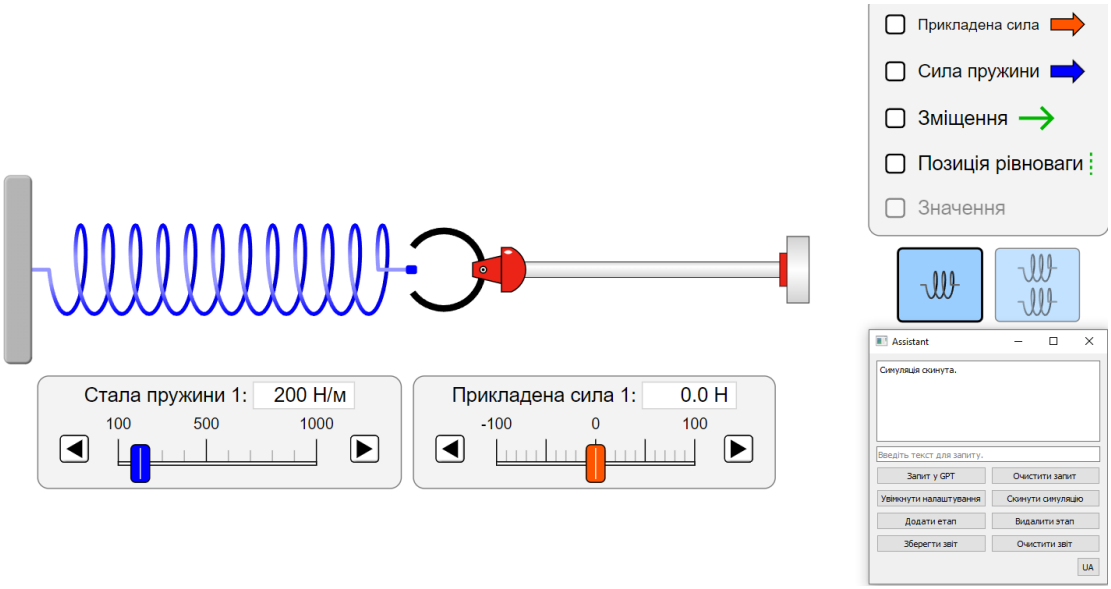

Рисунок 3.12 – Повне очищення симуляції

Також помічник має можливість змінити локалізацію для спрощення взаємодії з додатком користувачам, які не розуміють українську мову. Натискання відповідної кнопки призводить до повної заміни усіх функціональних компонентів на англійську мову. Результат роботи локалізації зображено на рисунку 3.13.

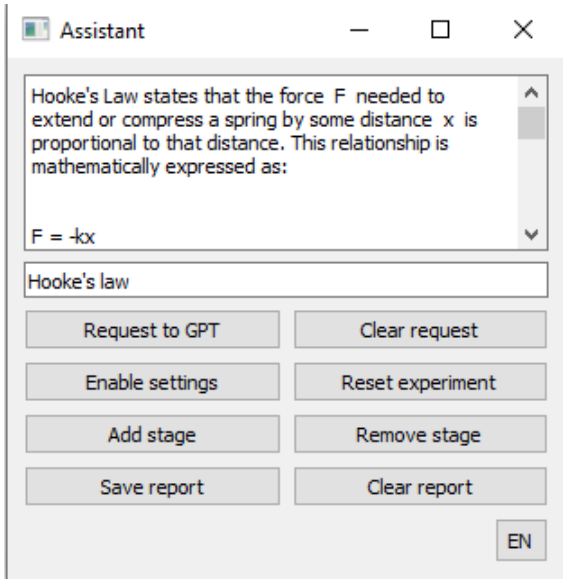

Рисунок 3.13 – Результат зміни локалізації

Розроблена система демонструє простоту використання завдяки продуманому інтерфейсу, який забезпечує легкий доступ до всіх ключових функцій. Інтуїтивність досягається завдяки чіткій організації елементів, зрозумілому маркуванню кнопок та логічній послідовності дій. Вона демонструє високу ефективність у вирішенні завдань, пов’язаних з інтерактивними моделями, і забезпечує зручність роботи для користувачів різного рівня підготовки. Агент може бути легко інтегрований в існуючі процеси та сприяти підвищенню продуктивності завдяки своїй адаптивності та багатофункціональності.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено програмне забезпечення інтелектуального агента як помічника в комп’ютерних інтерактивних моделях. Система дозволяє здійснювати запити до GPT-моделі, додавати або видаляти етапи симуляції, очищувати історію дій, налаштовувати параметри, створювати звіти та управляти ними, що забезпечує повну підтримку процесу інтерактивного моделювання.

На етапі розробки було виконано аналіз вимог до системи, визначено необхідний функціонал та створено архітектуру, яка забезпечує простоту інтеграції та зручність використання. Інтерфейс системи реалізовано з акцентом на інтуїтивність та доступність, що дозволяє ефективно використовувати програму навіть недосвідченим користувачам.

Під час роботи було протестовано всі основні функції системи, включаючи введення текстових запитів, очищення симуляцій, створення та редагування звітів. Проведені тестування підтвердили, що система відповідає визначеним вимогам, забезпечує стабільність роботи, коректно обробляє помилки та надає очікувані результати в різних сценаріях використання.

Результати розробки свідчать про те, що створена система має широкий потенціал для застосування в галузі комп’ютерних інтерактивних моделей. Вона дозволяє автоматизувати рутинні завдання, прискорювати процеси аналізу та прийняття рішень, а також підвищувати продуктивність роботи користувачів.

У програмній частині проєкту реалізовано взаємодію між користувачем, інтелектуальним агентом та системою генерації звітів. При виконанні програмної частини проєкту було обрано середовище розробки Visual Studio Code та мови програмування Python, а також HTML та CSS для веб-сторінки відповідно. Для реалізації графічного інтерфейсу користувача було використано бібліотеку PyQt5, що забезпечує створення інтуїтивного та

зручного інтерфейсу. А інтеграція з GPT-моделлю здійснена через API.

У подальшому система може бути вдосконалена завдяки розширенню функціональності, зокрема інтеграцію з іншими платформами, додавання підтримки різних мов інтерфейсу, а також удосконалення інструментів аналізу та візуалізації даних. Удосконалення звітності та підвищення її адаптивності під потреби користувачів сприятиме ще більшій ефективності застосування системи в різних контекстах.

Таким чином, розроблене програмне забезпечення відповідає сучасним вимогам та демонструє високу ефективність у вирішенні задач, пов’язаних із комп’ютерними інтерактивними моделями, забезпечуючи користувачам зручний та потужний інструмент для досягнення їхніх цілей.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. OpenAI developer platform [Електронний ресурс]. – Режим доступу: https://platform.openai.com/docs/overview (дата звернення: 16.09.2024). – Назва з екрана.

2. Interactive Simulations for Science and Math [Електронний ресурс] // simulations. – Режим доступу: https://phet.colorado.edu/ (дата звернення: 29.09.2024). – Назва з екрана.

3. Рассел, С. Штучний інтелект: сучасний підхід. Том 3 [Текст] : 4-е вид. / Стюарт Рассел, Пітер Норвіг ; пер. з рос. – К. : Прентіс Холл, 2022. – 640 с. – ISBN 9786177987733.

4. Лі, Х. Методи машинного навчання [Текст] : 1-е вид. / Хан Лі, Лу Лінь, Хуаньцян Цзен. – Сінгапур : Springer, 2023. – 548 с. – ISBN 978-9819939169.

5. Google Assistant [Електронний ресурс] // Google. – Режим доступу: https://assistant.google.com/intl/ru_ru/ (дата звернення: 15.10.2024). – Назва з екрану.

6. Selenium WebDriver як інструмент для автоматизованого тестування [Електронний ресурс] // QATestLab Training Center. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/selenium-webdriver/ (дата звернення: 18.10.2024). – Назва з екрану.

7. PyTorch — ваш фреймворк глибокого навчання [Електронний ресурс] // Habr. – Режим доступу: https://habr.com/ru/articles/334380/ (дата звернення: 21.10.2024). – Назва з екрану.

8. Introducing ChatGPT [Електронний ресурс] // OpenAI. – Режим доступу: https://openai.com/index/chatgpt/ (дата звернення: 24.10.2024). – Назва з екрану.

9. Тидвелл, Д. Розробка інтерфейсів. [Текст] : 3-е вид. / Дженифер Тидвелл, Чарли Брюэр, Эйнн Валенсия. – М. : Print2print, 2022. – 558 с. –

ISBN 978-5-4461-1646-1.

10. Flask’s documentation. [Електронний ресурс] // Pallets Projects. – Режим доступу: https://flask.palletsprojects.com/en/stable/ (дата звернення: 05.11.2024). – Назва з екрану.

11. Comprehensive Python Bindings for Qt v5 [Електронний ресурс] // PyPI. – Режим доступу: https://pypi.org/project/PyQt5/ (дата звернення: 06.11.2024). – Назва з екрану.

12. Дронов, В. Python 3 і PyQt 5. Розробка додатків [Текст] / Владимир Дронов, Николай Прохоренок. – СПб. : BHV-СПб, 2018. – 832 с. – ISBN 978-5-9775-3978-4.

13. PEP 8 – керівництво по написанню коду на Python [Електронний ресурс] // Pythonworld. – Режим доступу: https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html (дата звернення: 09.11.2024). – Назва з екрану.