ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Підсистема розробки програмного забезпечення на основі штучного інтелекту»

на здобуття освітнього ступеня магістра
зі спеціальності F3 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олексій ГОРОХ
(підпис) (Ім'я, ПРІЗВИЩЕ здобувача)

Виконав:
здобувач вищої освіти Олексій ГОРОХ
група КНДМ-62

Керівник:
науковий ступінь, вчене звання
Сергій ІЩЕРЯКОВ
к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання
_____
(Ім'я, ПРІЗВИЩЕ)

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність F3 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Гороху Олексію Руслановичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Підсистема розробки програмного забезпечення на основі штучного інтелекту

керівник кваліфікаційної роботи Сергій ІЩЕРЯКОВ к.т.н., доцент,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025р. №467

2. Строк подання кваліфікаційної роботи «26» грудня 2025р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, системи розробки програмного забезпечення, принципи функціонування моделей штучного інтелекту.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження впливу використання технологій штучного інтелекту на оптимізацію часу розробки програмного забезпечення

Аналіз існуючих систем розробки програмного забезпечення на основі штучного інтелекту
Розробка підсистеми написання програмного забезпечення на основі штучного інтелекту для зменшення часових витрат на створення програмного продукту

5. Перелік графічного матеріалу: презентація
   1. Актуальність та мета роботи
   2. Об’єкт та предмет дослідження
   3. Існуючі системи для програмування природною мовою
   4. Зменшення часових витрат на виконання операцій
   5. Порівняння підходів до розробки ПЗ
   6. Загальні вимоги до підсистеми
   7. Архітектура розробки підсистеми
   8. Загальний алгоритм роботи підсистеми
   9. Інтерфейс підсистеми
   10. Розроблені шаблони промтів
   11. Засоби реалізації підсистеми
   12. Висновки

6. Дата видачі завдання «15» вересня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.25	виконано
2	Вивчення матеріалів для аналізу систем розробки програмного забезпечення	05.11-12.11.25	виконано
3	Дослідження принципу генерування програмного коду системами з штучним інтелектом	13.11-18.11.25	виконано
4	Аналіз особливостей використання штучного інтелекту для написання програмного коду	19.11-23.11.25	виконано
5	Дослідження впливу використання штучного інтелекту для зменшення часу розробки програмного забезпечення	24.11-30.11.25	виконано
6	Написання програмного застосунку для поліпшення програмування	01.12-08.12.25	виконано
7	Оформлення роботи: вступ, висновки, реферат	09.12-12.12.25	виконано
8	Розробка демонстраційних матеріалів	13.12-15.12.25	виконано

Здобувач вищої освіти _____ Олексій ГОРОХ
(підпис) (Ім’я, ПРІЗВИЩЕ)

Керівник
кваліфікаційної роботи _____ Сергій ІЩЕРЯКОВ
(підпис) (Ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 83 сторінок, 29 рисунків, 1 таблиця, 11 джерел.

Об'єкт дослідження - системи автоматизації програмування та вплив використання штучного інтелекту на тривалість виконання завдань.

Предмет дослідження - зниження часових витрат та оптимізація процесу програмування природньою мовою за рахунок взаємодії з системами OpenAI.

Мета роботи - підвищення ефективності процесу розробки програмного забезпечення шляхом застосування штучного інтелекту.

Методи дослідження – аналіз методів розробки програмного забезпечення на основі штучного інтелекту та їх вплив на тривалість виконання задач, проектування та реалізація прототипу програмного застосунку.

В роботі проаналізовано найпопулярніші існуючі системи програмування на основі штучного інтелекту. Виявлено залежність між різними типами використання систем зі штучним інтелектом та тривалістю виконання поставлених задач. Обгрунтовано необхідність створення підсистеми NLP (Natural Language Processing) на базі технологій OpenAi з поліпшеною підтримкою української мови та базовими шаблонами промтів.

Галузь використання - IT-індустрія

АВТОМАТИЗАЦІЯ ПРОГРАМУВАННЯ, ЧАС ВИКОНАННЯ, ПРОМТ, ШТУЧНИЙ ІНТЕЛЕКТ, ПРИРОДНЯ МОВА ПРОГРАМУВАННЯ, NLP, OPENAI.

ABSTRACT

Text part of the master's qualification work: 83 pages, 29 figures, 1 table, 11 sources .

Object research - programming automation systems and the impact of using artificial intelligence on task completion time.

Subject of research - reducing time costs and optimizing the natural language programming process through interaction with OpenAI systems.

Goal works - increase the efficiency of the software development process through the use of artificial intelligence.

Research methods – analysis of software development methods based on artificial intelligence and their impact on task completion time, design and implementation of a software application prototype.

In work analyzes the most popular existing programming systems based on artificial intelligence. The dependence between different types of use of artificial intelligence systems and the duration of the tasks is revealed. The need to create an NLP (Natural Language Processing) subsystem based on OpenAi technologies with improved support for the Ukrainian language and basic prompt templates is justified.

Branch usage - IT industry

PROGRAMMING AUTOMATION, EXECUTION TIME, PROMT, ARTIFICIAL INTELLIGENCE, NATURAL LANGUAGE PROGRAMMING, NLP , OPENAI .

ЗМІСТ

ВСТУП

Динаміка розвитку сучасного світу відзначається стрімким технологічним розвитком, все більше сфер людського життя потребують постійних інновацій та вдосконалень в галузі інформаційних технологій.

З кожним роком розробники програмного забезпечення стикаються з більш складними та масштабними викликами у галузі розробки програмного забезпечення. Такий розвиток подій об'єктивно висуває вимогу оптимізації процесу створення та обслуговування програмного забезпечення, шляхом пошуку нових можливостей та інструментів. Серед найперспективніших рішень виокремлюється використання штучного інтелекту для створення розумних технологічних рішень, спрямованих на автоматизацію та оптимізацію процесів розробки програмного забезпечення.

Актуальність теми магістерської роботи зумовлена кількома факторами. По-перше, розширення попиту на ІТ-продукти серед дедалі більшої кількості сфер бізнесу та життєдіяльності людини. По-друге, поступове підвищення складності розроблюваного програмного забезпечення, що зумовлене зростанням запитів замовника та користувача. По-третє, такий потужний інструмент як штучний інтелект, надає можливості для більш ефективного виконання завдань з розробки, моніторингу та покращення розроблюваного програмного продукту, що потребує більш глибокого вивчення можливостей що надають інтелектуальні інструменти та їх впровадження в повсякденній професійній діяльності.

Метою магістерської роботи є дослідження впливу використання засобів штучного інтелекту на тривалість процесів розробки, тестування та налагодження програмного забезпечення, а також визначення методів оптимізації цих часових витрат. З метою досягнення поставленої мети в рамках дослідження будуть виконані наступні завдання:

1. Аналіз існуючих інструментів, розроблених для написання програмного коду, їх функціональних особливостей, переваг та недоліків на прикладі GitHub Copilot та Amazon Code Whisperer.

2. Дослідження впливу використання технологій штучного інтелекту на швидкість виконання таких завдань як: написання програмного коду, тестування та налагодження програмного забезпечення. Визначення основних аспектів та вимог до підсистеми автоматизації перетворення природної мови в програмний код.

3. Проведення розробки підсистеми перетворення природної мови в програмний код з використанням підготовлених промтів та налаштувань, дослідження ефективності написання програмного коду в порівнянні з традиційним підходом до взаємодії з системами штучного інтелекту в часовому вимірі.

4. Розробка практичних рекомендацій щодо системних вимог та інструкції користувача з необхідними роз’ясненнями щодо роботи з програмою для отримання вихідного коду.

Об'єктом дослідження є процеси автоматизації розробки програмного забезпечення та вплив використання систем на основі штучного інтелекту на тривалість виконання завдань, на відміну від класичного підходу до розробки та написання програмного продукту. Предмет дослідження - підсистеми розробки програмного забезпечення на основі штучного інтелекту які використовуються для оптимізації процесу написання програмного коду.

Основна увага зосереджуватиметься на теоретичних аспектах використання штучного інтелекту для написання програмного коду у порівнянні з класичним виконанням подібних завдань, що дозволить отримати достатньо широке уявлення про перспективу розвитку подібних засобів розробки програмного забезпечення у майбутньому.

Наукова новизна роботи полягає у дослідженні сучасних методів розробки програмного забезпечення та розробці необхідного інструменту для оптимізації витрат часу на написання програмного коду природно. (українською) мовою. Практична цінність дослідження полягає створенні необхідного інструменту для оптимізації виконання розробником поставлених задач та надання рекомендацій щодо його використання.

Розроблене програмне рішення може використовуватись програмістами різних рівнів кваліфікації як для виконання рутинних завдань так і для пошуку рішень для впровадження у подальшій роботі. Також може використовуватись для професійного розвитку та вдосконалення знань у сфері програмування, процесу створення та тестування програмного забезпечення, поліпшуючи його доступність для користувачів, які не мають глибоких знань у програмуванні.

Структура магістерської роботи складається з чотирьох основних розділів. Перший розділ містить основну теоретичну інформацію та огляд найпопулярніших інструментів для написання програмного коду на основі штучного інтелекту, їх особливостей та відмінностей. Другий розділ передбачає дослідження впливу використання технологій ШІ на оптимізацію написання програмного коду в часовому вимірі, та визначення необхідних вимог, функцій для розробки відповідної підсистеми. Третій розділ фокусується на розробці підсистеми для написання програмного коду природною мовою з послідовним описом реалізації та дослідження продуктивності виконання завдань з застосуванням реалізованого застосунку в порівнянні з прямим використанням моделі штучного інтелекту. Четвертий розділ має практичні рекомендації та приклад використання підсистеми.

Представлена магістерська робота націлена на розширення розуміння потенціалу використання штучного інтелекту у сфері створення програмних продуктів, оптимізації процесів написання програмного забезпечення в часовому вимірі для збільшення продуктивності як групи так і окремих розробників.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз існуючих систем розробки програмного забезпечення на основі штучного інтелекту

Штучний інтелект стає все більш важливим елементом у сучасній розробці програмного забезпечення. Його потенціал у забезпеченні інтелектуальних функцій і можливостей для автоматизації процесів робить його ключовим інструментом для вдосконалення розробки програмного забезпечення. У зв'язку з цим, необхідним етапом на шляху до успішної розробки підсистем на основі штучного інтелекту є аналіз існуючих систем розробки.

Цей розділ присвячений дослідженню та оцінці існуючих систем розробки програмного забезпечення на базі штучного інтелекту. В роботі досліджуються різноманітні підходи, методи та інструменти, які використовуються у таких системах. Мета - зрозуміти сильні та слабкі сторони схожих систем, їхні можливості та обмеження, а також визначити оптимальні підходи до розробки програмного забезпечення з використанням штучного інтелекту.

Аналіз існуючих підсистем розробки, що базуються на технологіях штучного інтелекту, допоміг виявити основні напрями розвитку у цій області, вибрати найбільш підходящі інструменти та методи для подальшої розробки підсистеми у рамках дипломної роботи [2].

Підсистема програмування - це частина комп'ютерної системи або середовища розробки програмного забезпечення, яка відповідає за надання інструментів та можливостей для створення, редагування, тестування і налагодження програмного коду.

Штучний інтелект – це галузь комп'ютерних наук, яка займається створенням систем та програм, здатних виконувати завдання, що зазвичай вимагають інтелектуальних здібностей людини. Основною метою штучного інтелекту є розробка алгоритмів та моделей, які дозволяють комп'ютерам аналізувати дані, робити висновки, вирішувати проблеми та вчитися з досвіду.

Основні характеристики штучного інтелекту включають в себе спроможність розпізнавання образів та мови, здійснювати обробку природної мови (NLP), аналізувати великі обсяги даних, розрізняти закономірності та тенденції, приймати рішення на основі наявних даних, вирішувати проблеми у складних ситуаціях, а також навчатися та вдосконалювати свої функції з часом.

Аналіз існуючих системи програмування природньою мовою на основі штучного інтелекту включає оцінку та порівняння різних інструментів та методів, які використовуються для розробки систем, здатних розуміти та генерувати тексти людською мовою. Основні етапи аналізу включають:

1. Огляд інструментів. Дослідження існуючих систем, що використовують штучний інтелект для обробки природної мови. Це включає платформи машинного навчання, бібліотеки обробки природної мови (NLP), інструменти автоматизованого аналізу текстів і т.д.

2. Аналіз функціональності. Оцінка можливостей системи з аналізу та генерації текстів природною мовою, включаючи рівень точності розуміння контексту запиту, розпізнавання патернів мови та автоматичну генерацію текстів.

3. Оцінка ефективності. Аналіз функціонування підсистеми в реальних умовах використання, включаючи швидкість обробки даних, витрати ресурсів та масштабованість.

4. Оцінка надійності і точності. Оцінка надійності системи у різних сценаріях використання з великою кількістю користувачів, та рівня точності розпізнавання та вирішення конкретних задач відповідно до запиту користувача.

5. Оцінка інтеграції. Можливості інтеграції підсистеми з існуючими програмними рішеннями та середовищами розробки, зокрема з мовами програмування та фреймворками.

6. Тренди та перспективи. Вивчення тенденцій у розвитку технологій обробки природної мови на основі штучного інтелекту та визначення перспективних напрямів впровадження та розвитку. [9]

1.2 Сучасні інструменти ШІ-програмування

1.2.1 Загальні відомості

Автоматизація рутинних процесів – головна перевага використання штучного інтелекту в процесі написання програмного коду. Аналізуючи шаблони в коді та автоматично генеруючи фрагменти коду, інструменти кодування зі штучним інтелектом допомагають скоротити час і зусилля, які розробники витрачають на написання повторюваних фрагментів. Це не тільки підвищує ефективність, але й звільняє час розробників для експериментів з новими, інноваційними підходами до розробки програм. Коли інтелектуальні технології беруть на себе виконання рутинних завдань, розробники можуть зосередитися на вирішенні проблем і пошуку нових ідей, що в кінцевому результаті веде до створення більш продуманого, досконалішого і оптимізованого коду.

Штучний інтелект, використовуючи алгоритмічні та логічні методи програмування, значно поліпшує генерацію та оптимізацію коду. Генерація фрагментів коду та оптимізація його продуктивності - це два завдання, які розробники мають можливість автоматизувати за допомогою інструментів кодування на основі штучного інтелекту. Ці інструменти виявляють повторювані проблеми в коді та пропонують рішення, допомагаючи розробникам створювати краще та надійніше програмне забезпечення. Це не тільки знижує витрати часу на пошук недоліків, але й забезпечує вищий рівень якості коду в цілому [10].

Інтелектуальне налагодження та виявлення помилок - це дві сфери, де системи штучного інтелекту мають якісний вплив на написання програмного коду у майбутньому. Пошук і виправлення помилок у коді за допомогою традиційних методів налагодження є досить кропітким, виснажливим і трудомістким процесом. Раніше розробникам доводилося шукати помилки в коді вручну, аналізуючи правильність написання кожної строки окремо та їх взаємодію, але тепер вони можуть покластися на інструменти налагодження на основі штучного інтелекту, які зроблять важку і довготривалу роботу за них. Ці інструменти можуть аналізувати код з набагато вищою швидкістю та одразу великими фрагментами, виділяти проблемні ділянки, та пропонувати варіанти їх

виправлення на основі відомих та заздалегідь проаналізованих успішних програмних рішень, що значно прискорює процес налагодження. Використовуючи інтелектуальні технології для прискорення, розробники можуть приділяти більше часу важливішим завданням і пошуку та впровадженню нових ідей.

Завдяки аналізу коду та рекомендаціям на основі штучного інтелекту, розробники кардинально змінюють свій підхід до кодування. Тепер, завдяки штучному інтелекту, можна покладатися на передові алгоритми та методи для аналізу, пошуку оптимальних рішень щодо структури майбутнього коду та надання кращих варіантів для їх реалізації розробникам. Ці інструменти зі штучним інтелектом можуть аналізувати запити користувача, його побажання та чіткі умови яким повинен відповідати майбутній програмний продукт, вносити пропозиції щодо реалізації та навіть пропонувати альтернативні підходи до вирішення поставлених задач. Завдяки використанню штучного інтелекту для пошуку найоптимальніших та розумних рішень, щодо вихідного продукту, розробники можуть пришвидшити робочий процес, зменшивши кількість помилок як з самого початку розроблення (на етапі планування та постанови задач) так і під час написання коду, створюючи якісніше програмне забезпечення.

Технології штучного інтелекту, безсумнівно, впливають на майбутнє кодування, і наступні представлені інтелектуальні інструменти суттєво трансформують індустрію програмування.

1.2.2 GitHub Copilot

За допомогою машинного навчання та штучного інтелекту GitHub Copilot, інструмент для створення коду на основі штучного інтелекту, може пропонувати фрагменти коду та автоматично доповнювати, під час його введення. GitHub Copilot, розроблений дослідницькою лабораторією OpenAI та компанією GitHub - це інструмент для створення коду, який підтримує більше десятка мов та працює на базі моделі OpenAi Codex. Він був навчений на мільярдах рядків коду, використовуючи дані з публічних репозиторіїв для вдосконалення моделі. Він

добре працює з популярними редакторами коду, такими як Microsoft Studio Code, і може запропонувати повні функції та алгоритми на основі описів природною мовою. З GitHub Copilot розробники можуть виконувати більшу кількість завдань за менший час і бути продуктивнішими, на відміну від традиційного методу проектування та написання коду самотужки.

Основні функції GitHub Copilot включають:

1. Автоматичне доповнення коду;
2. Підказки для використання API та бібліотек;
3. Документацію для функцій та класів;
4. Виявлення потенційних помилок;
5. Створення тестів.

GitHub Copilot позиціонують як продукт, що виконує нецікаві рутинні завдання, даючи змогу розробникам фокусуватися на високорівневих завданнях. Автори застосунку не мали на меті побудувати систему що може замінити розробників автоматизуючи послідовно кожен етап процесу створення програмного забезпечення і не передбачали виконання завдань зі складною логікою, проте з кожним новим продуктом його можливості зростають. Так, Copilot Labs пропонує функції рефакторингу, автоматичної документації, перекладу та пояснення коду, а також покращену функцію генерації тестів (рис. 1.1), Copilot X — автоматичну генерацію pull-реквестів та чат із підтримкою GPT-5. Компанія відкриває доступ до нових продуктів спочатку для середовища Visual Studio Code, а після офіційного релізу – для всіх інших IDE [2].

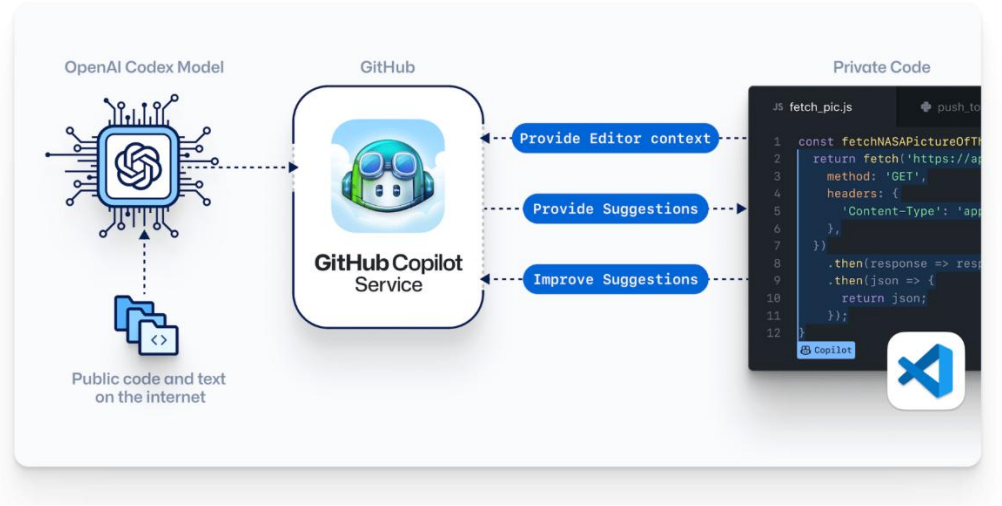


Рисунок 1.1 — GitHub Copilot

Переваги GitHub Copilot:

1. Прискорення розробки.

2. Швидке створення прототипів програмного коду для експериментів та валідації ідей.

3. Спрощення рутинних завдань, таких як генерація шаблонного коду або документації.

4. Підтримка багатьох мов програмування (найкраще працює з популярними мовами як Python, JavaScript, C++ та ін..) та фреймворків.

Недоліки GitHub Copilot:

1. Код не завжди працює і не тестується інструментом.

2. У коді трапляються критичні помилки та вразливості, які потребують додаткового доопрацювання.

3. Якщо розробник не розуміє, як працюють згенеровані рішення, це може призвести до виникнення вразливостей та нездатності швидко змінювати програмний код.

4. Ліцензійна політика GitHub Copilot та непрозора схема збереження даних.

5. Інструмент не завжди розуміє специфічні вимоги та контекст проєкту.

6. Проблеми з доступністю без використання VPN та перебоями роботи сервісу на території України, що створює додаткові труднощі для українських розробників, особливо якщо потрібно інтегрувати Copilot у робочий процес.

1.2.3 Amazon CodeWhisperer

Amazon CodeWhisperer - це інструмент на основі штучного інтелекту, який розроблений компанією AWS, навчання моделі якого здійснювалося на даних що включають публічно доступний код, внутрішній код Amazon, документацію, форуми та може генерувати, оптимізовувати та покращувати якість коду розробників. Новий код генерується шляхом аналізу існуючого коду з урахуванням наявних шаблонів та найкращих практик, а потім застосовує методи машинного навчання до цих висновків. CodeWhisperer є досить потужним ресурсом (рис. 1.2) для будь-якої команди розробників завдяки своїй здатності виявляти та виправляти помилки.

Рисунок 1.2 – Amazon CodeWhisperer загальні можливості

CodeWhisperer у режимі реального часу в інтегрованому середовищі розробки (IDE) генерує пропозиції для коду, починаючи від його фрагментів і закінчуючи повноцінними функціями, на основі поточних коментарів та наявного

коду. Сервіс також підтримує заповнення інтерфейсу командного рядка та переклад у ньому з природної мови на команди Bash [2].

Недоліки:

1. Відсутня повноцінна підтримка української мови;

2. Відсутність логіки задання промтів до системи,

3. Орієнтовність на AWS екосистему (якщо розробник використовує інші хмарні сервіси або локальну інфраструктуру, використання цього інструменту може бути менш ефективним та зручним)

Було розглянуто два інструменти на основі штучного інтелекту, їх потенціал та основні функції, які допомагають розробникам писати код швидше та ефективніше. Обидва інструменти можуть генерувати програмний код на основі природної мови, коментарів та наявного початкового коду. В обох ситемах прослідковується спільна проблема, а сааме – відсутність прямої взаємодії з українською мовою та відсутні промт-інструкції для кращої можливості взаємодії розробника та системи.

1.3 Обгрунтування необхідності створення підсистеми

Існуючі рішення, такі як GitHub Copilot та Amazon CodeWhisperer, орієнтовані переважно на англомовних користувачів і не забезпечують належної підтримки української мови. Українські розробники часто стикаються з проблемами через відсутність якісних інструментів для автоматизованої обробки текстів та генерації коду українською мовою. Створення спеціалізованої системи NLP з поліпшеною підтримкою української мови, за рахунок прямої взаємодії з інтелектуальними технологіями OpenAI, допоможе вирішити ці проблеми та підвищить продуктивність роботи, а базові шаблони промтів допоможуть більш ефективній взаємодії між користувачем та системою.

У контексті глобалізації існує ризик зниження використання та популярності української мови в технологічній сфері з причин домінування англійської мови у сфері IT, недостатньої локалізації технологічних продуктів та освітніх тенденцій, коли навчальні курси з програмування ведуться англійською

мовою з використанням англомовних матеріалів, а українські переклади відсутні або є не повними. Створення системи NLP, з фокусом на українську мову, сприятиме її збереженню та активному розвитку, підвищуючи її статус та застосування в сучасних технологіях.

Україномовні користувачі зможуть отримувати більш якісні та релевантні результати, використовуючи систему, яка розуміє їхню мову та культурний контекст. Це значно покращить загальний користувацький досвід та підвищить задоволеність від використання технологій на базі штучного інтелекту.

Розробка системи передбачає створення базових шаблонів промтів, які забезпечать користувачів ефективними інструментами для генерації текстів та програмного коду. Ці шаблони будуть спеціально налаштовані для використання української мови, враховуючи її граматичні, лексичні та стилістичні особливості. Це значно спростить використання системи та підвищить її корисність для широкого кола користувачів.

Висновки до розділу 1

Обидва інструменти, GitHub Copilot та Amazon CodeWhisperer, мають свої унікальні переваги та недоліки. Copilot є більш старим продуктом з ширшою підтримкою мов програмування та глибокою інтеграцією з VS Code, але він менш ефективний для неангломовних користувачів. CodeWhisperer, з іншого боку, пропонує кращу інтеграцію з Amazon Web Services, але є новішим продуктом з потенційними обмеженнями в підтримці мов. Обидва сервіси не мають шаблонів промтів для користуачів та відсутня повноцінна підтримка української мови.

Необхідність створення системи NLP на базі OpenAI з поліпшеною підтримкою української мови та розробленими базовими шаблонами промтів є очевидною. Така система буде сприяти розвитку технологій, збереженню мови, поліпшенню освіти та підвищенню конкурентоспроможності українських компаній. Вона стане важливим кроком у підтримці та розвитку української мови в цифрову епоху, забезпечуючи україномовних користувачів ефективними інструментами для роботи з промтами та кодом.

Завдання на магістерську роботу:

Дослідити вплив використання систем штучного інтелекту на витрати часу яке необхідне для виконання завдань різної складності на відміну від класичного методу програмування.

Створити набір стандартних шаблонів для генерації програмного коду, охоплюючи різні аспекти програмування.

Інтегрувати OpenAI GPT-5 для генерації коду на основі текстових запитів користувачів, оптимізувати параметри взаємодії з API для забезпечення якості згенерованого коду

Створити веб-інтерфейс для введення запитів природньою мовою та отримання коду, реалізувати серверну частину з Flask для обробки запитів та взаємодії з OpenAI.

2 ДОСЛІДЖЕННЯ І ПОСТАНОВКА ЗАДАЧІ

2.1 Дослідження впливу використання технологій штучного інтелекту на оптимізацію часу розробки програмного забезпечення.

З кожним роком все більше сфер людського життя пов’язує себе з використанням програмного забезпечення, як то у повсякденних побутових процесах так і у корпоративних, економічних та керівних що ставить перед сучасними розробниками програмного забезпечення досить серйозні завдання. В свою чергу ускладнення завдань потребує росту ефективності розробки та вдосконалення програмного забезпечення а також його постійної підтримки в робочому стані. Теперішні економічні тенденції накладають додаткові обмеження та труднощі підприємствам які спеціалізуються на розробці програмного забезпечення та підприємствам які в своєму штаті мають окремий IT відділ.

Для більшості підприємств стає занадто витратно наймати додатковий персонал а законодавство європейських країн досить суворо слідкує за дотриманням норм з праці. Людський фактор теж накладає досить серйозні обмеження на ресурс уваги та продуктивності працівника, найвища продуктивність людини триває протягом чотирьох годин, після чого іде на спад, і тривала праця на виснаження кратно підвищує шанс допущення помилок, виправлення яких вартує додаткової витрати часу та фінансових втрат підприємства. Окрім цього зачасту вирішення задач з написання програмного забезпечення, його підтримки носить терміновий характер, особливо коли відбувається позаштатна ситуація пов’язаною з працездатністю системи, пов’язана з потребами клієнта у нових функціях, збільшення навантаження на програмний комплекс або зовнішнім впливом на цілісність та безпеку системи та баз даних які вона використовує.

Враховуюче вищезазначене, надважливим показником для виконання операцій з програмним забезпеченням являється часовий показник. Зменшення витрат часу на виконання завдань підвищує можливість виконувати більше завдань за певний проміжок часу що веде до росту прибутку як компаній так і

окремих розробників програмного забезпечення, збільшуючи привабливість відповідних професій серед поколінь які тільки визначаються з майбутньою професією так і старших поколінь які вирішили опанувати додаткові навички.

Класичний спосіб написання програмного коду – досить кропіткий процес який вимагає чіткого розуміння мети та очікуваного кінцевого результату. Розробка починається з постановки завдання та побудови архітектури майбутнього програмного забезпечення, проектуються функції та взаємодія кожного блоку з іншими і закінчується тестуванням цілісного програмного застосунку на надійність, безпеку та ефективності використання системних ресурсів. Тому цей спосіб потрибує найбільшої затрат часу на виконання роботи та має шанс виникнення помилок та випадкових програмних вразливостей. Чим складніший проект тим вищий рівень знань вимагається від програміста які за часту неможливо опанувати швидко що у свою чергу те призводить до збільшення часу виконання завдання.

Сучасна тенденція зменшення витрати часу шляхом покладання процесів на системи штучного інтелекту має досить широкий потенціал. Використання відповідних систем суттєво прискорює розробку програмного забезпечення та написання коду шляхом автоматизаці рутинних процесів, пошуку та виправлення помилок, обрання оптимальних алгоритмів.

2.1.1 Використання генеративних моделей для автоматизації написання програмного коду.

Окремі сучасні розробники програмного забезпечення та підприємства ринку інформаційних технологій прагнуть пришвидшити розробку програмного забезпечення. Розробники можуть очікувати революційної економії часу завдяки використанню генеративного штучного інтелекту. Однак, щоб використати весь потенціал цієї революційної технології їм знадобиться більше ніж просто інструменти.

Наразі саме генеративна модель штучного інтелекту може задовольнити потреби розробників ПЗ тому що цей тип системи здатний створювати новий

контент, який імітує структуру, стиль або логіку даних, на яких вона була навчена. Такі моделі не просто аналізують чи класифікують інформацію а і генерують нові тексти, зображення, аудіо, відео або програмний код, що є унікальними але статистично схожими на вихідні приклади.

Основою генеративних систем є глибокі нейронні мережі, які навчаються виявляти закономірності у великих обсягах даних. У процесі навчання модель формує ймовірнісне представлення зв’язків між елементами даних, що дозволяє відтворювати схожі, проте нові комбінації.

Основні характеристики:

1. Творча здатність – моделі здатні генерувати контент які виглядають створеними людиною;
2. Навчання на великих об’ємах інформації – ефективність напряму залежить від якості та обсягу даних;
3. Випадковість – результати щоразу можуть бути різними, навіть за однакового запиту;
4. Адаптивність – можливість до навчання на специфічних джерелах інформації (наприклад - технічна документація, оцифрованінавчальні посібники).

Моделі штучного інтелекту, розроблені компанією OpenAi, зокрема серії GPT (Generative Pre-trained Transformer), вважаються одними з найефективніших рішень для автоматизованого написання програмного коду. Їхня перевага полягає у використанні трансформанної архітектури, яка забезпечує глибоке розуміння контексту та логічних залежностей у текстах програмування. Завдяки навчаннях на величезних об’ємах даних, що включають вихідні коди з різних мов програмування, моделі OpenAi здатні генерувати синтаксично правльний, логічно зв’язаний і структурований код для вирішення конкретних завдань.

Крім того дані моделі здатні адаптуватися до різних стилів кодування та враховувати контекст запиту користувача, що значно поліпшує результат. Вони підтримують широкий спектр мов програмування, включаючи Python, Java, C++, SQL та інші, що робить їх універсальними інструментами для розробників. Завдяки використанню механізмів контекстного аналізу та логічного

прогнозування такі системи можуть не лише писати код а і пояснювати його, знаходити помилки, та пропонувати рішення для оптимізації.

У поєднанні з високою швидкістю генерації, здатністю до діалогогової взаємодії та контекстного мислення, моделі OpenAi можна важати одним з найкращих інструментів для автоматизованої розробки програмного забезпечення.

Завдяки правильному підвищенню кваліфікації та забезпечення можливостей для розвитку відповідних навичок досягається приріст швидкості виконання задач що у чвою чергу означає підвищення продуктивності порівняно з попередніми результатами.

Дослідження компанії McKinsey (рис. 2.1) показує значне зменшення витрат часу на виконання завдань, однак економія часу може суттєво відрізнятися залежно від складності завдання та досвіду розробника [11].

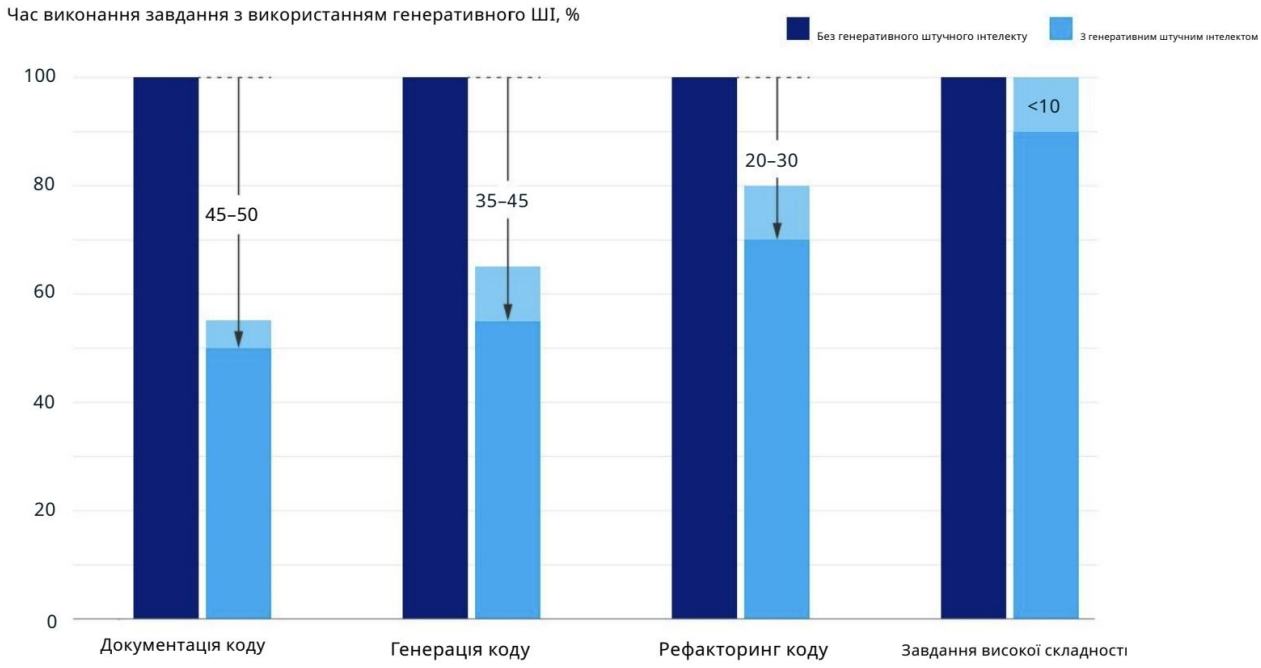


Рисунок 2.1 – Час виконання операцій.

Економія часу скорочується до менш ніж 10 % для завдань, які розробники вважали тими що мають високу складність, наприклад, через недостатнє знайомство з необхідним програмним фреймворком. Подібний результат состерігався серед розробниківз досвідом роботи менше року.

Використання цих інструментів не призводить до втрати якості заради швидкості. Якість коду стосовно помилок, зручності обслуговування та читабельності (легкості розуміння розробником вихідного коду) була дещо кращою в коді, написаному за допомогою штучного інтелекту. Відгуки учасників дослідження свідчать про те що вони активно використовували інструменти для досягнення вищої якості що говорить про те що найкращим рішенням буде використовувати подібну технологію для розширення можливостей розробників та поліпшення робочого процесу а не для їх заміни. Для підтримання якості коду на професійному рівні розробникам потрібно розуміти характеристики, що визначають якісний код, і надавати інструменту відповідні вказівки для отримання коректних результатів.

Інструменти на основі штучного інтелекту забезпечують значне підвищення продуктивності у чотирьох ключових сферах:

1. Прискорення ручної та монотонної роботи. Генеративний ШІ може виконувати рутинні завдання, такі як автоматичне заповнення стандартних функцій, шо використовуються в кодуванні, завершення кодових операторів під час введення розробником та документування функціональності коду в заданому стандартному форматі на основі підказок розробника. Виконуючи такі завдання ці інструменти можуть звільнити вільний час для вирішення складніших завдань та пришвидшити впровадження нових можливостей програмного забезпечення.

2. Швидкий старт першого варіанту нового коду. Зіткнувшись з порожнім екраном, розробники з генеративними інструментами на основі штучного інтелекту можуть формувати запити як в окремому вікні так і в інтегрованому середовищі (IDE), яке вони використовують для розробки програмного забезпечення. Генеративні інструменти можуть надавати корисні пропозиції щодо коду. Це дозволяє подолати творчу кризу та швидше розпочати роботу.

3. Прискорення оновлень існуючого коду. Використовуючі ці інструменти з ефективними підказками розробник може швидше вносити більшу кількість змін до існуючого коду. Наприклад, що-б витрачати менше часу на адаптацію коду з онлайн-бібліотеки та вдосконалення попередньо написаного коду розробник має

змогу копіювати та надсилати ітераційні запити, надаючи завдання інструментові налаштувати параметри на основі заданих критеріїв.

4. Підвищення здатності розробників вирішувати нові завдання. Хоча економія часу розробників завдяки генеративним інструментам на основі штучного інтелекту була скромнішою для складних завдань, ці технології можможуть допомогти розробникам швидко освіжити знання незнайомої кодової бази, мови чи фреймворку, необхідних для виконання завдання. Крім того, коли розробники стикаються з новим викликом, вони можуть використати ці інструментіи що-б отримати допомогу яку вони могли-б отримати від досвідченого колеги, наприклад – пояснити нові концепції, синтезувати інформацію (наприклад порівняти код з різних репозиторіїв) та надати покрокові інструкції щодо використання фреймворку для виконання роботи. Таким чином, розробники, які використовують генеративні інструменти на основі штучного інтелекту для виконання складних завдань, мають більше шансів виконати ці завдання у відведені терміни, ніж ті, хто не користується цими інструментами.

2.1.2 Застосування інтелектуальних систем для прискорення тестування та налагодження програмного забезпечення.

Розробка програмного забезпечення супроводжується високими вимогами до якості та надійності продукту. Одним з найважливіших етапів життєвого циклу програмного продукту є тестування, яке забезпечує виявлення помилок, перевірку відповідності вимогам і гарантує стабільність роботи системи. Однак процес тестування часто є трудомістким, дорогим і займає значну частину часу розробки. Саме тому застосування інтелектуальних систем у цій сфері набуває дедалі більшого значення.

Одним із ключових напрямів застосування інтелектуальних систем є автоматичне генерування тестів. На основі аналізу вимог або вихідного коду програми такі системи можуть створювати тестові випадки, які охоплюють більшу кількість можливих сценаріїв, ніж це може зробити людина. Це дозволяє

суттєво зменшити час, необхідний для ручного створення тестів і зменшує шанс пропуску проблемної ділянки коду.

Ще одним важливим аспектом є оптимізація ресурсу регресійного тестування. За допомогою алгоритмів машинного навчання можна виявити ті частини коду, які найбільш схильні до помилок після внесення змін, і відповідно пріоретизувати виконання тестів. Це скорочує загальний час тестування, оскільки ресурси зосереджуються на найбільш критичних ділянках системи.

Інтелектуальні системи також активно використовуються для аналізу результатів тестування. Завдяки застосуванню методів класифікації та аналізу аномалій, штучний інтелект здатен автоматично визначати ймовірні причини збоїв, групувати схожі помилки та навіть пропонувати шляхи їх усунення. Це значно підвищує ефективність роботи команди тестувальників та скорочує час на пошук та усунення дефектів програмного коду.

Документування та можливість пояснення процесу тестування, виявлення дефектів та аномалій коду та пропонування робочих рішень дає змогу розробникам та тестувальникам підвищувати свою кваліфікацію під час виконання завдання що суттєво економить час на опануванні додаткових навичок та знань у відповідній сфері діяльності.

Уся отримана інформація під час тестування програмного продукту дає чітке розуміння щодо подальшого налагодження системи та можливостей вдосконалення та розширення функціоналу зберігаючи при цьому достатній рівень швидкодії системи.

Можна зазначити що застосування інтелектуальних систем для налагодження програмного забезпечення є ключовим кроком для створення більш надійних, ефективних та безпечних ІТ-рішень. Використання систем штучного інтелекту дозволяє не лише скоротити час виявлення та виправлення помилок, але й налагоджувати програмне забезпечення під швидке та безпомилкове виконання завдань що вимагається конкретною цільовою аудиторією користувачів. У майбутньому такі системи, ймовірно, стануть невід’ємною частиною кожного

етапу розробки програмного забезпечення, забезпечуючи новий якісний рівень автоматизації.

2.1.3 Порівняння продуктивності традиційних і ШІ-орієнтовних підходів до розробки програмного забезпечення.

Традиційний підхід передбачає послідовний процес розробки програмного забезпечення, який включає планування, написання коду, тестування, налагодження та розгортання. Основними інструментами при цьому є середовища розробки (IDE), системи контролю версій (Git), фреймворки для тестування (JUnit, PyTest) тощо. Усі дії виконуються в ручну або частково автоматизовані скриптами.

Наприклад, під час створення веб-додаткукоманда з п’яти розробників витрачає в середньому 180-200 годин на реалізацію основного функціоналу, при цьому близько 25-30% часу йде на налагодження помилок. Аналіз коду здійснюється вручну або з допомогою статистичних аналізаторів, які лише виявляють синтаксичні помилки, але не надають рекомендацій щодо оптимізації логіки або структури коду.

Недоліки традиційного підходу полягають у високій трудомісткості, суб’єктивності рішень та залежності від досвіду розробників. Крім того, він має обмежену гнучкість у реагуванні на зміни вимог замовника або технічні виклики. Усі ці фактори значно збільшують витрати часу на розробку програмного забезпечення.

ШІ-орієнтований підхід до розробки програмного забезпечення базуються на використанні інтелектуальних інструментів, які автоматизують частину або більшість процесів створення програмного забезпечення. Прикладами таких інструментів є системи які спеціалізуються на генерації коду (як OpenAI Codex, GitHub Copilot), автоматизовані тестувальні системи на базі машинного навчання (Diffblue Cover, Testim), а також інструменти аналітики помилок і продуктивності з елементами штучного інтелекту.

У реальних умовах даний підхід демонструє значне скорочення часу розробки. Наприклад, за інформацією з звітів компанії GitHub у 2024 році, використання GitHub Copilot дозволяє зменшити час написання коду всередньому на 55%, а кількість помилок у первинному коді – на 37% у порівнянні з традиційним підходом. Це зумовлено тим, що системи штучного інтелекту допомагає розробникам автоматично доповнювати код, генерувати тести та навіть оптимізовувати структури функцій.

Слід зауважити що зменшення помилок у програмному коді зумовлено ще й мінімізацією людського фактору коли по причині виснаження та наявності негараздів у особистому житті та корпоративних відносинах відбувається зниження уваги та контролю а відволікання на сторонні розмови та перерви здатні перервати логічний хід думок шо також підвищує шанс на допущення помилок чи випадкового пропуску проблемних ділянок коду при тестуванні.

Порівняльна таблиця демонструє, що ШІ-орієнтований підхід значно підвищує ефективність та знижує час на написання програмного коду (таб. 2.1).

Таблиця 2.1 – Порівняння підходів до розробки ПЗ

	Традиційний підхід	ШІ-рієнтований підхід	Відмінність (%)
Середній час розробки модуля	180 годин	110 годин	-40%
Кількість помилок на 1000 рядків коду	8-10	5-6	-40%
Час на тестування та налагодження	60 годин	30 годин	-50%
Якість коду (за оцінкою аналізатора)	7.2/10	9.0/10	+31%

Основна перевага полягає в автоматизації рутинних завдань таких як написання шаблонного коду, виправлення типових помилок та тестування. Це дозволяє розробникам зосередитися на логіці та архітектурі системи, а не на технічних деталях.

Розглянемо приклад оптимізації модуля управління користувачами у веб-додатку SmartTask Manager. У традиційному підході команда розробників витратила п’ять робочих днів на створення, тестування та виправлення помилок. При застосуванні інструментів на основі штучного інтелекту (Copilot, SonarQube

AI nf Test GPT) цей самий обсяг роботи був виконаний за три дні. Крім того, продуктивність системи після оптимізації збільшилася на 15%, а обсяг коду зменшився на 12% за рахунок усунення надлишкових фрагментів.

Використання штучного інтелекту також дозволило виявити потенційні вразливості безпеки, які не були помічені під час ручного аналізу. Це свідчить про те що інтелектуальні системи не лише прискорюють процес розробки, але й підвищують якість і безпечність кінцевого продукту.

Можна зробити висновок що ШІ-орієнтовані підходи більш ефективні в умовах сучасної розробки програмного забезпечення. Вони сприяють підвищенню продуктивності, покращення якості та зменшення часу на розробку, що робить їх перспективним напрямом розвитку IT-індустрії в найближчі роки.

2.2 Загальні вимоги до підсистеми

Розробка програмного забезпечення — це сучасна сфера бізнесу, що постійно розвивається, і вимагає постійних інновацій та ефективного вирішення проблем. Штучний інтелект стає потужним інструментом у різних галузях, і його вплив на розробку програмного забезпечення не є винятком. Одним із найпомітніших досягнень у сфері систем штучного інтелекту є розробка OpenAI GPT-5 – передової мовної моделі. Ця модель надає можливості використання генеративного штучного інтелекту, відкриваючи нові можливості для команд розробників програмного забезпечення.

GPT-5 розроблено для забезпечення ефективної комунікації між користувачем і комп’ютерними системами за допомогою природної мови. Його універсальність і адаптивність роблять його цінним інструментом для різноманітних додатків, включаючи підтримку клієнтів, віртуальних помічників, створення контенту та, що найважливіше, розробку програмного забезпечення.

У процесі розробки програмного забезпечення варіанти використання визначаються за конкретними сценаріями або умовами, у яких певне програмне забезпечення використовується для досягнення поставлених цілей або отримання прогнозованих результатів. Ці варіанти сприяють у визначенню особливостей

взаємодії між користувачами та програмним забезпеченням, описуючи очікувану поведінку та функціональні можливості системи. Вони використовуються для фіксації та передачі вимог, керування процесами розробки та перевірки функціональності програмного забезпечення. Ось кілька типових прикладів використання в розробці програмного забезпечення:

1. Взаємодія з користувачем. Приклади використання описують, як користувачі взаємодіють із програмним забезпеченням, зокрема, які дії вони вживають та як система очікує відповіді. Це включає такі кроки, як вхід, створення профілів користувачів, заповнення форм та здійснення транзакцій.

2. Системна інтеграція. Варіанти використання описують способи інтеграції програмного забезпечення з іншими системами, такими як сторонні API, бази даних або зовнішні сервіси. Це сприяє ефективному обміну даними та взаємодії між різними компонентами програмного забезпечення.

3. Обробка помилок. Варіанти використання визначають процеси обробки помилок або винятків у програмному забезпеченні. Вони включають кроки з виявлення, звітування та обробки помилок, щоб запобігти збоям системи та надати користувачам зрозумілі повідомлення про помилки.

4. Управління даними. Варіанти використання охоплюють операції з даними, такі як збереження, отримання, оновлення та видалення даних у програмній системі або базах даних. Це допомагає забезпечити цілісність даних, їх безпеку та належне застосування.

5. Звітність і аналітика. Варіанти використання передбачають створення звітів, проведення аналізу даних і надання розуміння на основі зібраних даних. Це допомагає користувачам приймати обґрунтовані рішення та отримувати цінну інформацію про перебіг системних процесів.

Варіанти використання є важливим інструментом для ухвалення та надання потрібної функціональності програмному забезпеченню, що сприяє ефективному процесу розробки, тестування та доставки надійних програмних систем.

Мовна модель GPT-5, як сучасний інструмент обробки природної мови (NLP), використовує передові технології машинного навчання. Він приймає

введені дані користувача і генерує пропрацьовані, актуальні та граматично правильні відповіді. Даний механізм на основі штучного інтелекту навчається на великому обсязі даних, таких як бесіди з онлайн-форумів, відкритий програмний код, інструкції, книги, та інше. Після цього генеративний штучний інтелект використовує цю інформацію для того, щоб зрозуміти контекст розмови та створити відповідь відповідно до цього контексту.

Моделі штучного інтелекту OpenAI, зокрема GPT (включно з GPT-5) навчаються за допомогою комбінованого підходу, який поєднує кілька ключових методів машинного навчання:

1. Навчання з учителем (Supervised Learning). На початковому етапі модель тренують на великих масивах текстів, де відомо, які відповіді є правильними. Мета етапу - навчити модель передбачати наступне слово в тексті на основі попереднього контексту. Для цього використовуються величезні масиви даних з інтернету, книг, наукових статей, програмного коду, тощо.

2. Навчання з підкріпленням від людського зворотного зв’язку (Reinforcement Learning from Human Feedback, RLHF). Після базового навчання модель додатково вдосконалюють за допомогою оцінок від людей-анотаторів. Люди порівнюють кілька варіантів відповідей моделі а потім модель вчиться віддавати перевагу тим варіантам, які більше відповідають людським очікуванням – тобто є точними, доречними та безпечними. Для цього застосовується метод Reinforcement Learning (RL) з алгоритмом Proximal Policy Optimization (PPO).

3. Інструкційне донавчання (Instruction Tuning). Це етап, коли модель навчається виконувати конкретні інструкції (наприклад, “поясни простими словами” або “перефразуй у науковому стилі”). Для цього використовують пари “запит → бажана відповідь”, створені людьми або іншими моделями. Мета цього єтапу – зробити модель керованою та зрозумілою у діалозі.

4. Мультимодальне донавчання (у новіших версіях, зокрема GPT-4 і GPT-5). Модель навчається працювати не лише з текстом, а й із зображеннями, звуком, кодом, відео тощо. Цей процес відбувається за допомогою спеціальних архітектур типу мультимодальних трансформерів (Multimodal Transformers) [1].

Переваги моделі штучного інтелекту GPT-5 від OpenAI:

1. Розробляється для розуміння та створення текстових відповідей стилістично-подібних до мовлення людини, включаючи складні запити, інтерпретацію контексту та генерацію відповідей, які відповідають контексту. Ця здатність розуміти природну мову сприяє більш інтуїтивній та інтерактивній комунікації.

2. Навчається за допомогою великих обсягів текстових даних з мережі Інтернет, що дозволяє йому мати доступ до різноманітних знань та інформації. Він може відповідати на запитання, пояснювати концепції та надавати розуміння різних тем. Наявність рейтингової оцінки конкретного блоку інформації в мережі інтернет реальними користувачами (система лайк-дизлайк або шкала баллів) додатково дає змогу визначити системі штучного інтелекту ступінь її актуальності та правдивості, а також полегшити збирання та подання статистичних даних згідно запиту користувача.

3. Має можливість надавати швидкі відповіді в режимі реального часу, що допомагає користувачам отримувати миттєві відгуки, вказівки та пропозиції. Це підвищує продуктивність і зменшує час очікування.

4. Має унікальну масштабованість і доступність як модель штучного інтелекту. Вона може ефективно керувати численними взаємодіями одночасно, демонструючи свою відмінну масштабованість. Це дозволяє обслуговувати велику кількість користувачів одночасно, забезпечуючи безперебійну доступність і швидкість реагування навіть у періоди підвищеного навантаження на систему.

5. Дозволяє розуміти та створювати текст у багатьох мовах, що робить його доступним для людей з усього світу. Ця можливість допомагає подолати мовні перешкоди та сприяє ефективному спілкуванню та підтримці у різних мовних контекстах.

6. Пропонує допомогу у складних завданнях, пропонуючи вказівки, пояснення та пропозиції щодо написання коду. Це може бути особливо корисним у сфері розробки програмного забезпечення, допомагаючи в таких завданнях, як

налагодження коду, надання допомоги щодо синтаксису та пропонування пояснень для алгоритмів.

7. Сприяє співпраці та обміну знаннями, створюючи можливості для взаємодії у спільнотах. Користувачі можуть брати участь у обговореннях, ділитися своїми думками та вчитися один від одного, що сприяє створенню атмосфери співпраці та підтримки.

Хоча розробка OpenAI має безліч переваг, він також має свої обмеження. Оскільки модель використовує шаблони, отримані з навчальних даних, іноді вона може генерувати неправильні або нелогічні відповіді. Це особливо важливо враховувати в ситуаціях, де потрібна точна та достовірна інформація. Користувачам рекомендується бути обережними та перевіряти інформацію, особливо в разі потреби прийняття важливих завдань. Самостійна перевірка та аналіз наданого контенту перед використанням може допомогти уникнути можливих непорозумінь або помилкових рішень.

Обмеження OpenAI:

1. Відсутність розуміння реального світу на рівні людської свідомості – відповіді генеративного інтелекту OpenAI ґрунтуються на шаблонах, напрацьованих у ході навчання, і не мають глибокого розуміння реального світу або здорового глузду. Це може вести до правдоподібних, але не точних або нелогічних відповідей, особливо на нечітко сформульовані або складні запитання.

2. Чутливість до формулювання вхідного запиту – інтелектуальна система може реагувати чутливо на навіть незначні зміни у вхідному запитанні чи підказці. Представлення одного й того ж запитання різними способами може призвести до різних відповідей, що може стати проблемою при спробі отримати послідовну або однозначну відповідь.

3. Обмеженість взаємодії – на відміну від людської розмови OpenAI не може задавати додаткові та наводящі запитання у відповідь, щоб конкретизувати окремі питання або запитати стосовно додаткової інформації. Здатність системи надавати точні відповіді може бути обмежена, оскільки вона покладаєтеся виключно на інформацію, надану в запиті чи підказці.

4. Можливість створювати впевнені відповіді, які насправді можуть бути неточними. Це підкреслює важливість самостійної перевірки будь-якої важливої інформації, особливо в темах, що потребують особливої уваги або представляють собою суспільно чи політично чутливу інформацію.

5. Вразливість до упередженого та образливого вмісту - система перевіряє дані, доступні в мережі Інтернет, які можуть містити упереджену або неполіткоректну лексику. Незважаючи на спроби поліпшити такі проблеми під час навчання, модель все ще може демонструвати упереджену поведінку або відповідати на недопустимі з точки зору моралі запити. Для вирішення цих проблем проводяться постійні дослідження та розробки.

6. Труднощі зі складним мисленням – штучний інтелект OpenAI генерує відповіді на основі шаблонів та отриманих даних в процесі навчання, але може мати проблеми зі складним мисленням і багатоетапним вирішенням проблем. Може давати неповні або неточні відповіді на складні запитання, які потребують глибокого розуміння та логічного обґрунтування.

7. Обмежене розуміння контексту – відповіді ґрунтуються переважно на безпосередньому контексті, наданому під час запиту. Він може не мати доступу до великої історії розмов або контекстної інформації, що може обмежити його здатність підтримувати послідовність і безперервність у тривалих дискусіях.

8. Етичні міркування - мовна модель штучного інтелекту OpenAI може створити текст, який може викликати певний етичний дискомфорт у деяких верств населення, наприклад: сприяння дезінформації, участь у шкідливій діяльності або сприяння зловмисним намірам по причині того что у мережі інтернет міститься не тільки достовірна інформація а і помилкова, висловлена тими хто погано розбирається у суті питання, навмисна дезінформація, химерні теорії змов та інші сумнівні дані. Зменшення цих ризиків вимагає відповідального використання, модерації вмісту та дотримання етичних принципів [1].

Аналізуючи функціональні можливості систем генеративного штучного інтелекту, розроблених OpenAI, постає необхідність в створені власного застосунку з модулем взаємодії з штучним інтелектом (GPT-5) та власними

розробленими формулами промтів для створення можливості більш ефективного у часовому вимірі програмування природньою мовою (NLP).

Очікувані результати розробки підсистеми:

1. Підвищення продуктивності: можливість використовувати природну мову (українську) для взаємодії з системою без постійного перекладу власної звичної мови на іноземну що робитиме процес написання коду швидшим та зручнішим для україномовних користувачів. Це дозволить ефективніше приймати рішення щодо виконання завдань, швидше вносити необхідні запити та прискорити процес написання якісного програмного коду.

2. Зниження вимог до кваліфікації: для людей, які не є професійними програмістами, можливість використовувати природну мову для написання коду може знизити поріг входження у світ програмування. Це сприяє розвитку та залученню нових користувачів у сферу програмування та підвищенню рівня їх професійних навичок на практиці.

3. Зменшення помилок: використання природної мови для взаємодії з системою допоможе мінімізувати кількість помилок у програмному коді, оскільки користувачі будуть використовувати готові шаблони промтів для написання коду.

4. Автоматизація рутинних завдань: застосунок зможе автоматизувати рутинні та повторювані завдання в програмуванні, що дозволить програмістам сконцентруватись на більш складних і творчих аспектах розробки програм.

5. Відкриття нових можливостей: створення застосунку з взаємодією з генеративним інтелектом для написання коду природньою мовою може відкрити нові можливості для розвитку та вдосконалення програмного забезпечення, а також сприяти інноваціям у сфері розробки програмного забезпечення.

Основною вимогою до розроблюваного програмного забезпечення є покращити програмування природньою мовою (українською) за допомогою шаблонів промтів з використанням модулю взаємодії GPT-5 та зменшити витрати часу на виконання завдань. У процесі обробки природньої мови мають видаватися результати які будуть відповідати вимогам користувача у вигляді програмного коду.

Для досягнення поставленої задачі мають входити наступні функії:

1. Можливість обрання шаблону промту “Постановка задачі (PSP)”.

2. Можливість обрання шаблону промту “Архітектура програми (PAP)”.

3. Можливість обрання шаблону промту “Логіка програми (PLP)”.

4. Можливість обрання мови програмування: Python, C++, Java.

5. Видання готового коду програмістові по наданому промту за рахунок взаємодії зі штучним інтелектом.

2.3 Мета створення

Метою створення даного програмного забезпечення є підвищення продуктивності отримання програмного коду та зниження витрат часу за рахунок генерації програмного коду штучним інтелектом на основі промтів, що задаються природньою (українською) мовою з використанням OpenAI GPT-5. Це означає, що система буде аналізувати запит користувача наданий природньою мовою (попередньо надаючи шаблони промтів для ефективної взаємодії) та надавати відповідь у вигляді програмного коду.

Основною причиною вибору такої мети є постійне зростання складності програмного забезпечення та потреба у виконанні інтенсивних обчислювальних завдань на швидкісних та ефективних системах.

2.4 Очікувані результати

Прогнозовані результати після розробки програмного забезпечення:

1. Швидке та правильне надання вихідного коду програми. Програма повинна правильно аналізувати наданий промт користувача і надавати відповідний програмний код згідно з запитом користувача. Результатом має бути код програми, готовий до компіляції.

2. Покращена продуктивність взаємодії за рахунок наданої моделі шаблонів промтів під час запиту в програмі. Очікується, що програма забезпечить зменшення тчасових витрат за рахунок надання шаблону промту, який необхідний для отримання максимально точної відповіді.

3. Відповідність вимогам та зручний інтерфейс користувача. Очікується, що програма буде мати зрозумілий та зручний графічний інтерфейс користувача, який дозволить легко використовувати програму та візуалізувати результати отриманого коду.

2.5 Функціональні вимоги

2.5.1 Вимоги до функцій

Програмне забезпечення для перетворення природньої мови у програмний код має реалізовувати такі основні функції:

1. Забезпечення необхідними шаблонами промтів;

2. Можливість обрання мови програмування;

3. Генерація програмного коду по заданим параметрам користувача, готового до компіляції та виконання;

Відповідно до функцій на програмне забезпечення покладаються наступні функціональні вимоги (рис. 2.2):

1. Надання необхідних шаблонів промтів для кращої взаємодії з штучним інтелектом OpenAI;

2. Надання можливості у виборі мови програмування: Python, C++, Java;

3. Надання можливості у програмуванні природньою мовою за рахунок взаємодії з системами OpenAI;

4. Генерація коду програми, готового до компіляції та виконання.

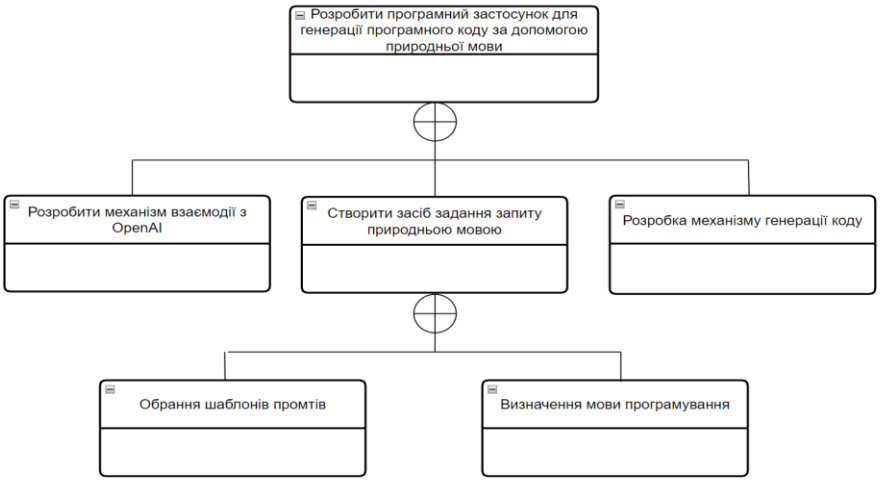


Рисунок 2.2 – Діаграма вимог

На основі визначених вимог створена діаграма варіантів використання (рис. 2.3).

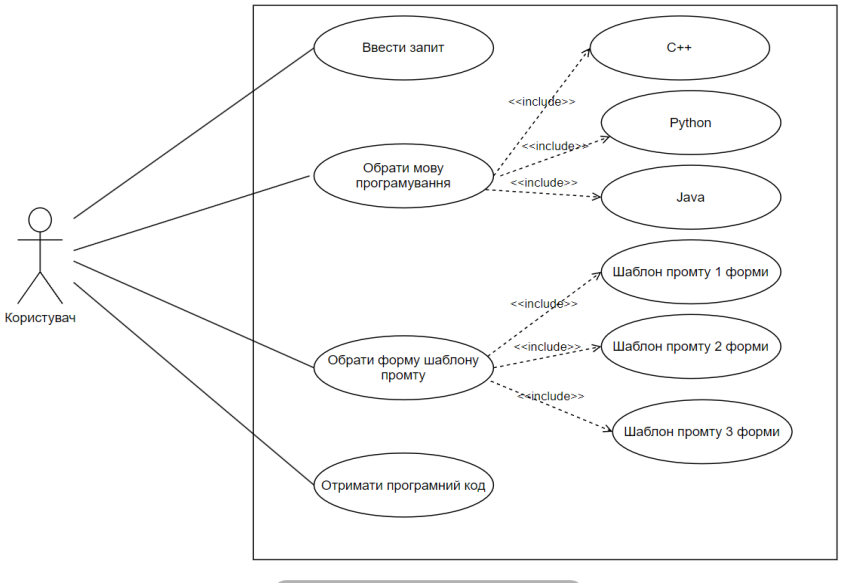


Рисунок 2.3 – Діаграма варіантів використання

Для опису вимог до програмного застосунку можна проілюструвати процеси за допомогою графічної нотації IDEF0 (рис. 2.4).

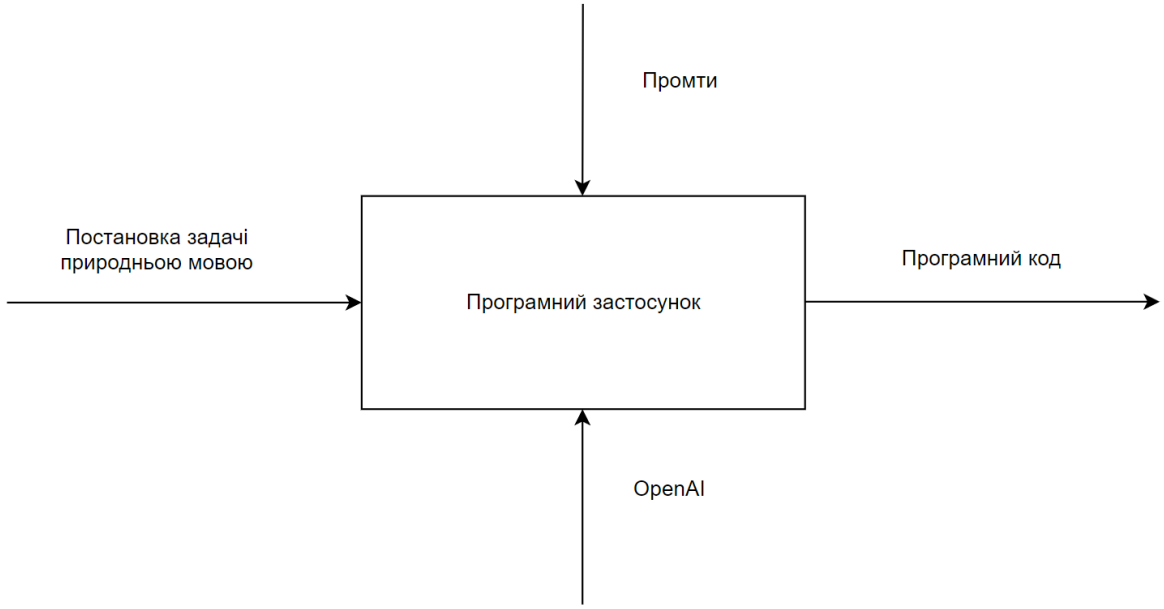


Рисунок 2.4 – Концептуальна модель застосунку

Кожна підсистема повинна виконувати певний набір функцій відповідно до виначених вимог.

Вимоги до функцій визначають, які завдання підсистема повинна виконувати та яким чином ці завдання мають бути реалізовані. В процесі розробки підсистеми важливо враховувати всі вимоги до функцій, які описані далі, щоб забезпечити правильну роботу програмного забезпечення та задовольнити потреби користувачів. Вимоги до функцій визначають функціональність системи і відображають, як система має працювати в реальних умовах. Вони є основою для подальшого проектування, розробки та тестування системи.

2.5.2 Вимоги до взаємодії з системою штучного інтелекту OpenAI.

Для повноцінного функціонування розроблювана підсистема має відповідати наступним вимогам:

1. Система повинна мати підтримку однієї з мов програмування, яка підтримується системою OpenAI.

2. Мережеві вимоги: забезпечити доступ до інтернету для відправки запитів до API серверів OpenAI. Проксі-сервери або брандмауери не повинні блокувати запити до домену api.openai.com.

3. Вимоги безпеки: використовувати HTTPS для всіх запитів до OpenAI та захисту даних під час передачі.

4. Управління доступом: обмежити доступ до API ключів тільки для авторизованих користувачів та процесів.

Вказані вимоги допоможуть забезпечити ефективну та зручну роботу з моделями OpenAI та завантаження вихідного коду програми.

2.5.3 Вимоги до задання промтів користувачами

Одною з найважливіших частин підсистеми є функція використання промтів під конкретне завдання що значно підвищує шанси на отримання більш якісних та релевантних результатів під час взаємодії з мовною моделлю. Для цього необхідно дотримання деяких вимог щодо побудови бази необхідних промтів:

1. Промт повинен бути чітким та конкретним, щоб модель точно розуміла, що від неї очікується. Необхідно уникати двозначних або розпливчастих формулювань.

2. Задання чіткого та цілісного контексту для розуміння завдання. Якщо промт занадто короткий або складається з нечітких узагальних формулювань, модель може не надати релевантної відповіді.

3. Використання наданих шаблонів промтів для структурування інформації у відповідному запиті, якщо завдання складне, його доцільно розбити на підпункти або вказати етапи виконання що-б уникнути узагальнених або неповних результатів.

Виконуючи дані вимоги можливо отримати більш точну відповідь, яка задовольнить потреби користувача.

2.5.4 Вимоги та рекомендації щодо використання налаштування температури під час задання промту

Температура (temperature) в контексті моделей GPT (Generative Pre-trained Transformer) — це параметр, який контролює ступінь випадковості або непередбачуваності відповідей, що генеруються моделлю. Цей параметр впливає на розподіл ймовірностей під час генерації коду штучним інтелектом наступним чином:

1. Налаштування “Температура” може приймати значення від 0 до 1. Вищі значення роблять відповіді більш випадковими, а нижчі — більш передбачуваними. Термін “температура” в контексті моделей OpenAI описує ступінь випадковості або впевненості, з якою модель обирає наступний токен або прогнозує вихідні дані. Це параметр, який регулює, наскільки близькою буде модель у виборі своїх прогнозів.

2. Висока температура (близько до 1): використовується для виконання більш узагальнених задач наприклад таких як вибір варіанту структури коду де остаточне рішення має прийняти розробник, це дозволить розглянути декілька запропонованих штучним інтелектом варіантів і вибрати більш підходящий виходячи з власних цілей та вподобань.

3. Низька температура (близько до 0): використовується для більш передбачуваних та точних відповідей таких як програмний код який має бути максимально якісним та функціональним.

2.5.5 Вимоги до обрання мови програмування

1. Користувач має можливість обрати одну с мов програмування, які надані програмою (Python, C++, Java).

2. Забороняється використання одразу двох мов програмування під час задання промту.

2.6 Нефункціональні вимоги

Мова програмування: програмне забезпечення має бути розроблене з використанням мови програмування Python.

Платформа: програмне забезпечення має бути сумісне з операційними системами сімейства Windows.

Швидкодія: система повинна забезпечувати ефективну обробку великих обсягів вихідного коду програми з мінімальними затратами апаратних ресурсів та мінімальною витратою часу на виконання операцій.

Масштабованість: система повинна бути масштабованою і здатною працювати з різними розмірами вихідного коду програми.

Програмне забезпечення повинно мати графічний інтерфейс користувача, який дозволяє:

1. Запустити Web-інтерфейс програми.

2. Запустити процес взаємодії з системами OpenAI.

3. Відобразити вихідний результат у вигляді програмного коду.

Для реалізації програмного забезпечення рекомендується використовувати наступні технології:

1. Мова програмування Python;

2. Бібліотека: Flask;

3. Бібліотека: request;

4. Бібліотека: render_template.

Програмне забезпечення повинно бути перевірене та протестоване для забезпечення відповідності вимогам.

Висновок до розділу 2

Під час дослідження впливу використання технологій штучного інтелекту на розробку програмного забезпечення було виділено основний показник за яким можна виміряти ефект, який зумовлює використання штучного інтелекту для розробки та написання програмного продукту. Оптимізація часових витрат на розробку програмного забезпечення – це один з найважливіших показників, який

впливає не лише на продуктивність команди розробників, а і на економічні показники підприємства-розробника вцілому.

Під час дослідження виявлено значний вплив використання генеративного штучного інтелекту на швидкість виконання задач, на відміну від традиційного написання програмного коду, де розробник самостійно виконує всі етапи завдання починаючи від проектування структури закінчуючи тестуванням готового програмного забезпечення.

Вибір такого типу штучного інтелекту як генеративний зумовлено його основними характеристиками, подібними до людських таких як творча здатність, навчання на великих об’ємах наявної інформації та адаптивність, що дозволяє з впевненістю зробити висновок, що на даний момент цей тип штучного інтелекту є найпродуктивнішим.

Зниження часових витрат позитивно впливає на кількість виконаних проектів за визначений проміжок часу, що підвищує прибуток як підприємства так і окремого розробника. Автоматизація написання програмного коду та його тестування також знижує рівень втомленості та стресу розробників, звільняє більше часу на прийняття рішень, та проектування візуальної складової програмного продукту як інтерфейс та анімація.

В цьому розділі також були розглянуті основні аспекти і вимоги до розроблюваного програмного забезпечення. Ця система має на меті автоматизувати процес перетворення природньої мови в програмний код, за допомогою використання шаблонів промтів для оптимізації часових витрат.

Основним призначенням розроблюваної системи є програмування природньою мовою за рахунок взаємодії з штучним інтелектом. Це дозволить користувачам писати код на природній мові, який буде автоматично трансформуватися в робочий програмний код за допомогою штучного інтелекту. Основним призначенням такої системи є спрощення процесу програмування, зниження порогу входу для новачків та підвищення продуктивності досвідчених програмістів. Система взаємодії з OpenAI буде виконувати роль "перекладача" з природної мови в програмний код.

Головними функціями системи є генерація вихідного коду програми, готового до виконання в програмному середовищі, можливість налаштування параметрів генерації, таких як температура та максимальна кількість токенів, та можливість використання шаблонів промтів.

Система має ряд вимог, таких як відповідність стандартам програмування, вимогами взаємодії OpenAI GPT-5, вимоги стосовно викристання шаблонів промтів. Виконання цих вимог дозволить забезпечити коректність та ефективність процесу генерації програмного коду.

Система має знизити часові витрати за рнахунок збільшення продуктивності та автоматизації процесу генерації вихідного коду програмного забезпечення, автоматизації рутинних завдань, допомоги новачкам у програмуванні, надаючи можливість писати код на природній мові, використання системи як навчального інструменту для розуміння синтаксису та логіки програмування.

3 РОЗРОБКА СИСТЕМИ

3.1 Розробка архітектури

Як було визначено у минулому розділі, програмний застосунок складатиметься з декількох функціональних модулів (рис. 3.1):

1. Модуль вибору форми шаблону промту;
2. Модуль вибору мови програмування вихідного коду;
3. Модуль перетворення вихідного коду з взаємодією з OpenAI;
4. Модуль генерації коду.

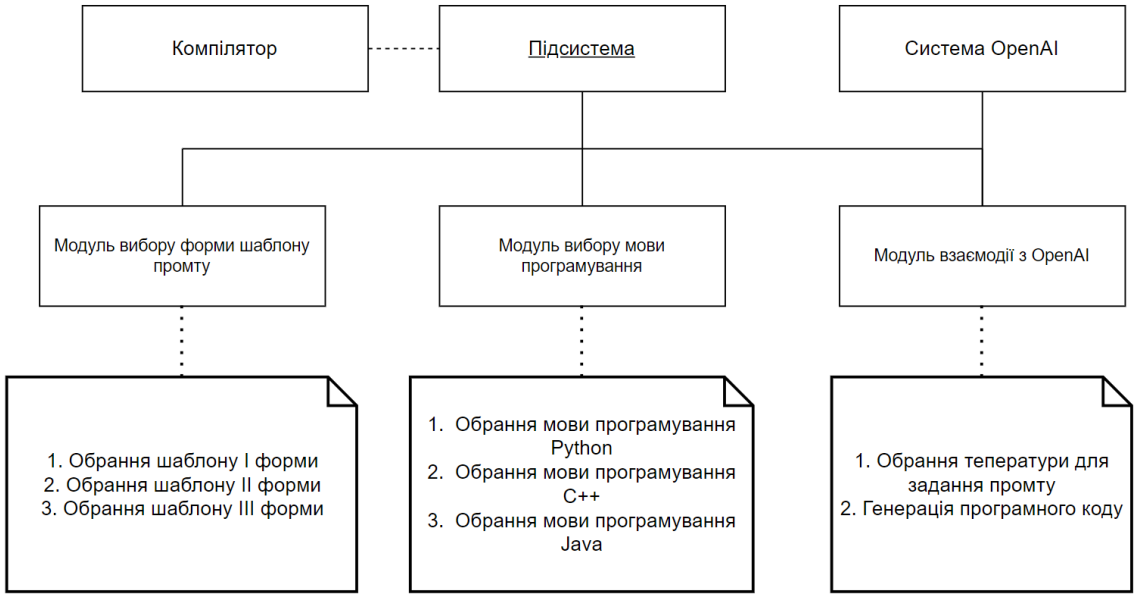


Рисунок 3.1 – Загальна структура системи

Процес перетворення природньої мови в програмний код здійснюється відповідно в декілька етапів і описується за допомогою алгоритмів. Для початку задіюється алгоритм попередньої обробки природної мови - такі методи як: токенізація, синтаксичний аналіз, семантичне моделювання допомогають “зрозуміти” текст користувача. Далі задіюються алгоритми перетворення тексту в абстрактне подання, накшталт Sec2Sec моделі що навчаються відображати послідовність слів (запит) у послідовність команд (код) – на цьому етапі система перетворює природну мову в проміжне представлення, яке ближче до коду.

Наступним кроком задіюються алгоритми нейронного кодування, такі як Transformer (що використовує механізм self-atttntion для визначення контексту між словами) та Pretrained Language models, які попередньо навчані на коді та тексті, що дозволяє згенерувати зв’язний код згідно змісту запиту що надається користувачем. Після генерації код проходить перевірку відповідними системами для виявлення потенційних помилок (Static code analysis) та відпрацьовує тестовий запуск, разом з цім модель здійснює навчання на правильних і неправильних результатах постійно вдосконалюючи результат.

3.2 Інструменти реалізації

Програма створена на мові програмування Python з використанням середовища розробки PyCharm Community 2023.3.2, та текстовому редакторі для веб-розробки Brackets.

Вибір мови обумовлений об'єктно-орієнтованим підходом до написання програми. На даний момент мова Python є одною з найпопулярніших, зручних і повноцінних серед об'єктно-орієнтованих мов програмування.

У процесі проектування системи, для взаємодії з моделлю GPT-5, розглядалися різні платформи та інструменти для забезпечення її функціональності, стабільності та масштабованості. Оглядаючи можливі рішення взаємодії GitHub Copilot, Amazon CodeWhisperer та OpenAI вибір припав на останню, через те що система OpenAI демонструє надзвичайно високий рівень розуміння та генерації природної мови, забезпечуючи точні та релевантні відповіді, GPT-5 надає добре задокументований API, що полегшує процес інтеграції з іншими системами та додатками. Можливість тонкого налаштування параметрів запиту, таких як температура генерації, дозволяє налаштувати модель під конкретні потреби, OpenAI активно розвиває свої моделі, регулярно випускаючи оновлення та оптимізує функціонування системи, що гарантує їхню актуальність і високу якість роботи, надійна інфраструктура та підтримка забезпечують безперебійний доступ до сервісів навіть при високих навантаженнях.

Для реалізації призначеного для користувача інтерфейсу було вирішено використовувати веб-інтерфейс, так як він простий для сприйняття і інтуїтивно зрозумілий. Для реалізації інтерфейсу використовується Flask. Це легкий мікрофреймворк для Python, який дозволяє швидко створювати, тестувати та розгортати веб-додатки для створення серверної частини розроблюваного веб-додатку, обробки HTTP-запитів та відправлення відповідей на запити. Для обробки запитів (requests), призначених для взаємодії з зовнішнім API (через HTTP-запити), використовується бібліотека для взаємодії з API моделі OpenAI, з метою отримання генерованої відповіді на запит користувача. Функція render_template використовується для відображення HTML-шаблонів.

3.3 Розробка алгоритмів

3.3.1 Загальний алгоритм роботи системи

Розробка алгоритму є ключовим етапом розробки програмного забезпечення системи перетворення природньої мови в програмний код. Розвиток сучасних програмних систем вимагає ефективного використання обчислювальних ресурсів.

Ініціалізація сервера та імпорт необхідних модулів: спочатку імпортується Flask для створення веб-сервера та необхідні модулі, такі як request, render_template для обробки HTTP-запитів і роботи з шаблонами HTML.

Функція send_gpt для взаємодії з моделлю OpenAI. Підготовка запиту: формується JSON-об'єкт з питанням користувача та іншими параметрами, такими як модель OpenAI, максимальна кількість токенів та температура генерації.

Взаємодія з API здійснюється POST-запитом до API моделі OpenAI за допомогою бібліотеки requests та переданими параметрами. Обробка відповіді: отримана відповідь у форматі JSON обробляється, із неї витягується згенероване повідомлення від моделі.

Маршрут для обробки HTTP-запитів: при GET-запиті на головну сторінку ("/") відображається HTML-шаблон форми для задання промту та вибору температури генерації та наявних шаблонів промтів та обрання необхідної мови

програмування. При POST-запиті обробляються дані з форми, викликається функція send_gpt для генерації відповіді, та результат виводиться на сторінці.

Запуск сервера: сервер Flask запускається на локальному хості з відлагодженням (debug=True) і працює на порту 80.

Схематично це можна зобразити за допомогою діаграми послідовності UML (рис. 3.2).

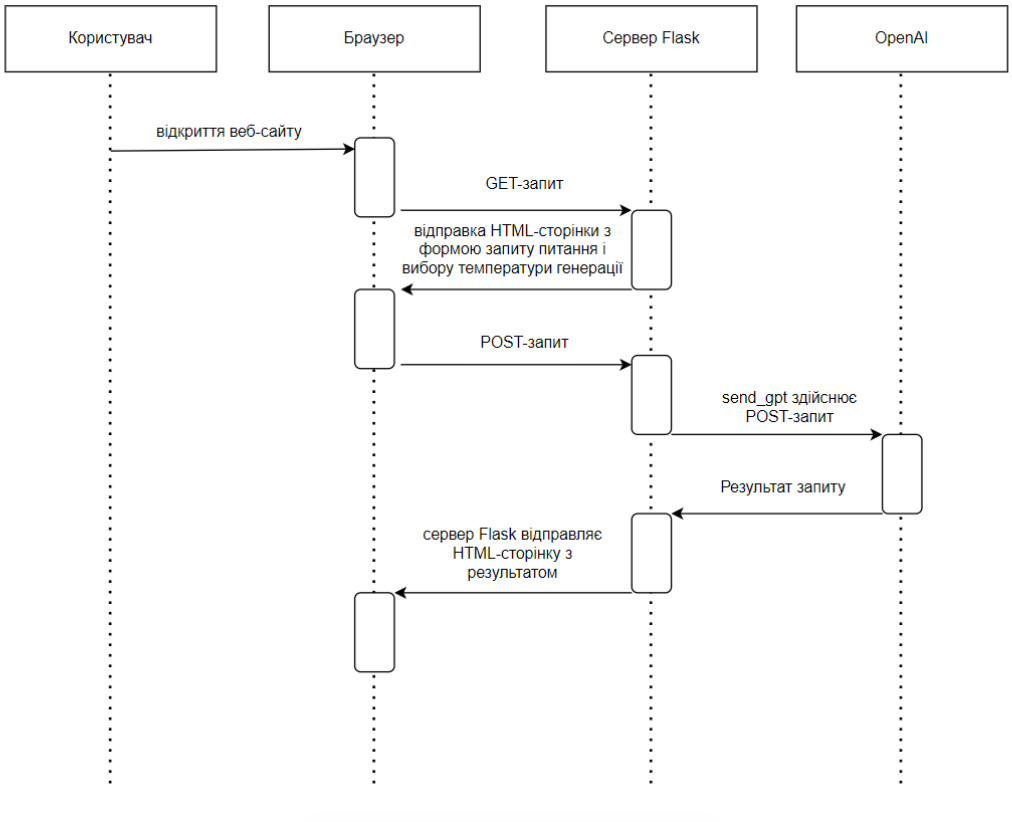


Рисунок 3.2 – Діаграма послідовності UML

3.3.2 Алгоритм задання промту

Алгоритм задання промту природньою мовою наступний:

1. Вибрати мову програмування, для якої потрібно розробити програмний код;

2. Обрання шаблону промту;

3. Задання промту згідно рекомендацій;

4. Обрання параметрів температури для промту.

3.3.3 Структура використання шаблонів промтів

1. Шаблон постановки задачі (PSP)

Domain: [Опишіть предметну область задачі]

Conditions:

Вхідні дані: [Опишіть вхідні дані]

Обробка: [Опишіть процес обробки]

Вихідні результати: [Опишіть очікувані вихідні результати]

2. Шаблон архітектури програми (PAP)

Lib: [Список бібліотек, які потрібно використати]

Classes:

TaskManager: клас для управління завданнями, містить методи для додавання, редагування, видалення та отримання завдань.

Task: клас для представлення окремого завдання, містить атрибути як-от назва, опис, дата завершення, пріоритет.

Functions:

add_task(title, description, due_date, priority): функція для додавання нового завдання.

edit_task(task_id, title, description, due_date, priority): функція для редагування існуючого завдання.

delete_task(task_id): функція для видалення завдання.

get_tasks(filter_by, sort_by): функція для отримання списку завдань з можливістю фільтрації та сортування.

DataStructures:

Список (List) для зберігання завдань.

Словник (Dictionary) для зберігання налаштувань фільтрації та сортування.

3. Шаблон логіки програми (PLP)

Опис алгоритму:

1. [Крок 1: Опис]

2. [Крок 2: Опис]

3. [Крок 3: Опис]

4. [Крок 4: Опис]

3.3.4 Алгоритм отримання програмного коду

Алгоритм отримання програмного коду:

1. Отримати отримання веб-сторінки з формою для заповнення;
2. Надіслати промт використовуючи кнопку “Надіслати”;
3. Отримати код програми у відповідному полі;
4. Протестувати код за допомогою зовнішнього компілятора;
5. Отримати результати тестування.

Простота побудови алгоритму роботи підсистеми зумовлює незначне навантаження на апаратну частину серверу та досить високу стійкість до помилок чи відмови роботи застосунку за умов його використання великою кількістю користувачів одночасно.

Якщо система отримує зворотній зв’язок (Рис. 3.3) (наприклад, код не компілюється або не проходить тести), вона може покращити свої параметри через Reinforsment learning – оптимізація дій на основі результатів. Отримання негативної відповіді від підсистеми слугуватиме підказкою користувачу що потрібно задати більш конкретні параметри до промту, це в перспективі відвищує рівень навичок поводження з промтами та спонукає до формування чітких команд що покращує розуміння взаємодії з системами штучного інтелекту, принципу їх функціонування та видачі запланованого результату. Це значно підвищує швидкість виконання завдань з написання програмного коду по заздалегідь побудованій структурі.

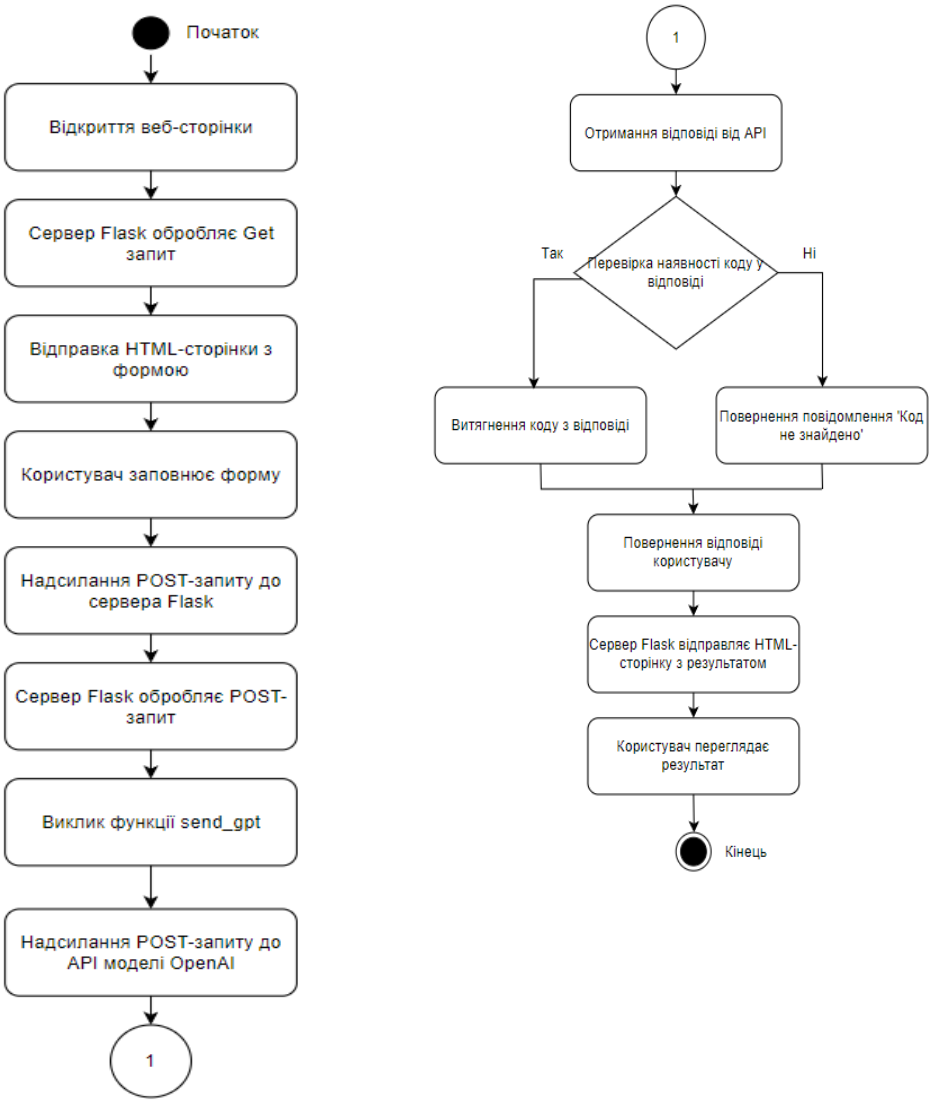


Рисунок 3.3 – Діаграма діяльності UML

3.4 Використання Flask

Для реалізації веб-інтерфейсу у системі було використано фреймворк Flask (рис. 3.4). Це легкий та гнучкий веб-фреймворк для програмування мовою Python, який дозволяє швидко створювати веб-додатки, API та мікросервіси. Його часто називають “мікрофреймворком”, оскільки він не нав’язує певну архітектуру та не містить зайвих модулів за замовчуванням, а лише базовий набір для обробки запитів, маршрутизації та шаблонів. За потрібності його функціонал легко доповнюється сторонніми бібліотеками (авторизації, роботи з базами даних, тощо), вбудований потужний шаблонізатор для динамічної генерації HTML-сторінок Jinja2 забезпечує гарний показник швидкодії веб-додатків. За рахунок

своєї простоти, Flask дозволяє зосередитись на функціональності програми без необхідності розбиратися в складних деталях веб-розробки.

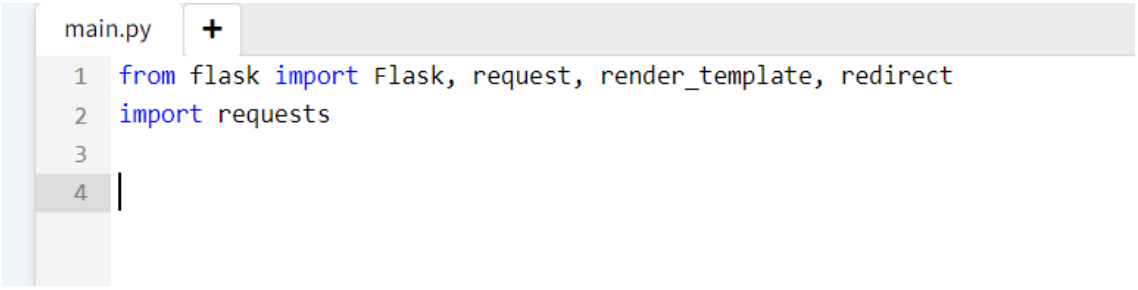


Рисунок 3.4 – Створення Flask-додатку

Наступним завданням є визначте базового маршруту для обробки GET і POST запитів (рис. 3.5) .

Рисунок 3.5 – Визначення базового маршруту

Наступним кроком є створення функції send_gpt для обробки запитів до OpenAI (рис. 3.6)

```python
url = "https://cheapest-gpt-5-turbo-gpt-5-vision-chatgpt-openai-ai-api.p.rapidapi.com/v1/chat/completions"
def send_gpt(prompt, tem):
    try:
        payload = {
            "messages": [
                {
                    "role": "user",
                    "content": prompt
                }
            ],
            "model": "gpt-5-turbo-2025-04-09",
            "max_tokens": 200,
            "temperature": tem
        }
        headers = {
            "content-type": "application/json",
            "X-RapidAPI-Key": "YOUR_RAPIDAPI_KEY",
            "X-RapidAPI-Host": "cheapest-gpt-5-turbo-gpt-5-vision-chatgpt-openai-ai-api.p.rapidapi.com"
        }
        response = requests.post(url, json=payload, headers=headers)
        text = response.json()['choices'][0]['message']['content']
        print(text)
        start_index = text.find("```")
        end_index = text.find("```", start_index + 1)
        if start_index != -1 and end_index != -1:
            code = text[start_index:end_index + 3]  # Включаем три обратные кавычки в конце блока
            return code
        else:
            return "Код не найден."
    except:
        return "Connection Error! Please try again."
```

Рисунок 3.6 – Функція send_gpt

Для запуску сервера Flask додаємо блок в кінці виконаного файлу (рис. 3.7).

```python
if __name__ == '__main__':
    server.run(debug=True, host='0.0.0.0', port=80)
```

Рисунок 3.7 – Функція запуску веб-сервісу

3.5 Опис реалізації

3.5.1 Загальний опис реалізації

Система реалізована на мові програмування Python у вигляді веб-інтерфейсу на основі фреймворку Flask. Дана технологія була обрана через простоту реалізації.

Згідно алгоритмів та обраної структури програми програмний застосунок складається з декількох функцій (рис. 3.8).

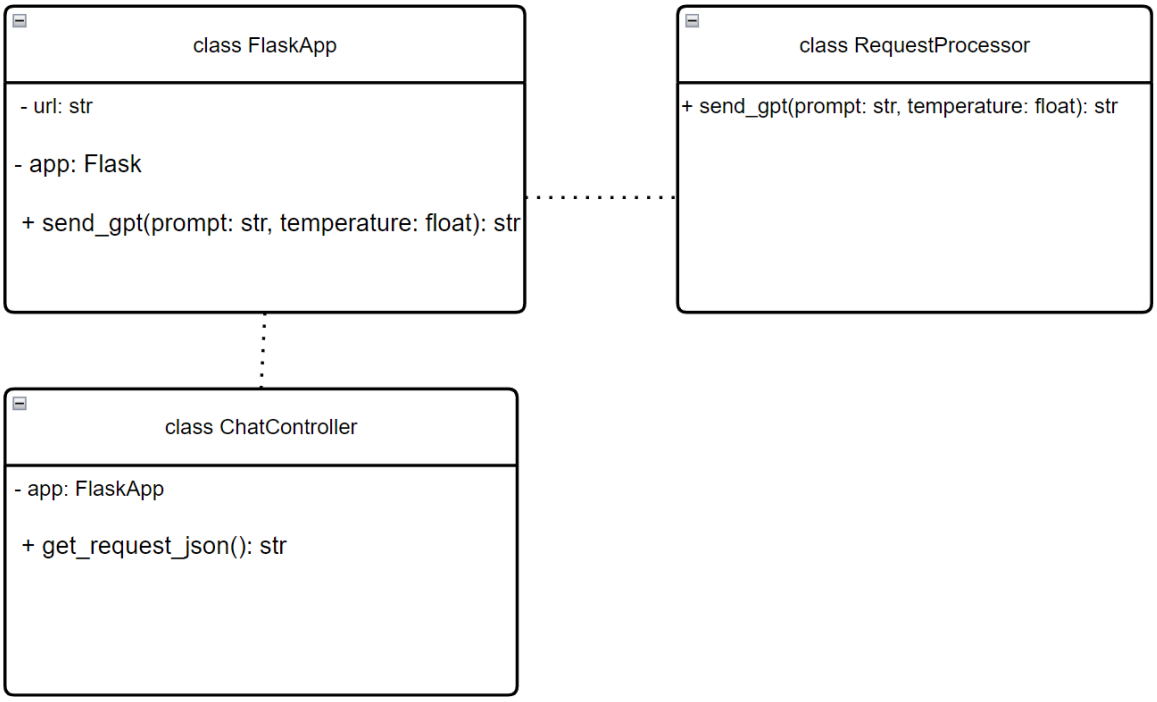


Рисунок 3.8 – Діаграма класів UML

3.5.2 Розробка інтерфейсу

Мета розробки полягає у створенні інтуїтивно зрозумілого та ефективного інтерфейсу користувача. Це значно оптимізує витрату часу на взаємодію розробника з додатком та робить його доступним для розуміння менш досвітченим працівникам.

Інтерфейс повинен бути створений з дотриманням таких принципів:

1. Простота – мінімум зайвих елементів коли кожен компонент має чітке призначення, просте формулювання тексту, кнопок, повідомлень та уникаючи перевантаження кольорами і анімацією.

2. Видимість – найважливіші функції мають бути легкодоступними та помітними що-б не змушувати користувача шукати ключові кнопки.

3. Ефективність – мінімізація кількості дій для досягнення цілі.

Визначивши задачі, які повинна виконувати програма, вимоги, які постають перед програмним застосунком, та функції, розроблена структура (рис. 3.9) та базова модель програмного застосунку (рис. 3.10).

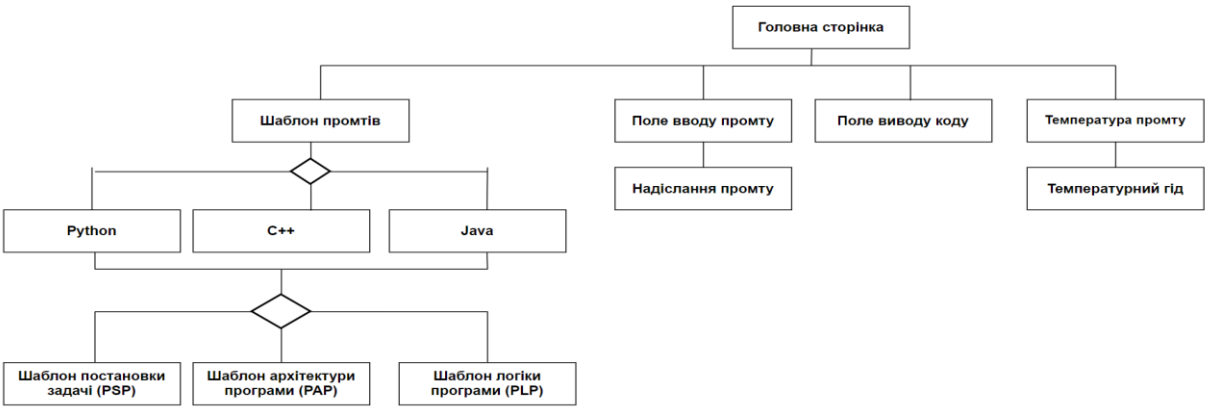


Рисунок 3.9 – Структура інтерфейсу користувача

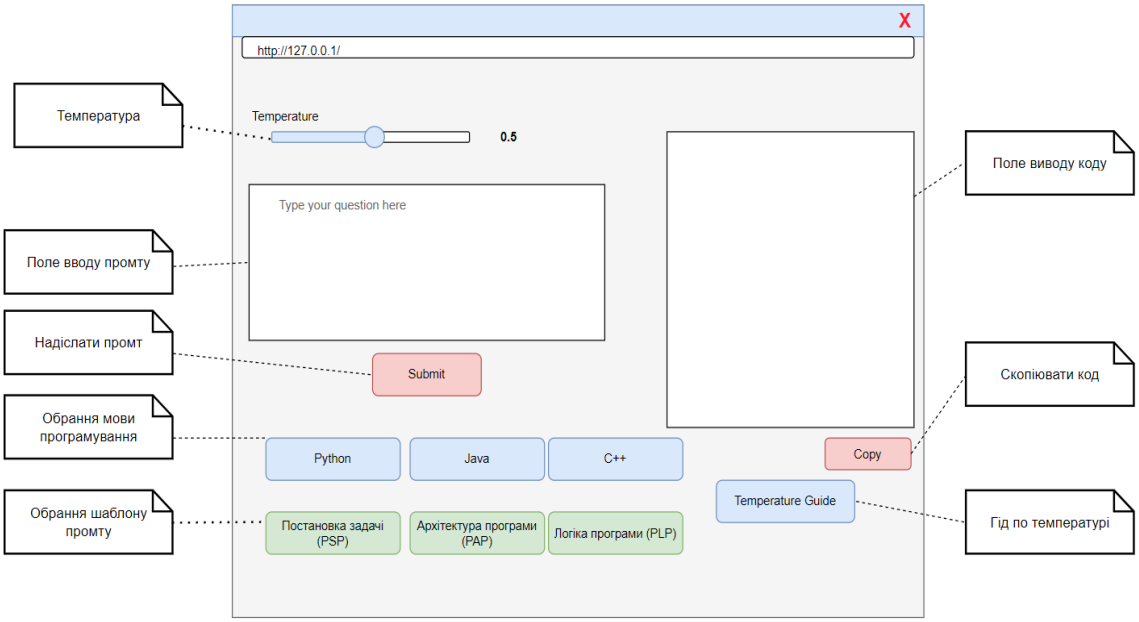


Рисунок 3.10 – Дизайн веб-інтерфейсу

Підвищена гнучкість досягається за рахунок простої реалізації лише необхідних інструментів.

Такий підхід полегшує користувачам розуміння програмного продукту і дозволяє працювати лише з тим функціоналом, який їм потрібен [6].

3.5.3 Вікно вводу промту

Вікно вводу промту (рис. 3.11) та все, що з ним пов’язано, розташовано всередині форми з ідентифікатором ‘questionForm’.

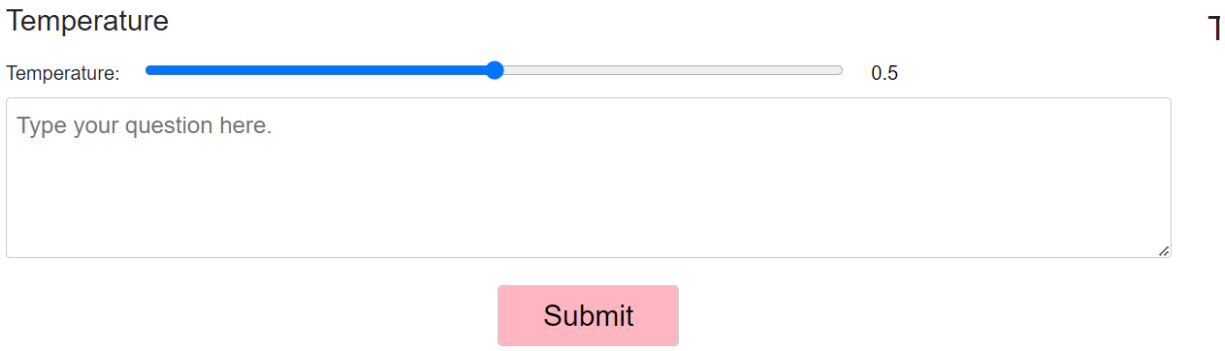


Рисунок 3.11 – Вікно вводу промту

Форма містить текстову область (textarea) для вводу питання (промту) та елемент для вводу температури (рис. 3.12).

```
<form id="questionForm" method="post" onsubmit="submit.disabled=true">
  <label for="temperature">Temperature:</label>
  <input type="range" name="temperature" id="temperature" min="0" max="1" step="0.1" value="0.5"
    onchange="show()">
  <input type="text" id="temperature-value" name="temperature-value" value="0.5">
  <br>
  <textarea name="question" id="question" placeholder="Type your question here." rows="4"></textarea>
  <br>
  <input type="submit" value="Submit" id="submit">
</form>
<div id="loading"><b>Waiting for response...</b></div>
```

Рисунок 3.12 –Текстова область веб-сторінки для вводу промту

Основні функції вікна вводу промту та пов’язаних з ним елементів:

1. Задання власного промту;
2. Обрання температури для промту;
3. Обрання мови програмування;
4. Обрання форми задання промту (PSP, PAP, PLP) [3,4].

Для автоматичного додавання промту в текстову область використовується JavaScript код (Рис 3.13)

main.js

```
<script>
  function addPrompt(promptText) {
    var questionTextArea = document.getElementById("question");
    var currentText = questionTextArea.value;
    if (currentText.trim() !== "") {
      currentText += " ";
    }
    questionTextArea.value = promptText + " " + currentText;
  }

  $("#sidebarToggle").click(function() {
    $(".sidebar").toggle();
    if ($(".sidebar").is(":visible")) {
      $(this).text("Hide Temperature Guide");
    } else {
      $(this).text("Show Temperature Guide");
    }
  });
</script>
```

Рисунок 3.13 – Скрипт для додавання промту в текстову область

У коді вікно вводу промту та все, що з ним пов’язано, включає форму для вводу промту та температури, текстову область для вводу питання, елементи вводу для температури, а також сценарії JavaScript для обробки введених даних та автоматичного додавання тексту до текстової області.

3.5.4 Функція взаємодії з OpenAI GPT-5

Функція ‘send_gpt’ у коді відповідає за взаємодію з OpenAI API. Вона приймає промт від користувача та температуру, відправляє запит до OpenAI API, отримує відповідь та повертає відповідний код або повідомлення (рис. 3.14).

Формування запиту `payload` формується тіло запиту, яке включає повідомлення користувача (prompt), модель (gpt-5-turbo-2025-04-09), максимальну кількість токенів (200) та температуру (tem).

Заголовки запиту ‘headers’ включають тип контенту (json), ключ API (X-RapidAPI-Key) та хост API (X-RapidAPI-Host).

Відправлення запиту використовується метод ‘requests.post’ для відправлення POST-запиту до OpenAI API з відповідним тілом запиту (payload) та заголовками (headers).

main.py

```python
import requests
url = "https://cheapest-gpt-5-turbo-gpt-5-vision-chatgpt-openai-ai-api.p.rapidapi.com/v1/chat/completions
def send_gpt(prompt, tem):
    try:
        # Формування запиту до OpenAI API
        payload = {
            "messages": [
                {
                    "role": "user",
                    "content": prompt
                }
            ],
            "model": "gpt-5-turbo-2025-04-09",
            "max_tokens": 200,
            "temperature": tem
        }
        # Заголовки для запиту
        headers = {
            "content-type": "application/json",
            "X-RapidAPI-Key": "19e283ce09msh757aed4b81e1134p1192d2jsn6e6555cd83ab",
            "X-RapidAPI-Host": "cheapest-gpt-5-turbo-gpt-5-vision-chatgpt-openai-ai-api.p.rapidapi.com"
        }
        # Відправлення запиту до API
        response = requests.post(url, json=payload, headers=headers)
        # Обробка відповіді від API
        text = response.json()['choices'][0]['message']['content']
        print(text)
        # Пошук та виділення коду у відповіді
        start_index = text.find("```")
        end_index = text.find("```", start_index + 1)

        if start_index != -1 and end_index != -1:
            code = text[start_index:end_index + 3]  # Включаємо три зворотні лапки в кінці блоку
            return code
        else:
            return "Код не знайдено."
    except:
        # Обробка помилок з'єднання
        return "Помилка з'єднання! Спробуйте ще раз."

```

Рисунок 3.14 – Реалізація взаємодії з OpenAI GPT-5

Обробка відповіді. Відповідь парсується для отримання тексту повідомлення (response.json()['choices'][0]['message']['content']).

Шукається блок коду у відповіді, обмежений трьома зворотніми лапками (```). Якщо такий блок знайдено, він повертається; якщо ні — повертається повідомлення "Код не знайдено".

Взаємодія застосунку з сервісами штучного інтелекту здійснюється через програмний інтерфейс (API).

Функція створення API ключа забезпечує основний механізм взаємодії з OpenAI API, дозволяючи надсилати промти та отримувати відповіді у вигляді коду або тексту, що потім відображається користувачу (рис. 3.15) [3]. Також API-ключ забезпечує аутентифікацію користувача.

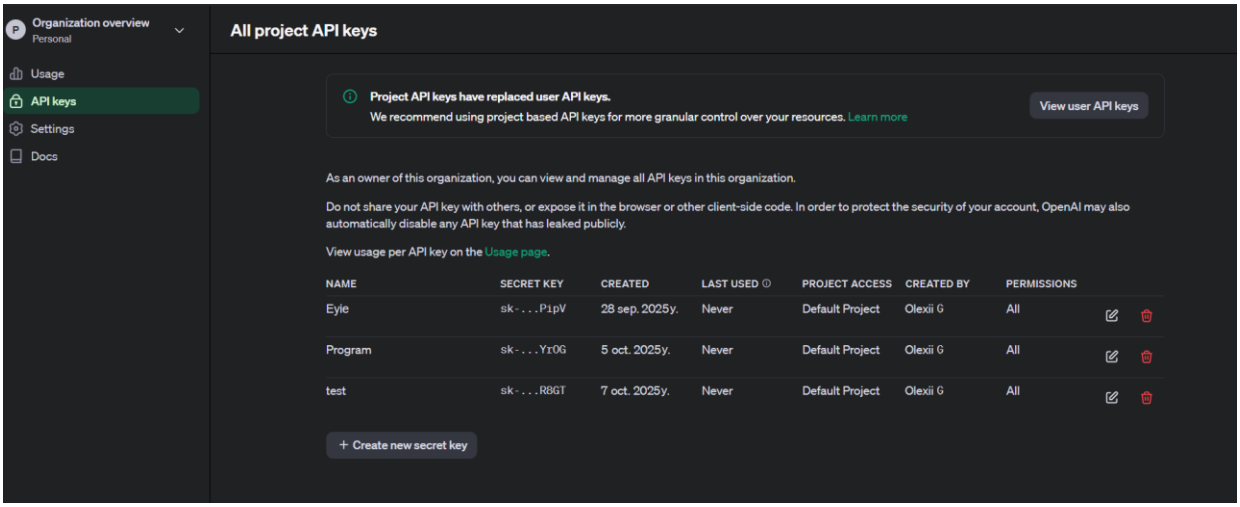


Рисунок 3.15 – Створення API ключа

3.6 Реалізація програмного застосунку

Розроблений програмний застосунок з веб-інтерфейсом демонструє ефективну взаємодію з потужним інструментом штучного інтелекту OpenAI, що дозволяє користувачам отримувати відповіді на запити в зручному та зрозумілому форматі. Основними аспектами реалізації даного проекту є програмування природньою мовою, створення шаблонів промтів, використання Flask для створення веб-сервера, обробка запитів через OpenAI API та відображення результатів у веб-інтерфейсі.

Використання Flask забезпечує легку та гнучку структуру для розробки веб-додатків на Python. Сервер обробляє запити, отримані від користувача, та забезпечує зв’язок з фронтендом. Маршрутизація запитів дозволяє розподілити запити на різні URL-адреси та методи HTTP (GET, POST), що спрощує обробку різних типів запитів і відповідей.

Функція send_gpt відповідає за відправлення запитів до OpenAI API. Вона формує необхідний payload, додає заголовки з API-ключем і надсилає POST-запит

до API. Отримані дані обробляються, щоб виділити потрібну частину тексту. У разі успіху відокремлюється блок коду або повертається текстова відповідь, в іншому випадку - повідомлення про помилку.

Основна сторінка представлена у вигляді HTML-документа, який містить форму для вводу запиту користувача, налаштування температури, вибір мови програмування, шаблони промтів та область для відображення результатів. Додаткові функції, такі як динамічне оновлення температури та додавання попередньо визначених запитів, реалізовані за допомогою JavaScript.

3.7 Порівняння витрат часу з іншими ШІ-орієнтованими підходами

Для оцінки часової ефективності використання даної системи з запрограмованою базою промтів, для різних етапів розробки програмного забезпечення, та іншими необхідними параметрами проведено порівняльний аналіз з звичайним діалоговим використанням систем штучного інтелекту.

Підхід з використанням підсистеми. У цьому випадку розробник взаємодіє з інтерфейсом, який містить заздалегідь підготовлені промти та їх шаблони, структуровані за типом задачі (постановка задачі, архітектура, логіка, функціонал та інші). Додатково система дозволяє одразу додавати цільову мову програмування та рівень конкретизації вихідного результату, що усуває потребу уточнення контексту під час кожного запиту.

Середній час формування запиту та отримання коду значно скорочується, оскільки зменшується кількість ітерацій уточнення.

Знижується когнітивне навантаження на розробника, що сприяє підвищенню продуктивності.

Традиційний підхід (ручний ввід даних). Розробник самостійно формує запити до ШІ-системи, описуючи необхідні параметри, контекст, обмеження та очікуваний результат. Часто цей процес включає кілька циклів уточнення промтів у режимі реального часу.

Часові витрати зростають через необхідність ручного опису вимог і послідовного вдосконалення запитів.

Результати залежать від рівня досвіду розробника у формулюванні ефективних промтів.

У середньому використання підсистеми з попередньо визначеними промтами дозволило скоротити загальні витрати часу на 30–50% залежно від складності проєкту. Основне скорочення часу спостерігається на етапах:

1. Формування запиту;

2. Уточнення технічних параметрів;

3. Перевірки відповідності отриманого коду початковим вимогам.

Використання підсистеми, із заздалегідь підготовленими промтами та вибором мови програмування, забезпечує суттєву оптимізацію процесу взаємодії з ШІ-помічником. Такий підхід є більш ефективним у часовому вимірі та сприяє стандартизації процесу генерації програмного коду. Для задач із високою складністю, нестандартними вимогами або великим контекстом підходить налаштування “температури” ближче до “1”, це дасть команду для генерації замість одного точного – декілька різних варіантів відповідей (накшталт варіанти структури коду), що-б розробник міг вибрати більш підходящий варіант під конкретну задачу з тими перевагами та функціями які розробник вважатиме більш доцільними до використання в його проекті. Це значно спростить процесс прийняття рішень, що також оптимізує часові витрати.

Висновки до розділу 3

Для вирішення поставленої задачі було розроблено програмний застосунок з веб-інтерфейсом, що дозволяє користувачам взаємодіяти з моделлю штучного інтелекту OpenAI для автоматизованого створення коду. Розробка включала кілька важливих етапів та аспектів:

1. Вибір платформи - серед можливих рішень було розглянуто GitHub Copilot, Amazon CodeWhisperer та OpenAI GPT-5. Вибір припав на OpenAI GPT-5 через високу точність, гнучкість і широкий спектр можливостей моделі.

2. Розробка інтерфейсу: для створення зручного та інтуїтивно зрозумілого інтерфейсу було використано Flask – веб-фреймворк на Python, що дозволяє

досить швидко розробити веб-застосунок. Використання Flask забезпечує простоту налаштувань і масштабованість системи.

3. Інтерактивність та налаштування: веб-інтерфейс включає поле для вводу промтів та налаштувань температури, що дозволяє користувачам точно контролювати генерацію коду, вибір мови програмування, шаблони задання промтів. Температура відповіді моделі визначає ступінь креативності відповіді – від детермінованих до більш варіативних і творчих.

4. Веб-застосунок відображає отриманий код або повідомлення про помилку, якщо код не був знайдений. Це забезпечує зворотний зв'язок для користувача, що є ключовим для інтерактивного застосунку.

Таким чином, розроблений програмний застосунок з веб-інтерфейсом успішно вирішує поставлену задачу зменшення витрат часу для створення коду в процесі програмуванні природньою мовою за допомогою моделі штучного інтелекту OpenAI GPT-5, наявних налаштувань для видачі конкретного результату. Це дозволяє користувачам отримувати швидкі та точні відповіді на свої запити.

Дослідження показують, що завдяки використанню підсистеми для автоматизації взаємодії зі штучним інтелектом, швидкість написання програмного коду зростає на 30%, порівняно з прямим використанням генеративної моделі за рахунок заздалегідь наявних продуманих шаблонів промтів, що значно скорочує час на їх формулювання та зменшує шанс наявності логічних помилок і неточностей. Особливо цінне використання підсистеми менш досвітченими розробниками, це значно підвищує швидкість і якість написання програмного коду та виконує освітню функцію – розробник вчиться правильно та достатньо детально формулювати запити, аналізуючи отриманий результат – дізнається та вивчає нові можливості програмного коду, що значно підвищує рівень професійних навичок та збільшує конкурентоспроможність на ринку праці у відповідних галузях.

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Загальні відомості

Програмне забезпечення призначене для програмування природньою мовою, за допомогою систем штучного інтелекту від OpenAI. Воно надає зручний веб-інтерфейс, який дозволяє користувачам вводити запити на генерацію коду, налаштовувати параметри вихідних даних, обирати мову програмування, використовувати шаблони промтів та отримувати результати у реальному часі.

4.2 Вимоги до системи

Для коректної роботи програмного забезпечення необхідно мати:

1. Операційну систему з підтримкою Python 3.7 і вище;
2. Встановлений веб-фреймворк Flask;
3. Доступ до Інтернету для роботи з API OpenAI;
4. Браузер для взаємодії з веб-інтерфейсом.

4.3 Використання

Запуск програми:

1. Відкрийте термінал або командний рядок;
2. Перейдіть до директорії з вихідним кодом застосунку;
3. Запустіть сервер Flask в терміналі за допомогою команди
python app.py;
4. Відкрийте браузер і перейдіть за адресою http://127.0.0.1:80.

При відкритті програмного застосунку на екрані з’явиться Головне вікно програмного застосунку (рис. 4.1).

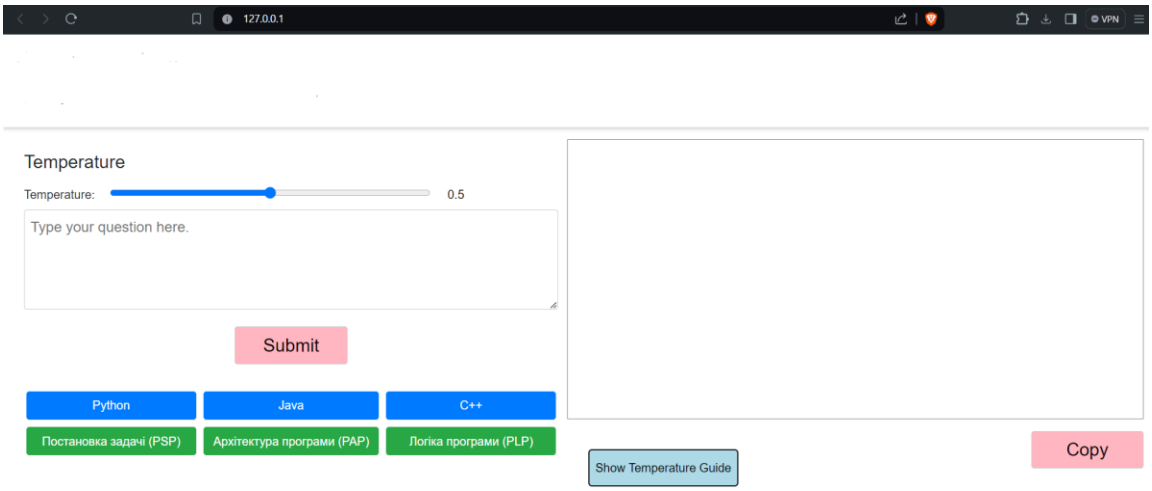


Рисунок 4.1 – Головне вікно програмного застосунку

Головне вікно містить меню та два поля для вводу коду. Для початку перетворення необхідно:

1. Обрати температуру задання промту (рис. 4.2). Контролює випадковість: зниження температури призводить до меншої кількості випадкових результатів. Коли температура наближається до нуля, модель стає детермінованою і повторюваною;

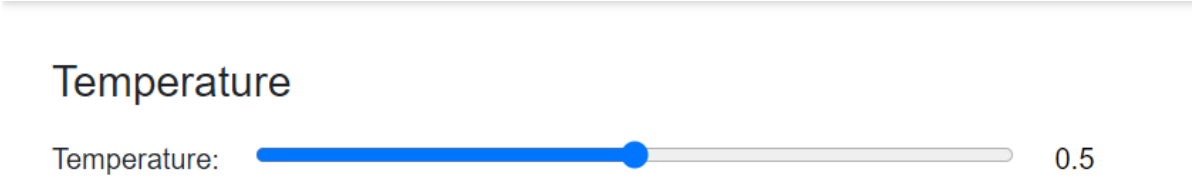


Рисунок 4.2 – Температура задання промту

2. Обрати мову програмування (рис. 4.3);

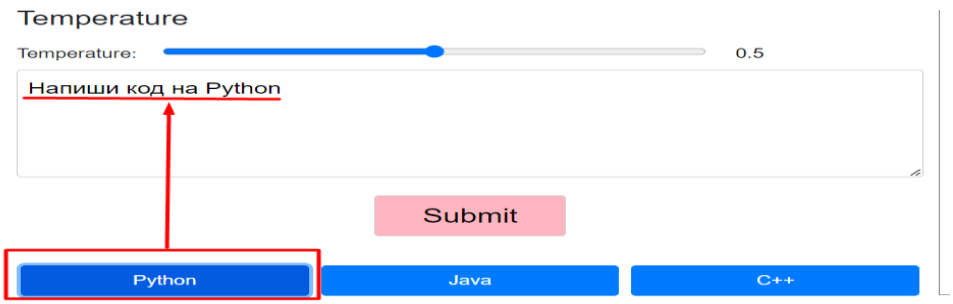


Рисунок 4.3 – Обрання мови програмування

3. Обрати шаблон задання промту (рис. 4.4);

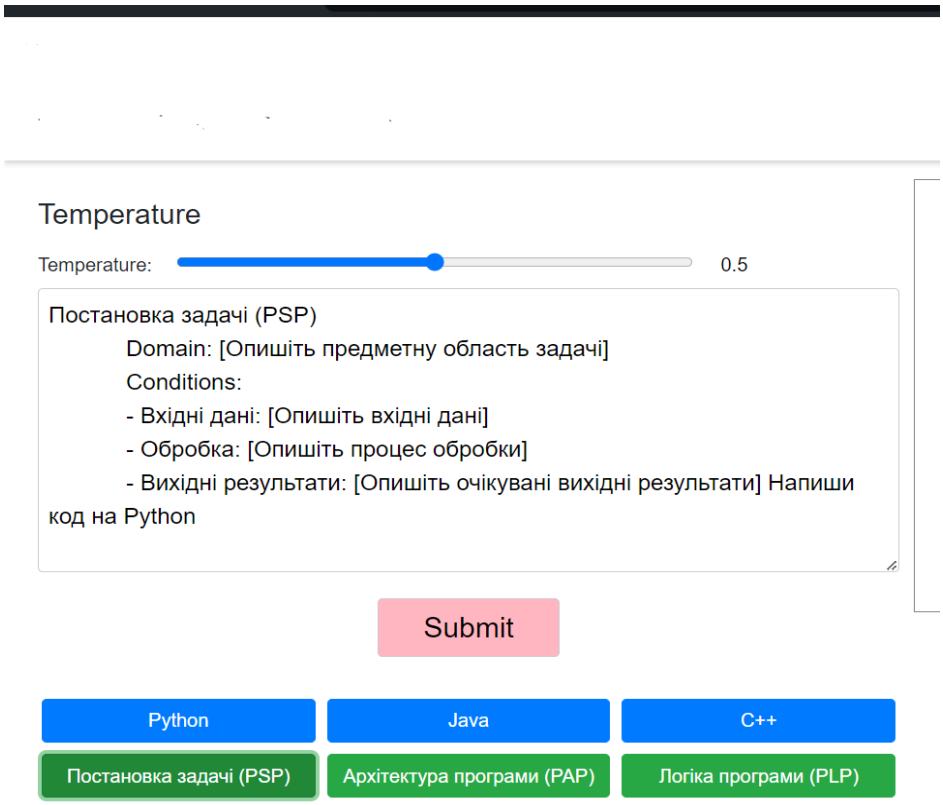


Рисунок 4.4 – Обрання шаблону для постановки задачі (PSP)

4. Використовуючи наданий шаблон вводимо необхідні параметри власного промту та надсилаємо;

5. У полі виводу результату отримуємо готовий код та копіюємо (рис. 4.5).

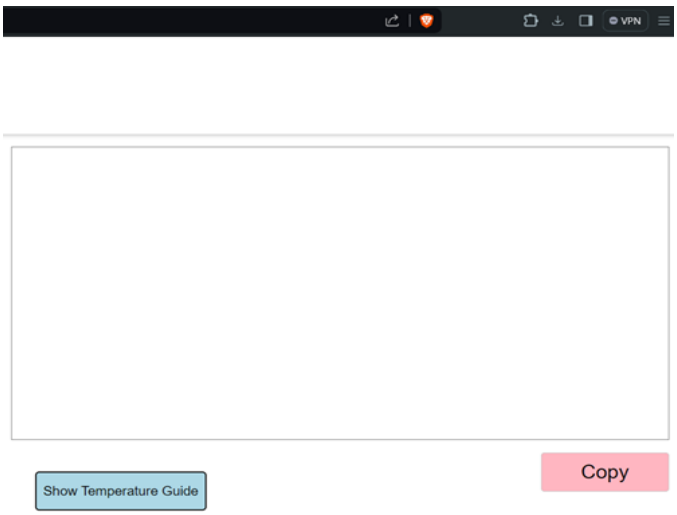


Рисунок 4.5 – Поле виводу результатів

4.4 Тестування системи

Тестування розробленого програмного забезпечення проводилось у кілька етапів для забезпечення його стабільності та відповідності вимогам:

1. Проведено модульне тестування системи. Перевірялися окремі функції та модулі програми, зокрема функції обробки запитів до OpenAI GPT-5 та функції веб-інтерфейсу (Рис. 4.6). Було створено тестові запити для різних типів завдань, щоб оцінити коректність згенерованого коду.

2. Проведено інтеграційне тестування системи. Протестовано інтеграцію між різними компонентами системи, такими як серверна частина на Flask та API OpenAI. Перевірялася взаємодія між веб-інтерфейсом і серверною частиною для забезпечення безперервної передачі даних.

3. Проведено функціональне тестування системи. Оцінювалася функціональність програми з боку користувача. Тестувалося введення різних запитів природньою мовою і перевірка коректності отриманого програмного коду (рис. 4.7).

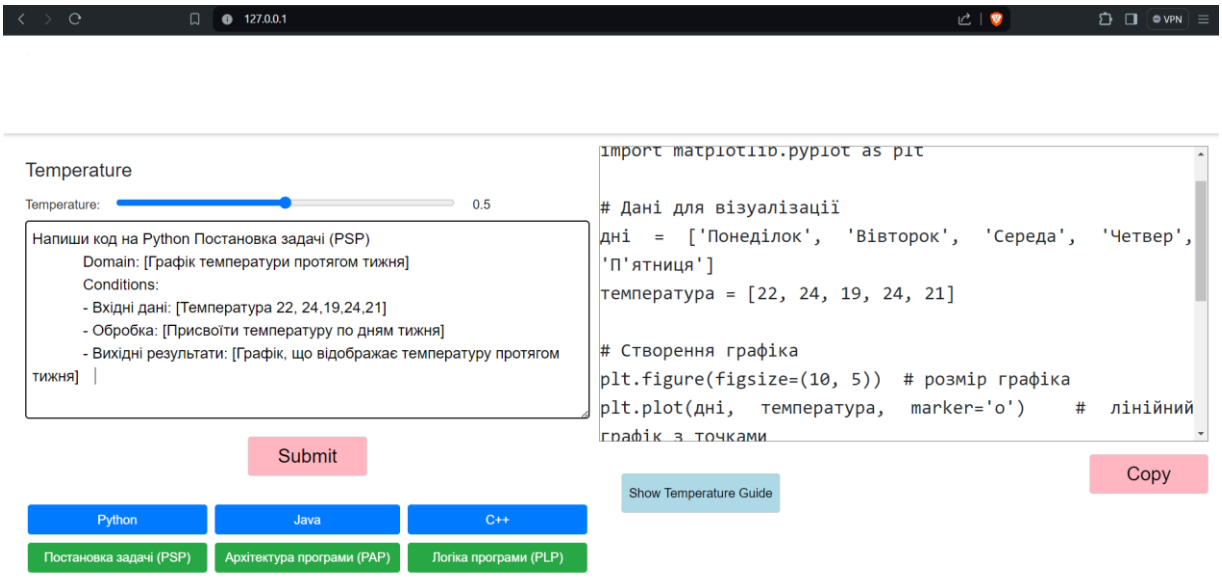


Рисунок 4.6 – Тестування програмного застосунку

main.py

```python
import matplotlib.pyplot as plt
# Дані для візуалізації
дні = ['Понеділок', 'Вівторок', 'Середа', 'Четвер', 'П'ятниця']
температура = [22, 24, 19, 24, 21]
# Створення графіка
plt.figure(figsize=(10, 5))  # розмір графіка
plt.plot(дні, температура, marker='o')  # лінійний графік з точками
# Додавання назви графіка і міток осей
plt.title('Температура протягом тижня')
plt.xlabel('День тижня')
plt.ylabel('Температура (°C)')
# Відображення графіка
plt.grid(True)
plt.show()
```

Рисунок 4.7 – Результати надані системою

Тестування показало, що розроблене програмне забезпечення успішно виконує поставлені завдання (рис. 4.8). Згенерований код відповідає запитам користувачів у більшості випадків, що свідчить про високу точність роботи підсистеми у зв’язці з моделлю GPT-5.

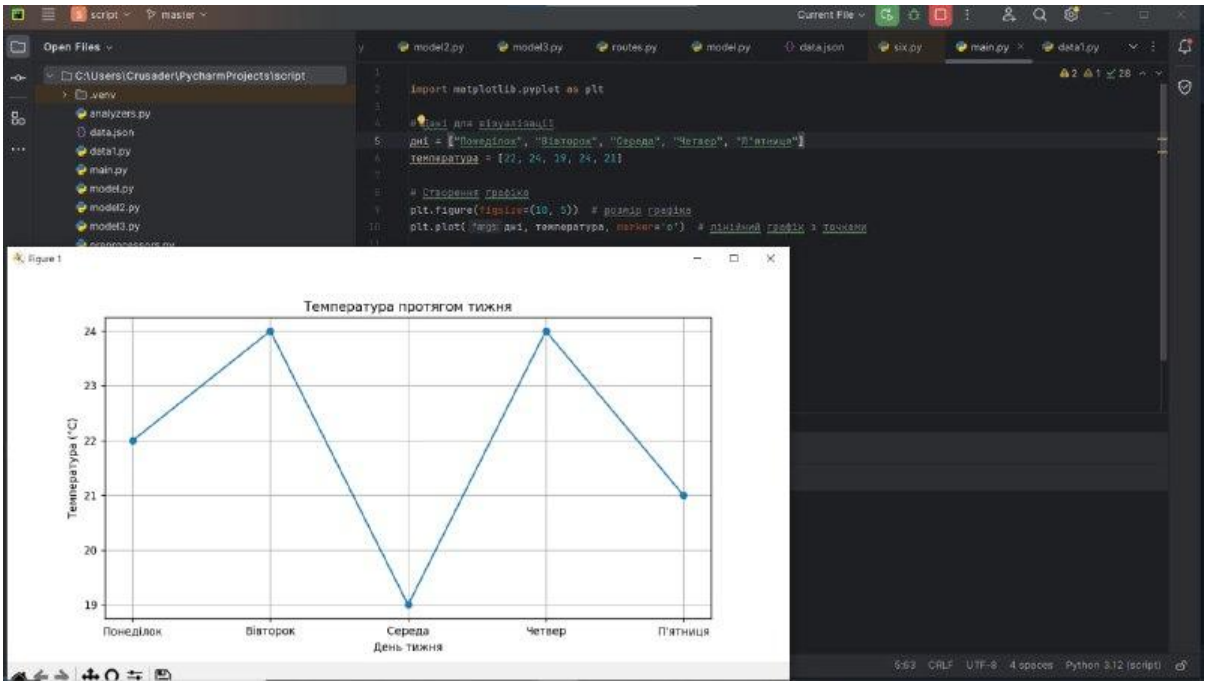


Рисунок 4.8 – Компіляція результатів

Веб-інтерфейс є інтуїтивно зрозумілим і простим у використанні. Система забезпечує надійну та ефективну взаємодію з користувачем, та є готовою до впровадження і використання в робочому середовищі. Також представлена підсистема має широкі можливості до вдосконалення шляхом збільшення кількості перевірених шаблонів промтів для розширення функціоналу на основі відгуків користувачів та статистиці подібних запитів інших розробників програмного забезпечення з усього світу.

Висновки до розділу 4

В цьому розділі була надана інструкція з використання програмного забезпечення – підсистеми для перетворення природньої мови в програмний код та проведено відповідні тестування. Інструкція охоплює основні кроки, які необхідно виконати для успішного використання розробленої підсистеми.

Задаючи промт користувачі мають можливість обрати мову програмування із запропонованих, необхідний шаблон задання промту (PSP, PAP, PLP) та за рахунок взаємодії з OpenAI GPT-5 отримати результати у вигляді програмного коду. Результати представлені у відповідному полі з можливістю повного копіювання для подальшого використання.

Ця інструкція допоможе користувачам ефективно використовувати представлену підсистему для перетворення природньої мови в програмний код та дає змогу отримати необхідний результат. Розроблена підсистема спростить процес перетворення природньої мови в програмний код і допоможе знизити витрати часу для розробки програмного продукту.

Функціональне тестування підсистеми показало, що розроблене програмне забезпечення є достатньо єфективним для виконання поставлени завдань. Згенерований код відповідає очікуваним результатам у більшості випадків, що свідчить про високу точність роботи підсистеми у зв’язці з моделлю GPT-5.

ВИСНОВКИ

У магістерській роботі проведено комплексне дослідження впливу використання штучного інтелекту на етапи написання програмного забезпечення та переваги використання інструментів автоматизації програмування на основі штучного інтелекту. На основі отриманих емпіричних даних та порівняльного аналізу інструментів (GitHub Copilot, Amazon Code Whisperer) сформульовано наступні висновки:

1. Проведено аналіз найпопулярніших інструментів написання програмного забезпечення на основі штучного інтелекту (GitHub Copilot, Amazon Code Whisperer), який показав переваги та недоліки використання подібних систем програмування, в результаті чого - встановлено потребу в розробці власної системи та окреслено вимоги до її параметрів.

2. Під час дослідженя впливу використання технологій штучного інтелекту для створення програмного забезпечення, виявлено зниження часових витрат на написання програмного коду на 35-45% за рахунок автоматизації рутинних завдань та зниження кількості помилок що зумовлені людським фактором, на тестування та налагодження програмного забезпечення на 50% за рахунок можливості тестування як окремих сегментів коду так і всього масиву одночасно. Встановлено ключові вимоги до розроблюваної підсистеми.

3. Проведено розробку підсистеми перетворення природної мови в програмний код з підтримкою використання генеративної мовної моделі GPT-5 на мові програмування Python з використанням підготовлених промтів українською мовою та необхідних налаштувань для отримання максимально чіткого та коректного результату. Підтверджено, що застосування розробленої системи програмування забезпечує істотну оптимізацію процесу написання коду: часові витрати зменшуються на 40% порівняно з традиційними підходами взаємодії із системами штучного інтелекту.

4. Сформульовано практичні рекомендації щодо системних вимог та чіткі інструкції користувача з необхідними роз’ясненнями, щодо роботи з програмою.

Практичне тестування підсистеми підкреслило її здатність до виконання завдань з написання програмного коду на достатньому рівні, що дозволяє використовувати її в середовищі розробки програмного забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

Використання ChatGPT для сценаріїв розробки програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://visuresolutions.com/uk.

The Future of Coding: How AI is Revolutionizing Programming [Електронний ресурс] – Режим доступу до ресурсу: https://nandbox.com/the-future-of-coding-how-ai-is-revolutionizing-programming/.

1. API keys [Електронний ресурс] – Режим доступу до ресурсу: https://platform.openai.com/organization/api-keys.

2. Промт інжиніринг [Електронний ресурс] – Режим доступу до ресурсу: https://prompt.com.ua/pidruchnik.

3. Програмний синтез з використанням природної мовиз використанням нейронних мереж [Електронний ресурс] – Режим доступу до ресурсу: https://semanticscholar.org/paper/Program-Synthesis-from-Natural-Language-Using-Lin.

4. Майбутнє програмування: як штучний інтелект змінює розробку програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://journals.maup.com.ua/index.php/it/article/view/4580/4887.

5. Штучний інтелект у розробці програмного забезпечення: огляд та перспективи [Електронний ресурс] – Режим доступу до ресурсу:

https://ieeexplore.ieee.org/document/10176436.

6. Як штучний інтелект революціонізує розробку програмного забезпечення на замовлення [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/pulse/how-ai-future-revolutionizing-custom-software-mariana-demian.

7. What is the Role of AI in DevOps [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/kubi-ya/what-is-the-role-of-ai-in-devops-1p6b.

8. Автоматизація розробки програмного забезпечення за допомогою штучного інтелекту: огляд [Електронний ресурс] – Режим доступу до ресурсу: https://dl.acm.org/doi/10.1145/3487043.

9. Unleashing developer productivity with generative AI: огляд [Електронний ресурс] – Режим доступу до ресурсу: https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/unleashing-developer-productivity-with-generative-ai

ДОДАТОК А
ЛІСТИНГ ВИХІДНОГО КОДУ ПРОГРАМИ

```
      from flask import Flask, request, render_template, redirect
      import requests

      url        =        "https://cheapest-gpt-5-turbo-gpt-5-vision-chatgpt-openai-ai-
api.p.rapidapi.com/v1/chat/completions"
      import re

      payload = [
            {
                  "content":   "Напиши   код   для   звичайної   програми   на   Python,
калькулятор.",
                  "role": "user"
            }
      ]
      headers = {
            "content-type": "application/json",
            "X-RapidAPI-Key": "19e283ce09msh757aed4b81e1134p1192d2jsn6e6555cd83ab",
            "X-RapidAPI-Host": "chatgpt-api8.p.rapidapi.com"
      }

      server = Flask(__name__)

      def send_gpt(prompt, tem):
          try:
              payload = {
              "messages": [
                  {
                      "role": "user",
                      "content": prompt
                  }
              ],
              "model": "gpt-5-turbo-2025-04-09",
              "max_tokens": 500,
              "temperature": tem
              }
              headers = {
              "content-type": "application/json",
              "X-RapidAPI-Key": "19e283ce09msh757aed4b81e1134p1192d2jsn6e6555cd83ab",
              "X-RapidAPI-Host":  "cheapest-gpt-5-turbo-gpt-5-vision-chatgpt-openai-ai-
api.p.rapidapi.com"
              }
              response = requests.post(url, json=payload, headers=headers)

              text = response.json()['choices'][0]['message']['content']
              start_index = text.find("```python")
              end_index = text.find("```", start_index + 1)

              code = text[start_index:end_index + 3] # Включаємо три обратні скобки в
конці блоку
              return code
          #['choices'][0]['message']['content']
          except:
              mess = "Connection Error! Please try again."
              return mess

      @server.route('/', methods=['GET', 'POST'])
      def get_request_json():
          if request.method == 'POST':
```

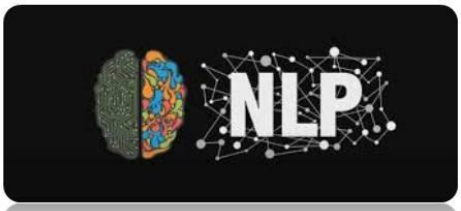
```
            if len(request.form['question']) < 1:
                return render_template(
                    'chat_2.html', question="NULL", res="Question can't be empty!",
temperature="NULL")
            question = request.form['question']
            temperature = float(request.form['temperature'])
            print("====================================")
            print("Receive the question:", question)
            print("Receive the temperature:", temperature)
            res = send_gpt(question, temperature)
            print("Q：\n", question)
            print("A：\n", res)

            return  render_template('chat_2.html',  question=question,  res=str(res),
temperature=temperature)
        return render_template('chat_2.html', question=0)

    if __name__ == '__main__':
        server.run(debug=True, host='0.0.0.0', port=80)
```

ПРЕЗЕНТАЦІЯ

Актуальність та мета роботи

Дана робота є актуальною в сучасному контексті швидкого розвитку технологій штучного інтелекту та автоматизації процесів розробки програмного забезпечення. Зокрема, використання моделей штучного інтелекту, таких як OpenAI, для перетворення природної мови в програмний код, з метою зменшення витрат часу на написання програмного коду, що відкриває нові можливості для програмістів і розробників.

Метою даної роботи є підвищення ефективності процесу розробки програмного забезпечення шляхом застосування штучного інтелекту.



Об’єкт та предмет дослідження

Об'єкт дослідження - системи автоматизації програмування та вплив використання штучного інтелекту на тривалість виконання завдань.

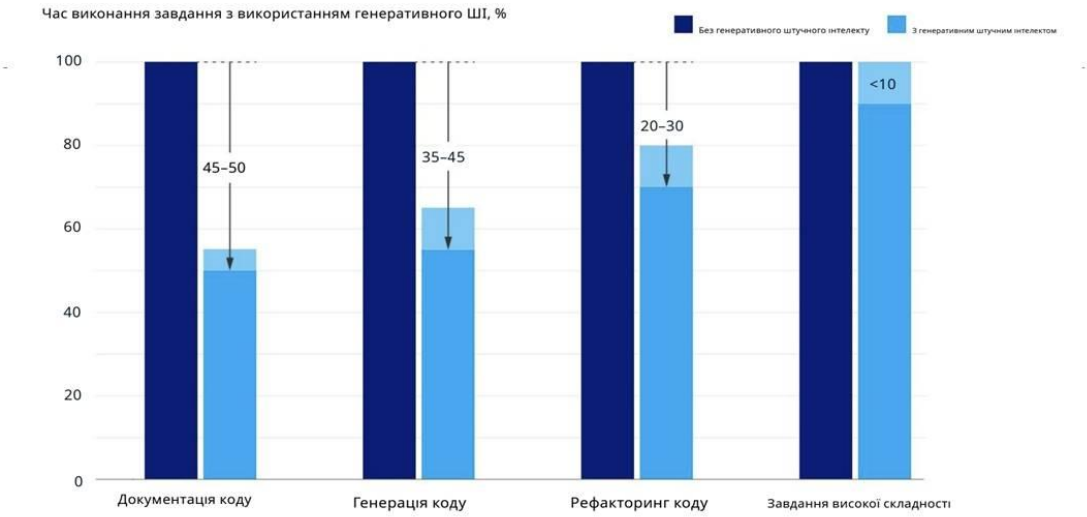
Предмет дослідження - зниження часових витрат та оптимізація процесу програмування природньою мовою за рахунок взаємодії з системами OpenAI.

В роботі проаналізовано найпопулярніші існуючі системи програмування на основі штучного інтелекту. Виявлено залежність між різними типами використання систем зі штучним інтелектом та тривалістю виконання поставлених задач. Обґрунтовано необхідність створення підсистеми NLP (Natural Language Processing) на базі технологій OpenAi з поліпшеною підтримкою української мови та базовими шаблонами промтів.

Існуючі системи для програмування природньою мовою

Підсистеми GitHub Copilot та CodeWhisperer можуть генерувати код на основі природної мови, коментарів та наявного коду. В обох ситемах прослідковується спільна проблема, а саме відсутність прямої взаємодії з українською мовою та відсутні промт-інструкції для кращої можливості взаємодії користувача та системи.

Зменшення часових витрат на виконання операцій



Порівняння підходів до розробки ПЗ

	Традиційний підхід	ШІ-орієнтований підхід	Відмінність (%)
Середній час розробки модуля	180 годин	110 годин	-40%
Кількість помилок на 1000 рядків коду	8-10	5-6	-40%
Час на тестування та налагодження	60 годин	30 годин	-50%
Якість коду (за оцінкою аналізатора)	7.2/10	9.0/10	+31%

Загальні вимоги до підсистеми

На програмне забезпечення підсистеми перетворення природньої мови у програмний код покладаються такі основні функції:

а) забезпечення необхідними шаблонами промптів;

б) можливість обрання мови програмування;

в) генерація програмного коду по заданим параметрам користувача, готового до компіляції та виконання.

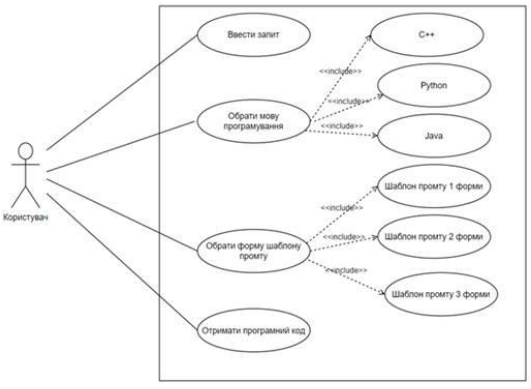


Рисунок 1 – Діаграма варіантів використання

Архітектура розробленої підсистеми

Процес перетворення природньої мови в програмний код здійснюється в декілька етапів декількома модулями:

а) Модуль вибору шаблону промту.

б) Модуль вибору мови програмування вихідного коду.

в) Модуль взаємодії з OpenAI для генерування вихідного коду.

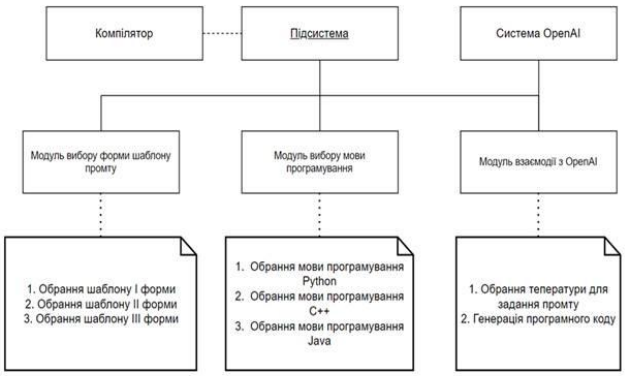


Рисунок 4 – Загальна структура системи

Загальний алгоритм роботи підсистеми

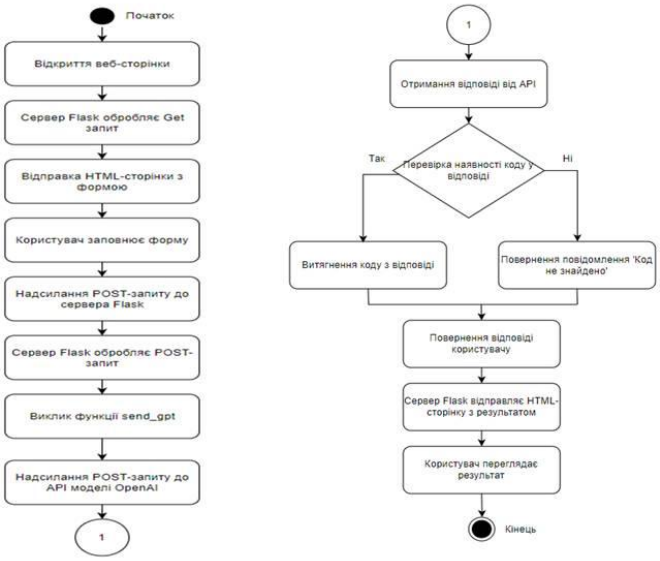


Рисунок 6 – Діаграма діяльності UML

Інтерфейс підсистеми

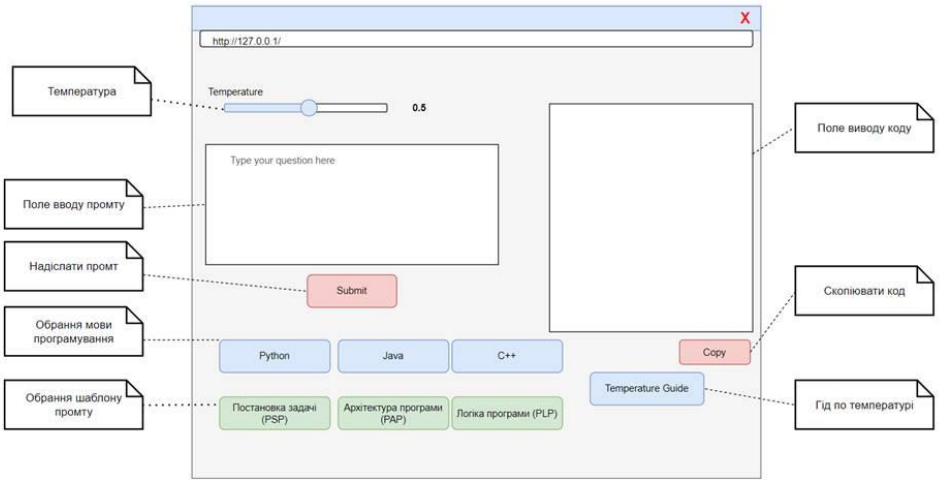


Рисунок 7 – Дизайн веб-інтерфейсу

Розроблені шаблони промтів

Постановка задачі (PSP)

Domain: [Опишіть предметну область задачі]

Conditions:

- Вхідні дані: [Опишіть вхідні дані]
- Обробка: [Опишіть процес обробки]
- Вихідні результати: [Опишіть очікувані вихідні результати]

Архітектура програми

(PAP)

Lib: [Список бібліотек, які потрібно використати]

Classes:

- TaskManager: Клас для управління завданнями, містить методи для додавання, редагування, видалення та отримання завдань.
- Task: Клас для представлення окремого завдання, містить атрибути як-от назва, опис, дата завершення, пріоритет.

Functions:

- add_task(title, description, due_date, priority): Метод для додавання нового завдання.
- edit_task(task_id, title, description, due_date, priority): Метод для редагування існуючого завдання.
- delete_task(task_id): Метод для видалення завдання.
- get_tasks(filter_by, sort_by): Метод для отримання списку завдань з можливістю фільтрації та сортування.

DataStructures:

- Список (List) для зберігання завдань.
- Словник (Dictionary) для зберігання налаштувань фільтрації та сортування.

Логіка програми

(PLP)

Опис алгоритму:

1. [Крок 1: Опис]
2. [Крок 2: Опис]
3. [Крок 3: Опис]
4. [Крок 4: Опис]

Засоби реалізації підсистеми

1. Середовища програмування:
   - PyCharm.
2. Мова програмування:
   - Python 3.12.0.

Висновки

У рамках дипломної роботи було проведено комплексне дослідження впливу використання штучного інтелекту на етапи написання програмного забезпечення, виявлено зниження часових витрат на написання програмного коду на 35-45% за рахунок автоматизації рутинних завдань та зниження кількості помилок що зумовлені людським фактором, на тестування та налагодження програмного забезпечення на 50%. Розроблено програмне забезпечення з веб-інтерфейсом для перетворення природної мови в програмний код. Розроблюваний застосунок має на меті поліпшити процес написання коду, зробивши його більш інтуїтивно зрозумілим і доступним для користувачів з різним рівнем технічної підготовки. Підтверджено, що застосування розробленої підсистеми програмування забезпечує істотну оптимізацію процесу написання коду: часові витрати зменшуються на 40% порівняно з традиційними підходами взаємодії із системами штучного інтелекту.