КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗОВАНОГО СТВОРЕННЯ ДОКУМЕНТАЦІЇ КОДУ З ВИКОРИСТАННЯМ ФУНКЦІОНАЛУ CHATGPT

Виконав студент 4-го курсу
Олексій КОНДУС

_____
(підпис)

Науковий керівник:
асистент, кандидат фіз-мат наук
Костянтин ЖЕРЕБ

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
« ___ » _____ 2023 р.,
протокол № ___
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 54 сторінки, 11 ілюстрацій, 3 таблиці, 21 джерело посилання.

CLI ЗАСІБ, PYTHON, TELEGRAM-БОТ, АВТОМАТИЗАЦІЯ ДОКУМЕНТАЦІЇ КОДУ, ДОКУМЕНТУВАННЯ КОДУ, МОЖЛИВОСТІ CHATGPT, ШТУЧНИЙ ІНТЕЛЕКТ.

Об’єктом розробки є програмний засіб для автоматичної документації коду. Предметом роботи є CLI засіб та телеграм-бот для автоматичної документації коду з використанням функціональності ChatGPT.

Мета кваліфікаційної роботи полягає у розробці засобу для автоматичної документації коду використовуючи функціонал ChatGPT.

Інструменти розроблення: Google Colab з мовою програмування Python та PyCharm. У процесі розробки також використовувалися API Telegram та API ChatGPT. API Telegram було використано для інтеграції з месенджером Telegram, забезпечуючи зручну комунікацію з користувачами. API ChatGPT було використано для застосування інтелектуальних можливостей моделі ChatGPT у процесі документування коду.

Під час створення програмних засобів також використовувалися додаткові бібліотеки, зокрема openai, pytelegrambotapi і doxygen. Ці бібліотеки надали додатковий функціонал та інструменти для розробки, взаємодії з Telegram API та генерації документації коду за допомогою Doxygen.

Результати роботи: розроблено CLI засіб для документування коду та чат-бот для автоматизованого створення документації коду. Окрім цього, проаналізовано потенційні переваги та обмеження ChatGPT та його можливості, також підтверджено важливість кодової документації.

ЗМІСТ

С.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AI – Artificial Intelligence, штучний інтелект;

API – Application Programming Interface, прикладний програмний інтерфейс;

CLI – Command Line Interface, інтерфейс командного рядка;

CNN – Convolutional Neural Network, згорткова нейронна мережа;

DRY – Don’t Repeat Yourself, не повторюйся;

GPT – Generative Pre-trained Transformer, генеративний попередньо навчений трансформер;

IDE – Integrated Development Environment, інтегроване середовище розробки;

LLM – Large Language Model, велика мовна модель;

LSTM – Long Short-Term Memory, довга короткочасна пам'ять;

RNN – Recurrent Neural Network, рекурентна нейронна мережа.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Документація коду є важливим аспектом розробки програмного забезпечення, який служить багатьом цілям і відіграє вирішальну роль у забезпеченні довгострокового успіху проекту.

Розробники часто висловлюють небажання або вважають, що їм важко документувати свій код через різні причини, такі як часові обмеження, уявна відсутність негайної вигоди та проблеми, пов’язані з підтримкою документації.

Актуальність роботи та підстави для її виконання. Реалізація програми з використанням ChatGPT для автоматизованого документування коду дуже актуальна в галузі розробки програмного забезпечення. Написання та підтримка документації коду є трудомістким завданням для розробників. Автоматизація цього процесу за допомогою ChatGPT може значно скоротити час і зусилля, необхідні для створення та оновлення документації, дозволяючи розробникам зосередитися на більш критичних завданнях, таких як написання коду та вирішення проблем. Також документація коду вручну може містити помилки, невідповідності та пропуски, що може призвести до плутанини та неефективності проектів розробки програмного забезпечення. Автоматизуючи документацію коду за допомогою ChatGPT можна допомогти покращити якість і точність документації, забезпечуючи її актуальність і синхронізацію з кодовою базою.

За допомогою можливостей ChatGPT можна досягти узгодженості і стандартизації, адже коли над проектом працюють кілька розробників, підтримувати узгоджену документацію стає складно. Додаток, який використовує ChatGPT, може забезпечити узгоджену практику

документування, надаючи стандартизовану документацію і гарантуючи, що вся кодова база відповідає тим самим умовам.

Нові члени команди, які приєднуються до проекту, часто стикаються з труднощами в розумінні кодової бази та її документації. Автоматизуючи документацію коду за допомогою ChatGPT, застосунок зможе надавати вичерпні пояснення, приклади та ідеї, що полегшує новим розробникам процес адаптації та покращую глибоке розуміння проекту.

Після вбудовування автоматичної документації в побудову проекту, застосунок зможе автоматично оновлювати документацію, коли відбуваються зміни коду, гарантуючи, що документація залишається синхронізованою з останньою кодовою базою, зменшуючи ризик застарілої інформації.

Можливості ChatGPT також допомагають покращити перевірку коду, написаного розробником. Вони включають оцінку не лише коду, але й супровідної документації. В майбутньому, додаток може допомогти в процесі перевірки коду, автоматично виявляючи та висвітлюючи проблеми з документацією, забезпечуючи ефективніші перевірки коду.

Підсумовуючи, створення програми для автоматизованого документування коду має велике значення для вирішення проблем якості документації, часових і ресурсних обмежень, співпраці, адаптації, безперервних оновлень, перевірки коду, масштабованості та відтворюваності. Він надає значні переваги групам розробників програмного забезпечення, покращуючи ефективність, точність і співпрацю, одночасно зменшуючи тягар завдань документування вручну.

Мета й завдання роботи. Мета кваліфікаційної роботи полягає у розробці засобу для автоматичної документації коду використовуючи функціонал ChatGPT. Для досягнення цієї мети поставлено такі завдання:

- Розглянути можливості ChatGPT;

– Підтвердити важливість кодової документації та проблеми, пов’язані з підтримкою точної та актуальної документації;

– Дослідити процес розробки інтеграції ChatGPT в програмний продукт та продемонструвати потенційні переваги та обмеження використання ChatGPT;

– Розробити CLI засіб для автоматизованого документування коду, який можна буде вбудувати в систему побудови проекту;

– Розробити зручний телеграм-бот для автоматизованого створення документації коду з посилання на GitHub.

Розв’язуючи ці завдання, буде розроблено застосунок з можливістю автоматичного додавання кодової документації використовуючи функціонал ChatGPT, а також буде проведено аналіз переваг, обмежень і потенційних застосувань ChatGPT у галузі розробки програмного забезпечення.

Об'єкт, методи й засоби розроблення. Об’єктом розробки є програмний засіб для автоматичної документації коду. В якості інструменту створення програмного засобу було обрано Google Colab – це безкоштовне інтерактивне хмарне середовище для роботи з кодом від Google, мовою програмування Python. Також в якості інструменту створення програмного засобу використовувався PyCharm.

У процесі розробки також використовувалися API Telegram та API ChatGPT. API Telegram було використано для інтеграції з месенджером Telegram, забезпечуючи зручну комунікацію з користувачами. API ChatGPT використовувалося для використання інтелектуальних можливостей ChatGPT по документуванню коду.

Під час створення програмних засобів також використовувалися додаткові бібліотеки, зокрема openai, pytelegrambotapi і doxygen. Ці бібліотеки надали додатковий функціонал та інструменти для розробки,

взаємодії з Telegram API та генерації документації коду за допомогою Doxygen.

Можливі сфери застосування. Програма може використовуватися командами розробників програмного забезпечення для автоматичного створення та підтримки документації коду, заощаджуючи час і зусилля розробників і забезпечуючи узгоджену та актуальну документацію для проектів.

В майбутньому, зробивши деякі покращення можлива інтеграція програми в редактори коду та IDE, які надаватимуть розробникам пропозиції та виправлення коду в режимі реального часу під час їх написання, допомагаючи розробникам дотримуватися найкращих практик і покращувати якість документації.

Платформи, які сприяють колективній розробці програмного забезпечення, наприклад GitHub або GitLab, можуть використовувати програму для автоматичного вдосконалення документації коду.

Програму можна інтегрувати в процеси перевірки коду, дозволяючи рецензентам швидко виявляти та виправляти проблеми з документацією коду. Це сприяє ефективнішому перегляду коду, що призводить до покращення якості документації.

Додаток можна використовувати на курсах кодування, семінарах і навчальних програмах, щоб надати учням автоматичний зворотний зв’язок щодо їхніх навичок документування коду. Це допомагає розробникам-початківцям удосконалити свою практику документування та виховує хороші звички з ранніх етапів їхньої подорожі програмування.

РОЗДІЛ 1 ОГЛЯД CHATGPT ТА ЙОГО МОЖЛИВОСТІ

1.1 Загальний огляд глибокого навчання

Протягом тривалого періоду часу глибоке навчання стало свідком значного прогресу, трансформуючи різні сфери штучного інтелекту та машинного навчання.

Глибоке навчання – це метод машинного навчання, який використовується для тренування штучних нейронних мереж шляхом автоматичного вивчення загальних правил з набору даних. Під час процесу навчання нейронна мережа налаштовує вагові коефіцієнти між своїми нейронами для ефективної обробки інформації. Часто навчання нейронних мереж, зокрема систем машинного зору, відбувається з використанням учителя, коли модель тренується на наборі прикладів з передбачуваними результатами.

За останні десятиліття глибоке навчання пройшло значний шлях і стало основою багатьох передових рішень у галузі штучного інтелекту та машинного навчання. На початку, нейронні мережі були обмежені і не мали достатніх обчислювальних потужностей для розв'язання складних завдань. Однак, з появою потужних обчислювальних ресурсів і великих наборів даних, глибоке навчання стало широко застосовуваним.

1.1.1 Історія розвитку моделей глибокого навчання

Історія розвитку моделей глибокого навчання пройшла довгий шлях від початкових концепцій до сучасних передових архітектур. Почнемо з основних етапів цього розвитку.

У 1940-х роках була запропонована перша концепція нейронних мереж – модель Маккалока-Піттса [1]. Вона використовувала прості математичні моделі для опису нейронних процесів у мозку.

У 1950-х роках Розенблатт запропонував перцептрон – модель штучного нейрону, яка мала здатність навчатися та вирішувати прості задачі класифікації. Це був важливий крок у розвитку нейромереж та викликав інтерес до їхнього потенціалу.

Протягом 1960-70-х років виникли різні підходи до нейромереж, включаючи зворотні нейронні мережі (RNN). RNN мають здатність зберігати внутрішні стани та передавати їх наступним шаром мережі, що дозволяє їм ефективно обробляти послідовні дані.

У 1980-х роках з'явилися згорткові нейронні мережі (CNN), які стали основним інструментом для обробки зображень. CNN використовують згорткові та пулінгові шари для виявлення візуальних ознак у зображеннях, що відкрило нові можливості у сфері комп'ютерного зору.

У 1990-2000-х роках нейронні мережі стали переживати спад популярності через обмеженості в обчислювальній потужності та доступній кількості даних для навчання. Проте, деякі дослідники продовжували працювати над покращенням нейромереж та алгоритмів навчання.

У 2006 році з'явилася робота «A Fast Learning Algorithm for Deep Belief Nets» Геффроя Гінтона, яка привернула увагу до потенціалу глибокого навчання. Це стало каталізатором для подальшого розвитку цієї галузі.

У 2012 році нейромережа AlexNet, яка використовувала глибокі згорткові шари, виграла змагання ImageNet з надзвичайною перевагою над іншими моделями. Цей успіх підкреслив потужність глибоких нейромереж у розпізнаванні зображень.

В 2014 році була запропонована архітектура довгої короткочасної пам'яті (LSTM) для рекурентних нейронних мереж. LSTM має додаткові

механізми, що дозволяють моделі керувати процесом забування та запам'ятовування, що зробило їх ефективними для обробки послідовних даних з тривалими залежностями.

У 2017 році була запропонована модель трансформер, яка стала проривом у обробці послідовних даних, зокрема в машинному перекладі. Трансформери використовують механізм уваги для ефективної обробки великих послідовностей та здатні до паралельної обробки, що робить їх особливо ефективними для багатьох завдань обробки мови.

Таким чином, від початкових нейронних мереж до сучасних трансформерів, історія розвитку моделей глибокого навчання є шляхом постійних покращень і інновацій, що дозволяють здійснювати більш складну обробку даних та досягати кращих результатів у сфері штучного інтелекту. Докладніше питання історії розвитку моделей глибокого навчання висвітлено в [1], а огляд глибокого навчання висвітлено в [2].

1.1.2 Огляд нейронних мереж

Нейронні мережі є основою глибокого навчання. Вони побудовані за аналогією зі структурою нервової системи людини і складаються зі штучних нейронів, які взаємодіють один з одним.

Кожен нейрон мережі отримує вхідні сигнали, обробляє їх, і передає результати наступним нейронам у мережі. Кожен нейрон має ваги, які визначають його вплив на вхідні дані. Після обробки вхідних сигналів, нейрон використовує активаційну функцію для генерації вихідного сигналу, який передається до наступного шару нейронної мережі.

Глибокі нейронні мережі складаються з багатьох шарів нейронів, які дозволяють моделі розрізняти складні залежності у вхідних даних. Перші шари нейронів виконують прості завдання розпізнавання, а подальші шари абстрагуються від вхідних даних та засвоюють все більш складні концепції.

Нейронні мережі навчаються за допомогою алгоритму зворотного поширення помилок, який дозволяє коригувати ваги нейронів залежно від отриманих результатів. Цей процес тренування здатний досягати вражаючої точності і використовується в багатьох глибоких моделях навчання.

Можемо розглянути декілька нейронних мереж включаючи згорткові нейронні мережі (CNN) для обробки зображень, рекурентні нейронні мережі (RNN) для обробки послідовностей та трансформери (Transformer) для моделювання залежностей між словами у тексті.

Згорткові нейронні мережі (CNN) є основним інструментом для обробки зображень. Вони використовуються для виявлення різних візуальних ознак у зображеннях шляхом застосування фільтрів. Кожен фільтр шукає конкретний патерн або ознаку, таку як край, форма або текстура, у вхідному зображенні. Послідовність фільтрів утворює згортковий шар, який допомагає відображати важливі особливості об'єктів на різних рівнях абстракції. Після згорткових шарів, зазвичай використовуються повнозв'язні шари для класифікації або розпізнавання об'єктів на зображенні.

Рекурентні нейронні мережі (RNN) здатні працювати з послідовними даними, такими як текст природньою мовою, музика або часові ряди. У RNN кожен вихід залежить від попереднього виходу та поточного вхідного сигналу. Це дозволяє RNN зберігати внутрішні стани та враховувати контекстуальну інформацію при обробці послідовності. RNN широко використовуються в машинному перекладі, генерації тексту, аналізі настрою тексту та розпізнаванні мови.

Трансформери (Transformer) представляють собою відносно нову архітектуру для обробки послідовностей, яка здобула велику популярність, зокрема в машинному перекладі та моделях обробки природної мови. Вони замінюють рекурентні шари механізмами уваги (attention mechanisms), які

дозволяють моделі фокусуватись на різних частинах вхідної послідовності при обчисленні вихідного сигналу. Це дозволяє трансформерам здійснювати ефективну та паралельну обробку довгих послідовностей, що приводить до кращої якості результатів у багатьох завданнях обробки мови.

Підсумовуючи, CNN, RNN та трансформери є потужними інструментами глибокого навчання, які використовуються для обробки різних типів даних та розв'язання різноманітних завдань у сфері штучного інтелекту. Кожна з цих архітектур має свої унікальні переваги та застосування, і вони продовжують розширювати можливості глибокого навчання.

1.2 Можливості використання ChatGPT

ChatGPT від OpenAI — це модель розмовної мови, яка може відповідати на запитання або виконувати завдання, які дають текстову відповідь. На перший погляд може здатися, що ChatGPT схожий на стандартний чат-бот, насправді він виявляється значно більш розширеним та потужним. Стандартні чат-боти зазвичай призначені для повернення попередньо визначеної відповіді на обмежену кількість запитань. ChatGPT є набагато більш узагальненим, оскільки він використовує своє розуміння мови для інтерпретації запитання чи завдання та визначення найбільш прийнятної відповіді. Це означає, що ChatGPT має широкий спектр потенційних застосувань.

1.2.1 Опис моделі GPT

ChatGPT є прикладом генеративної моделі ШІ. Generative AI — це підмножина штучного інтелекту та машинного навчання, де модель створює новий вміст на основі шаблонів інформації, яку вона вже бачила.

У випадку ChatGPT згенерованим вмістом є текстом, але існують і інші моделі для створення зображень, аудіо та навіть відео.

Творці ChatGPT використали комбінацію контрольованого навчання (Supervised Learning) і навчання з підкріпленням (Reinforcement Learning) для точного налаштування ChatGPT, але саме компонент навчання з підкріпленням робить ChatGPT унікальним.

GPT побудовано на основі архітектури Transformer. Ця архітектура використовує механізми самоконтролю, які дозволяють моделі зосереджуватися на різних частинах вхідного тексту, дозволяючи їй фіксувати складні контекстуальні залежності. Ефективно аналізуючи та генеруючи послідовності слів, GPT чудово справляється з такими завданнями, як переклад мов, резюмування, відповіді на запитання тощо.

Навчання GPT включає двоетапний процес: попереднє навчання (pre-training) та тонке налаштування (fine-tuning). Під час попереднього навчання модель вчиться на великому наборі загальнодоступного тексту, набуваючи широкого розуміння граматики, фактів і семантичних зв’язків. Цей етап навчання надає GPT повну базу знань, завдяки чому він здатний генерувати послідовні та контекстуально релевантні відповіді. Тонке налаштування відбувається після переднього навчання моделі і включає навчання моделі на конкретних наборах даних, створених людиною, щоб узгодити її поведінку з бажаними результатами, використовуючи приклади та зворотний зв'язок.

Однією з відмінних особливостей GPT є використання розробниками спеціальної техніки під назвою Reinforcement Learning from Human Feedback, яка використовує відгук людини в циклі навчання, щоб мінімізувати шкідливі, неправдиві та/або упереджені результати. Ітеративно вдосконалюючи модель на основі зворотного зв’язку від оцінювачів, GPT може навчитися генерувати більш точні, надійні та правильні відповіді. Докладніше модель ChatGPT висвітлена в [3] – [7].

1.2.1.1 Нейронна мережа Transformer

Transformer зосереджується на механізмах уваги, включаючи самоувагу (self-attention) та перехресну увагу (cross-attention). Ці механізми допомагають моделі встановлювати зв'язки між різними словами в тексті.

Самоувага (self-attention) використовується для встановлення зв'язків між словами в межах однієї послідовності. Кожному слову надається увага відносно інших слів у тій же послідовності. Таким чином, модель може враховувати контекст і встановлювати важливі зв'язки між словами.

Перехресна увага (cross-attention) використовується для встановлення зв'язків між словами в різних послідовностях. Наприклад, в машинному перекладі, коли вхідна послідовність перекладається на вихідну послідовність, перехресна увага допомагає моделі приділити увагу важливим словам у вихідній послідовності для кожного слова у вхідній послідовності.

На рис. 1 можемо побачити приклад самоуваги, де слова в послідовності з'єднуються з іншими частинами тієї ж послідовності. У перехресній увазі відбувається подібний процес, але зв'язок встановлюється між двома різними послідовностями.

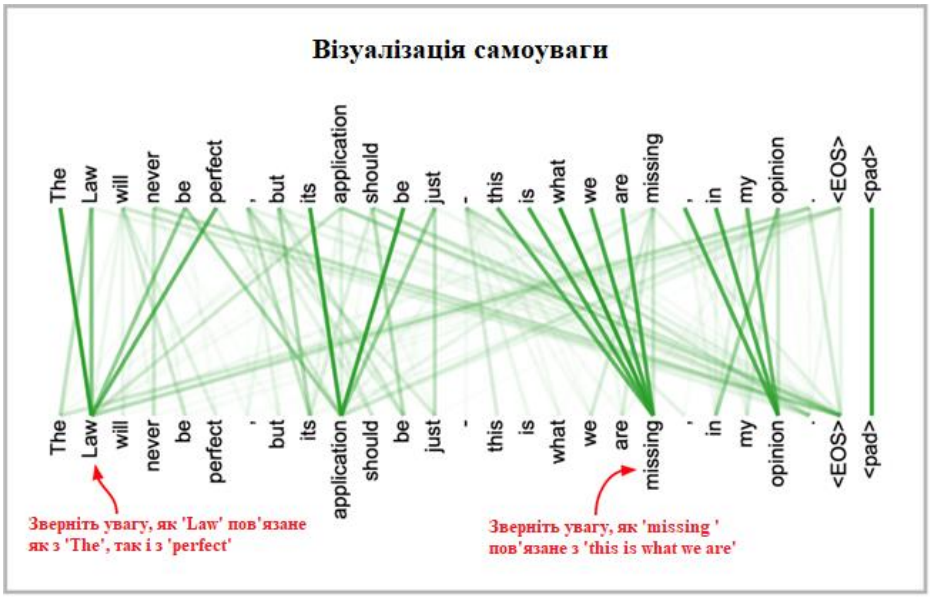

Рисунок 1 – Приклад самоуваги [8]

Transformer складається з двох основних компонентів: декодувальника та кодувальника. Декодувальник відповідає за перетворення вхідної послідовності у внутрішнє представлення, яке включає контекст інформації про всю послідовність. На рис. 2 можемо побачити схему декодувальника.

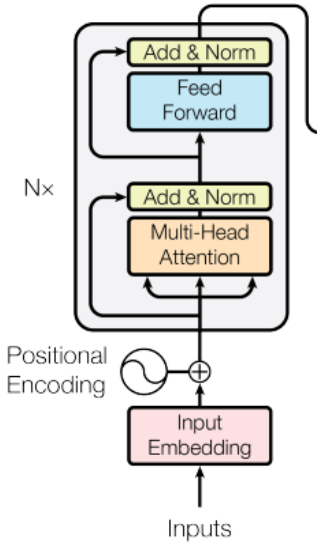

Рисунок 2 – Схема декодувальника [8]

Суть ідеї полягає в тому, що кожне слово в процесі обробки пропускається через різні шари, зображені на рис. 2.

Деякі з цих шарів є стандартними повнозв'язними шарами (fully-connected layers), а деякі використовують з'єднання короткого зв'язку (shortcut connections). Нововведенням у цих шарах є «Multi-head attention» – спеціальний шар, який дозволяє кожному вхідному вектору взаємодіяти з іншими словами за допомогою механізму уваги (attention mechanism).

В результаті цих шарів та їх взаємодії слова отримують здатність уважно співпрацювати між собою і забезпечити краще розуміння зв'язків між ними у контексті обробки послідовності. Кодувальник використовує це представлення для генерації вихідної послідовності. Основна структура кодувальника включає кілька шарів самоуваги та перехресної уваги, а також повнозв'язані шари. На відміну від декодувальника, кодувальник працює зі зсувом, тобто він генерує послідовність виходів крок за кроком.

На початку декодування, вхідний вектор (який може бути початковим токеном або вихідним вектором попереднього кроку) проходить через шар самоуваги, що допомагає моделі звернути увагу на різні частини послідовності. Потім виконується перехресна увага, де кодувальник отримує доступ до контексту декодувальника і враховує його при генерації наступного вектора виходу. Цей процес повторюється для кожного кроку генерації послідовності.

Крім того, кодувальник має вбудований механізм уваги на декодувальник, який допомагає моделі вирішувати проблему довгих залежностей і зв'язків між вхідною та вихідною послідовностями. Це досягається шляхом використання уваги на контекст декодувальника під час генерації вихідної послідовності. Докладніше питання архітектури трансформера висвітлено в [4], [8] – [9]. Загальну схему трансформера можемо побачити на рис. 3.

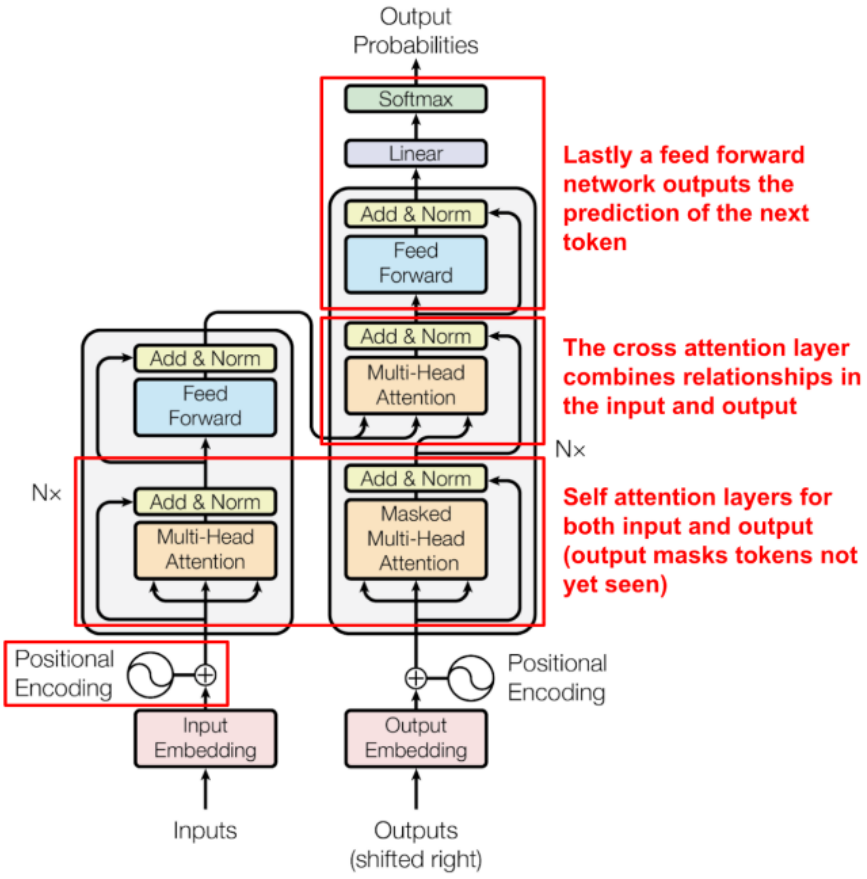

Рисунок 3 – Архітектура моделі трансформера [8]

1.2.2 Огляд можливостей та обмежень моделі ChatGPT

Незважаючи на те, що ChatGPT є цінним інструментом, який може виконувати величезну різноманітність завдань, є деякі обмеження, про які слід знати, щоб використовувати його ефективно. У ChatGPT всі процеси відбуваються всередині великої мовної моделі. LLM інтерпретує підказку та генерує відповідний текст у відповідь на основі свого розуміння мови.

Перше обмеження: ChatGPT навчався на даних до певної дати, і модель не підключена до Інтернету чи інших зовнішніх джерел, тому вона не знає про події після цієї дати. Ще одним обмеженням є можливий вплив помилкових даних на навчання. ChatGPT навчався на величезному наборі текстових даних з різних джерел, включаючи книги, статті та веб-сайти, але ці дані можуть містити упередження. Модель може засвоїти ці упередження та дати упереджені відповіді.

ChatGPT має можливість спиратися на інформацію та контекст з попередніх розмов, тому може вносити подальші виправлення. Однак якщо тема розмови змінюється кілька разів, ChatGPT може неправильно відстежувати контекст і генерувати неточні або нерелевантні відповіді. Гарне емпіричне правило [10] полягає в тому, щоб обговорювати одну тему та створювати нові розмови для інших тем.

Іншою поширеною проблемою під час взаємодії з ChatGPT є галюцинації, коли модель впевнено повідомляє нам неточну інформацію. Це часто трапляється під час спроби вийти за межі обмеження знань або здібностей ChatGPT. Наприклад, якщо попросити ChatGPT узагальнити профіль LinkedIn за URL-адресою – то замість того, щоб повідомляти нам, що він не може відкривати URL-адреси, ChatGPT впевнено надає абсолютно неправдиву інформацію.

Усвідомлення цих обмежень допоможе зрозуміти, як можна ефективно та відповідально використовувати ChatGPT.

1.2.3 Порівняння моделі ChatGPT з іншими моделями

ChatGPT відкриває нові можливості для взаємодії з мовними моделями, але важливо зрозуміти його місце в порівнянні з іншими моделями. На фоні швидкого розвитку глибокого навчання та штучного інтелекту, з'явилися ряд нових моделей і надали значний внесок у цю галузь. У цьому розділі розглянемо деякі з них та порівняємо з ChatGPT.

1.2.3.1 ChatGPT vs XLNet

XLNet — це вдосконалена мовна модель, як і ChatGPT. Він був розроблений Google Research на основі архітектури Transformer. Подібно до ChatGPT, XLNet використовує авторегресійний підхід для обробки мовних запитів. У таблиці 1 можемо побачити відмінності між ChatGPT та XLNet.

Таблиця 1 – Відмінність ChatGPT/XLNet

Функції	**ChatGPT**	**XLNet**
Набір даних	З точки зору розміру, цей складний чат-бот AI становить 45 ТБ.	Ця мовна модель навчена з більш ніж 130 ГБ даних високоякісного тексту.
Архітектура	Чат-бот на основі ШІ працює на архітектурі Transformer. Він використовує модель авторегресії, яка дозволяє враховувати лише лівий контекст під час прогнозування.	Він також працює на архітектурі Transformer. Однак він ефективно фіксує двонаправлений контекст і залежності. Як наслідок, він здатний надавати точні відповіді на завдання обробки природної мови.

Продуктивність	Існують деякі обмеження, оскільки авторегресійний підхід попереднього навчання не такий ефективний, як інші традиційні мовні моделі.	Забезпечує кращу продуктивність завдяки підходу до автокодування з усуненням шумів під час попереднього навчання.
Обмеження довжини послідовності	Він має обмеження на довжину послідовності. Довжина його контексту становить 32 768 токенів.	Довжина послідовності не обмежена.

1.2.3.2 ChatGPT vs PaLM 2

Pathways Language Model 2 (PaLM 2) — мовна модель, представлена Google. Це наступник PaLM, потужної мовної моделі, яка використовувалася для широкого спектру програм. Сучасна модель мови здатна обробляти мовні завдання високого рівня, такі як міркування, кодування та підтримка багатомовного перекладу. У таблиці 2 можемо побачити відмінності між ChatGPT та PaLM 2.

Таблиця 2 – Відмінність ChatGPT/ PaLM 2

Функції	**ChatGPT**	**PaLM 2**
Розмір набору даних	Розмір набору даних ChatGPT становить 45 ТБ.	Мовна модель PaLM 2 містить 540 мільярдів параметрів. Крім того, він має найбільшу системну конфігурацію на основі TPU з понад 6000 чіпами.

Багатомовність	Це багатомовний чат-бот. Він використовує методи машинного навчання для розуміння, створення та перекладу тексту більш ніж 50 мовами.	Він навчається на текстах з кількох мов. Він навчається більш ніж 100 мовами, що дозволяє йому розуміти, створювати та перекладати тексти. Це також дозволяє вивчати та розуміти складні лінгвістичні особливості, такі як загадки, вірші, ідіоми тощо.
Особливості	ChatGPT може запропонувати такі послуги NLP: генерація тексту, класифікація тексту, відповіді на запитання, машинний переклад, аналіз настроїв і розпізнавання тексту.	PaLM 2 може виконувати такі операції: розуміння природної мови, міркування здорового глузду, розуміння прочитаного в контексті, відповіді на запитання, завершення коду, семантичний аналіз, резюмування, логічні ланцюжки висновків, розпізнавання образів і пояснення жартів.

1.2.3.3 ChatGPT vs Bard

Bard — останній внесок Google у розробку мовних моделей. Подібно до ChatGPT, цей чат-бот є системою штучного інтелекту, розробленою для розмов. Незважаючи на те, що під час запуску він зустрів певну критику через широкий спектр збоїв, розробники внесли подальші вдосконалення,

щоб подолати це. У таблиці 3 можемо побачити відмінності між ChatGPT та Bard.

Таблиця 3 – Відмінність ChatGPT/Bard

Функції	**ChatGPT**	**Bard**
Доступність	Він доступний через веб-сайт *OpenAI* .	Для доступу до Google Bard користувачам потрібен обліковий запис *Google*.
Архітектура	Він заснований на технології генеративного штучного інтелекту, відомої як Generative Pre-Trained Transformer (GPT).	Google Bard AI засновано на мовній моделі для діалогових програм (LaMDA).
Архіви даних	Його відповіді базуються на даних до 2021 року.	Його відповіді базуються на інформації в реальному часі з Пошуку Google.
Багатомовність	Це багатомовний чат-бот, який підтримує щонайменше 50 мов. Таким чином він може швидше відповідати людською мовою.	Це також багатомовна система ШІ, яка підтримує 40 мов. Це допомагає Bard створювати людські реакції.

Google Bard AI — це досить нова система штучного інтелекту, запущена в перші місяці 2023 року. Отже, ще занадто рано визначати, чи створює вона сильну конкуренцію ChatGPT чи ні.

Докладніше питання порівняння ChatGPT з іншими моделями висвітлено в [11] – [12].

1.2.4 Правила написання підказок (prompts)

На самому базовому рівні ChatGPT передбачає текст на основі вхідних даних, які називаються підказками. Але щоб отримати найкращі результати, потрібно написати чітку підказку з достатнім контекстом.

Розробка підказок – це процес написання підказок, щоб максимізувати якість і релевантність відповіді. Якщо надана нами підказка не містить достатнього контексту або не написано так, щоб її легко було сприймати, швидше за все, це знизить якість відповіді.

Перше правило — бути чітким і конкретним. Це звучить як здоровий глузд, але нам потрібно переконатися, що підказка містить увесь необхідний контекст для створення бажаної відповіді. Друге правило — писати підказки лаконічними. Це означає видалення будь-якої непотрібної інформації чи мовних наворотів, які не створюють додаткового контексту для поточного завдання – ці додаткові елементи лише розбавлять важливу інформацію. Також використовуйте правильну граматику та правопис у підказках. ChatGPT використовує граматику для тлумачення завдань, тому треба переконатися, що запит написаний без помилок.

Під час створення підказок GPT також корисно вказати кількість слів для відповіді. Це потрібно, щоб не отримувати відповідь із 500 слів, коли потребували речення (або навпаки). Можна навіть використовувати діапазон прийнятної довжини.

Наприклад, якщо потрібна відповідь із 500 слів, можна надати підказку на зразок «Напишіть короткий виклад цієї статті на 500–750 слів». Це дає штучному інтелекту гнучкість для генерування відповіді в межах зазначеного діапазону. Також можна використовувати менш точні терміни, як-от «короткий» або «довгий».

Ще один спосіб підсилити підказки – навести приклади. У багатьох випадках це набагато швидший спосіб надання додаткового контексту, ніж написання вичерпного пояснення бажаних результатів.

Завдяки всім цим правилам та підказкам ChatGPT буде виводити саме те, що хочемо, і в потрібному форматі. Докладніше питання створення правильних підказок висвітлено в [10].

РОЗДІЛ 2 ДОКУМЕНТАЦІЯ КОДУ

Документація з розробки програмного забезпечення відіграє важливу роль у забезпеченні зрозумілості та ефективного використання програмного продукту для розробників, їхніх колег і користувачів. Види документації привертають увагу різних аудиторій, в залежності від їхніх ролей у процесі створення, перегляду або використання програмного забезпечення. Документування програмного забезпечення сприяє ефективному завершенню проектів розробниками і надає користувачам необхідний ресурс для роботи з програмним продуктом та усунення проблем.

2.1 Типи документації коду

Документація для розробки програмного забезпечення складається з різноманітних текстових матеріалів, які створюються розробником для відстеження процесу виробництва та використання програмного продукту від початку до кінця.

Розробники створюють різні форми програмної документації — одні для своєї власної довідки, а інші для полегшення взаємодії користувачів з програмним забезпеченням та надання допомоги їм у розумінні продукту та його використанні.

Розглянемо декілька типів документації коду та їх призначення:

1. Коментарі всередині коду.

Це невеликі пояснення або пояснювальні примітки, які розташовуються безпосередньо всередині коду. Вони допомагають розробникам краще зрозуміти функції або блоки коду, що підвищує його читабельність та сприяє швидшому розумінню того, що відбувається в програмі.

2. Документація на рівні функцій чи методів.

Це детальні описи функцій або методів, включаючи їх призначення, вхідні параметри, вихідні значення та приклади використання. Ця документація надає розробникам інформацію про те, як користуватися певними функціями або методами, спрощуючи їх використання та зменшуючи кількість помилок.

3. Документація на рівні модулів або бібліотек.

Це описи модулів або бібліотек, включаючи їх функціональні можливості, залежності, використання та приклади коду. Ця документація надає велику картину програмного продукту, допомагає розробникам швидше зорієнтуватися у проекті та сприяє повторному використанню коду.

4. Документація API.

Це опис публічного інтерфейсу програмного забезпечення (API), який вказує, як взаємодіяти з програмним продуктом через публічні методи або сервіси. Документація API надає важливу інформацію для розробників, які хочуть інтегрувати програмний продукт у свої проекти.

Ці типи документації коду спільно сприяють полегшенню читання, розумінню та використанню коду розробниками. Вони допомагають зменшити час, необхідний для освоєння програмного продукту, полегшують співпрацю між розробниками та підвищують продуктивність командної роботи. Правильно організована документація коду є важливою складовою успішного розроблення програмного продукту. Докладніше питання важливості документації коду висвітлено в [13].

2.2 Мови та інструменти для документації коду

Існує багато мов та інструментів, які можуть бути використані для документування коду. Кожна мова та інструмент має свої особливості,

переваги та недоліки. Нижче розглянемо кілька популярних мов та інструментів для документації коду:

1. Python Docstrings: Python має вбудовану можливість документування коду за допомогою docstrings. Docstrings – це рядки документації, які розміщуються на початку модуля, класу або функції і містять опис, приклади використання та іншу корисну інформацію. За допомогою докстрінгів можна автоматично створити документацію за допомогою інструментів, таких як Sphinx. Детальніше про Python Docstrings можна дізнатися з [14].

2. Sphinx: Sphinx є потужним інструментом для створення документації в різних форматах, включаючи HTML, PDF, EPUB і багато інших. Він використовує розмітку reStructuredText і може автоматично створювати заголовки, зміст, хід документації на основі докстрінгів і додаткових конфігурацій. Детальніше про Sphinx можна дізнатися з [15].

3. Javadoc: Javadoc є інструментом для документування коду на Java. Він використовує спеціальні коментарі, розміщені перед класами, методами, полями і іншими елементами коду, для генерації HTML-документації. Javadoc надає можливість створювати описи, параметри, повернені значення та інші деталі кожного елемента коду. Детальніше про Javadoc можна дізнатися з [16].

4. Doxygen: Doxygen є універсальним інструментом для документування коду на багатьох мовах програмування, включаючи C++, C#, Java, Python і багато інших. Він використовує спеціальні коментарі, подібні до Javadoc, для створення документації. Doxygen може генерувати HTML-документацію, зображення залежностей, діаграми класів та багато іншого. Детальніше про Doxygen можна дізнатися з [17].

Насправді, було розглянуто лише кілька прикладів мов та інструментів для документування коду. Вибір конкретної мови або інструменту залежить від потреб та вимог проекту.

2.3 Приклади документації коду

Кілька прикладів різних типів документації коду:

1. Документація всередині коду. Приклад на Python:

```python
def calculate_average(numbers):
    """
    Функція приймає список чисел та повертає їх середнє значення.

    Аргументи:
    numbers (list): Список чисел.

    Повертає:
    float: Середнє значення чисел.

    Приклад використання:
    >>> calculate_average([1, 2, 3, 4, 5])
    3.0
    """
    total = sum(numbers)
    average = total / len(numbers)
    return average
```

2. Документація на рівні функцій/методів. Приклад на Java:

```java
/**
 * Розраховує середнє значення чисел.
 *
 * @param numbers Масив чисел.
 * @return Середнє значення чисел.
 * @throws IllegalArgumentException Якщо передано порожній масив.
 * @since 1.0
 */
public static double calculateAverage(int[] numbers) throws
IllegalArgumentException {
    if (numbers.length == 0) {
        throw new IllegalArgumentException("Масив чисел не може бути
порожнім.");
    }

    int sum = 0;
    for (int number : numbers) {
        sum += number;
    }
    double average = (double) sum / numbers.length;
    return average;
}
```

3. Документація на рівні модулів або бібліотек. Приклад на javascript:

```javascript
/**
 * Модуль для роботи з датами і часом.
 *
 * @module DateTimeUtils
 */
```

```
/**
 * Повертає поточну дату у форматі "рік-місяць-день".
 *
 * @returns {string} Поточна дата.
 * @example
 * var currentDate = DateTimeUtils.getCurrentDate();
 * console.log(currentDate); // "2023-05-17"
 * @since 1.0.0
 */
function getCurrentDate() {
    // Код для отримання поточної дати
    return formattedDate;
}
```

Кожен з цих прикладів надає необхідну інформацію для розуміння та використання коду розробниками.

2.4 Набір кращих практик та рекомендацій для ефективного документування коду

Не мати жодних документів так само погано, як мати надмірну чи невідповідну документацію. Наведемо декілька основних правил створення корисної документації коду. Детальніше це питання висвітлено у [18].

Будьте простим та лаконічним. Дотримуйтесь принципу DRY (не повторюйте себе). Коментувати кожен окремий рядок коду необов'язково. Краще використовувати коментарі для пояснення складних або неочевидних аспектів, які потребують додаткового пояснення.

Використовуйте зрозумілий та однорідний стиль. Важливо встановити стандарти стилізації коду і дотримуватися їх у всьому проекті. Це полегшить читання та розуміння коду розробниками, особливо якщо в команді працює кілька людей.

Надання прикладів використання функцій або методів у документацію допомагає розробникам швидше зрозуміти, як користуватися певними компонентами коду. Це спрощує їх використання та зменшує кількість помилок.

Завжди оновлюйте документацію. Найкраще документувати код крок за кроком, як він написаний, замість того, щоб записувати коментарі до коду, який був написаний кілька місяців тому. Тим самим заощаджується час і документацію стає точною і повною. Використовуйте належне керування версіями, щоб відстежувати всі зміни в документі.

Документуйте будь-які зміни у вашому коді. Документування нових функцій або доповнень досить очевидно. Однак також потрібно документувати застарілі функції, фіксуючи будь-які зміни в продукті.

Використовуйте просту мову та правильне форматування. Документи коду зазвичай пишуться англійською мовою, щоб будь-який розробник міг прочитати коментарі, незалежно від його рідної мови. Найкращі практики для написання документації вимагають використання наказового способу, теперішнього часу та другої особи.

Використовуйте інструменти автоматизованого документування, вони можуть допомогти автоматично генерувати документацію на основі анотацій, коментарів або маркерів у коді. Окрім цього, пришвидшують та спрощують процес створення та підтримки документації.

Що стосується останнього, є багато інструментів, які зроблять важку роботу за вас. Такі інструменти зазвичай залежать від мови:

- Doxygen (C, C++, C#, Java, Objective-C, Python);
- GhostDoc (C#, Visual Basic, C++, JavaScript);
- Javadoc (лише Java);
- Docurium (Ruby);
- jsdoc (Javascript).

Ці рекомендації допоможуть забезпечити зрозумілість та ефективне використання коду розробниками. Докладніше питання ефективного документування коду висвітлено в [18] – [19].

РОЗДІЛ 3 РОЗРОБКА МОДЕЛІ АВТОМАТИЧНОГО СТВОРЕННЯ ДОКУМЕНТАЦІЇ КОДУ НА ОСНОВІ CHATGPT

3.1 Вибір інструментів і технологій для розробки моделі

У роботі було розроблено 2 застосунки: застосунок у вигляді CLI засобу та телеграм-бот для документування репозиторіїв. Розглянемо детальніше кожен з них.

1. Застосунок у вигляді CLI засобу

Цей застосунок дозволяє розробникам локально документувати свій код на мові Python. Розробник передає шлях до файлу або директорії як параметр і отримує задокументований код, який відповідає стандартам PEP 257.

Мовою програмування було обрано Python, як IDEs було обрано PyCharm. PyCharm – це інтегроване середовище розробки (IDE) для мови програмування Python. Воно надає розробникам широкий набір інструментів, включаючи функції автозаповнення, відладчик, систему керування версіями та багато іншого.

2. Телеграм-бот для документування репозиторіїв

Телеграм бот розроблений як зручний застосунок для документування коду на мові Python з GitHub. Користувач надсилає посилання на репозиторій, який потрібно задокументувати і в результаті отримує два посилення: на задокументований код та документацію, яка була створена використовуючи Doxygen.

Мовою програмування було обрано Python, як IDE було обрано Colab.

3.2 Опис алгоритму роботи та деталі реалізації застосунків

3.2.1 CLI засіб

Розглянемо основні модулі CLI засобу. Опишемо найголовніший модуль, де і відбувається генерація документації, відправка запиту до ChatGPT та обробка результату.

```
class DocumentationGenerator
```

Клас DocumentationGenerator надає можливість автоматичного генерації документації (docstrings) для кодових блоків мовою програмування Python з використанням штучного інтелекту. Нижче наведений опис основних методів класу:

1. `def __init__(self, language, api_key)`

В конструкторі ініціалізується об'єкт DocumentationGenerator з мовою програмування language та ключем API api_key.

2. `def _get_prompt(self, code)`

Цей метод повертає prompt для генерації документації на основі мови програмування. В даному випадку, для мови Python генерується prompt, який описує завдання, яке потрібно виконати, та вхідний код, який потрібно задокументувати.

3. `def generate_docs(self, code)`

Метод генерує документацію для заданого кодового блоку. Це основний метод, де і відбувається спілкування з ChatGPT. В методі отримується prompt на основі коду, створюється запит по OpenAI API, отримується відповідь, а потім повертається згенеровану документацію разом з кодом.

4. `def is_method(self, function_node, ast_tree)`

Метод перевіряє, чи є задана функція (function_node) методом класу, шляхом перевірки, чи є її батьківський вузол екземпляром ast.ClassDef.

5. `def generate_docs_for_block_and_change_node(self, tree, node, debug=False)`

Метод генерує документацію для кодового блоку (node) і змінює вузол node на новий вузол, що містить згенеровану документацію. Викликає метод generate_docs для генерації документації.

6. `def generate_docs_for_code_from_file(self, file_path, debug=False)`

Метод зчитує вміст файлу за вказаним шляхом (file_path), парсить код у вигляді AST, ітерується по AST і викликає метод generate_docs_for_block_and_change_node для кожного класу та функції в коді. Змінений код з документацією записується назад у файл.

7. `def generate_docs_for_code_from_dir(self, dir_path)`

Метод рекурсивно обробляє всі файли в директорії, викликаючи generate_docs_for_code_from_file для файлів з розширенням .py (для мови Python).

8. `def is_ignored_directory(self, directory_name)`

Метод перевіряє, чи є задана директорія в списку директорій для ігнорування. Використовується в `generate_docs_for_code_from_dir`.

Загалом, DocumentationGenerator є головною частиною програми, яка допомагає автоматизувати процес генерації документації на основі вхідного коду, забезпечуючи зрозумілу та докладну документацію для розробників.

Перейдемо до наступного модуля програми, який відповідає за генерацію doxygen документації у вигляді HTML та LaTeX. Doxygen – це потужний інструмент для автоматичної генерації документації з коментарів у вихідному коді. За допомогою цього модуля програми можна створювати професійну документацію, яка охоплює класи, функції, модулі та інші частини програми.

Клас DoxygenGenerator містить два методи:

1. `def generate_Doxyfile(self, path)`

Метод генерує файл «Doxyfile» для конфігурації Doxygen. Він використовує заданий шлях «path», щоб вказати на вхідні файли для

документації, встановлює параметри Doxygen, такі як генерація HTML та LaTeX, рекурсивний обхід каталогу та шляхи до вихідних файлів документації. Також створюються необхідні каталоги для збереження результатів документації.

2. `def generate_doxygen_documentation(self, path)`

Цей метод викликає generate_Doxyfile для згенерування конфігураційного файлу «Doxyfile», а потім запускає команду doxygen для генерації документації за заданим файлом «path». Використовується системний виклик «os.system» для запуску команди doxygen.

Цей клас дозволяє генерувати файл конфігурації Doxygen та запускати процес генерації документації з використанням цього конфігураційного файлу.

3.2.2 Телеграм-бот

Розглянемо основні модулі телеграм-боту. Першим модулем, який буде розглянуто є `class GitManager`.

Клас GitManager відповідає за керування репозиторіями Git і надає функціонал для отримання інформації про репозиторії, клонування репозиторію та отримання інформації про мови програмування, використані у репозиторії. Цей клас реалізує патерн проектування «Одинак» (Singleton) для забезпечення наявності лише одного екземпляру класу. Це здійснюється за допомогою статичного методу get_instance(), який повертає єдиний екземпляр класу.

Основні методи цього класу:

1. ```
   @staticmethod
   def get_instance()
   ```

Цей статичний метод повертає єдиний екземпляр класу GitManager та реалізує Singleton-поведінку.
   ```

2. `def __init__(self)`

Конструктор ініціалізує атрибути класу, які використовуються для зберігання URL репозиторію, назви репозиторію, локального шляху до директорії з кодом та локального шляху до директорії з документацією.

3. `def get_url(self)`

Повертає посилання на репозиторій

4. `def get_repo_name(self)`

Повертає назву репозиторія

5. `def get_local_path_to_code_dir(self)`

Повертає локальний шлях до коду на диску

6. `def get_local_path_to_docs_dir(self)`

Повертає локальний шлях до згенерованої Doxygen документації на диску

7.
```
@staticmethod
def is_git_repo(text)
```

Цей статичний метод перевіряє, чи є заданий URL репозиторієм Git на платформах GitHub або GitLab.

8. `def set_git_repo(self, text)`

Цей метод приймає URL репозиторію Git, перевіряє його валідність та встановлює значення атрибутів url та repo_name відповідно. Повертає True, якщо URL репозиторію валідний, інакше повертає False.

9. `def clone(self, pathToDir)`

Цей метод клонує репозиторій за допомогою команди git clone до гугл-диску та зберігає локальний шлях до директорії з кодом та директорії з документацією.

10. `def extract_owner_and_repo(self, url)`

Цей метод знаходить ім'я власника репозиторію та назву репозиторію з URL. Повертається кортеж (owner, repo).

11. `def get_repo_languages(self, url)`

Цей метод отримує мови програмування, використані у репозиторії, за допомогою запиту до GitHub API. Якщо параметр url не вказаний, використовується збережений URL репозиторію self.url. Повертається словник, який містить інформацію про мови програмування у репозиторії, або порожній словник у випадку помилки при запиті до API.

12. `def repo_contains_python_code(self, url=None)`

Цей метод перевіряє, чи містить репозиторій код на мові програмування Python. Якщо параметр url не вказаний, використовується збережений URL репозиторію self.url. Використовуючи метод get_repo_languages, отримується інформація про мови програмування у репозиторії. Перевіряється наявність мови Python у цьому словнику, і повертається True, якщо Python знаходиться у списку мов, інакше повертається False.

Наступним розглянемо: `class GoogleDriveManager`

Клас GoogleDriveManager відповідає за управління Google Drive. Цей клас також реалізує патерн проектування «Одинак», так як потрібно, щоб GoogleDriveManager був лише один.

Основні методи цього класу:

1. ```
   @staticmethod
   def get_instance():
   ```

Цей статичний метод повертає єдиний екземпляр класу GoogleDriveManager. Якщо екземпляр класу вже існує, метод повертає його. В іншому випадку, створюється новий екземпляр.

2. `def __init__(self)`

У конструкторі ініціалізуються необхідні параметри для роботи з Google Drive, зокрема під'єднується диск, проводиться аутентифікація, створюється об'єкт GoogleDrive для роботи з API Google Drive, і встановлюється локальний шлях до кореневої папки pathToDrive. Він перевіряє, чи вже існує екземпляр класу GoogleDriveManager. Якщо так, то
   ```

видається виключення. У протилежному випадку, екземпляр класу зберігається у змінній _instance для забезпечення Singleton-поведінки.

3. `def get_path_to_drive(self)`

   Метод повертає локальний шлях до кореневої папки Google Drive.

4. `def get_files_in_directory(self, directory)`

   Метод приймає шлях до папки directory і повертає список всіх файлів, що знаходяться в цій папці та її підпапках. Він рекурсивно переглядає всі файли та папки у вказаній директорії, додає файли Python до списку files і рекурсивно викликає себе для обробки підпапок.

5. `def is_directory(self, path)`

   Метод перевіряє, чи є path директорією.

6. `def is_ignored_directory(self, directory_name)`

   Метод перевіряє, чи є директорія з ім'ям directory_name ігнорованою. Він повертає True, якщо directory_name є в списку ігнорованих директорій, і False - якщо не є.

7. `def is_python_file(self, filename)`

   Метод перевіряє, чи має filename розширення .py (тобто, чи є це файл Python).

   Методи отримання ідентифікатора файлу:

8. `def get_file_id_from_file_list(self, file_path, file_list)`

   Метод повертає ідентифікатор файлу зі списку file_list за шляхом file_path.

9. `def get_file_id_from_path(self, file_path, codeOrDoc)`

   Метод повертає ідентифікатор файлу зі списку кореневої директорії Google Drive за шляхом file_path залежно від значення codeOrDoc. Якщо codeOrDoc має значення «doc», то шукається відповідний файл у папці «docs». Якщо codeOrDoc має значення «code», то шукається відповідний файл у кореневій директорії. Якщо файл не знайдено, видається виключення.

Методи створення архіву та отримання посилання на завантаження:

10. `def create_archive(self, folder_path, codeOrDoc)`

Метод створює архів для заданої директорії folder_path в Google Drive. Залежно від значення codeOrDoc («code» або «doc»), архів буде створений у відповідній папці («docs» або кореневій директорії). Метод повертає булеве значення та шлях до архіву.

11. `def get_download_link(self, folder_path, codeOrDoc)`

Метод повертає посилання на архів для заданої директорії folder_path в Google Drive. Метод спочатку створює архів за допомогою методу create_archive(), а потім отримує ідентифікатор файлу та створює посилання для завантаження. Якщо створення архіву не вдалося з першої спроби, метод спробує ще декілька разів.

12. `def authenticate(self)`

Метод виконує аутентифікацію користувача.

13. `def get_root_file_list(self)`

Метод повертає список файлів та папок у кореневій директорії Google Drive.

14. `def get_folder_by_name(self, file_list, folder_name)`

Метод повертає об'єкт папки зі списку file_list за назвою folder_name.

15. `def insert_permission(self, file_id)`

Метод встановлює права доступу для файлу з вказаним ідентифікатором file_id.

16. `def create_zip_file(self, repo_name, codeOrDoc)`

Метод створює архів для заданого репозиторію з назвою repo_name залежно від значення codeOrDoc ("code" або "doc"). В результаті повертається шлях до створеного архіву.

17. `def check_zip_file(self, zip_path)`

Метод перевіряє наявність архіву за вказаним шляхом zip_path та повертає True, якщо архів існує та False у протилежному випадку.

Що стосується найважливішого класу DocumentationGenerator – то його реалізація повторно використовується. Він призначений для автоматичної генерації документації коду на основі наданого вихідного коду. Реалізація класу DoxygenGenerator також повторно використовується.

Розглянемо алгоритм роботи телеграм бота:

1. Отримуємо посилання на репозиторій від користувача;
2. Перевіряємо чи відповідає посилання вимогам;
3. Клонуємо даний репозиторій на GoogleDrive;
4. Переглядаємо всі склоновані файли і для кожного файлу з розширенням .py викликаємо генерацію документації;
5. Генерація документації включає в себе розбиття коду на AST-дерево (абстрактне синтаксичне дерево), кожен вузол з якого відправляється до ChatGPT з заданим prompts;
6. Обробляємо відповідь від ChatGPT та перезаписуємо потрібні вузли AST-дерева;
7. Перетвоюємо AST-дерево назад у код та перезаписуємо відповідні файли;
8. Викликаємо генерацію doxygen документації, але перед цим ствоємо файл Doxyfile з відповідними налаштуваннями

3.3 Взаємодія з програмою

3.3.1 CLI засіб

Для того, щоб запустити процес документування коду, треба запустити команду `py generate-documentation.py --path $path$ --doxygen`, де `$path$` – шлях до директорії або файлу, який потрібно задокументувати. Параметр `--doxygen` не є обов’язковим, він використовується, щоб окрім документації коду, генерувалася ще й doxygen документація. Якщо ж

виклик відбувся з параметром `--doxygen`, то результат роботи буде знаходитися в директорії «docs» всередині проекту.

Окрім запуску документування вручну, розробник також має можливість використовувати автоматичне документування як частину процесу побудови проекту. Інструкції щодо цього наведені у розділі 3.4.

3.3.2 Телеграм-бот

Для того, щоб почати користуватися ботом, треба знайти його в телеграм та натиснути кнопку «Start». Початковий дизайн боту можна побачити на рис. 4.

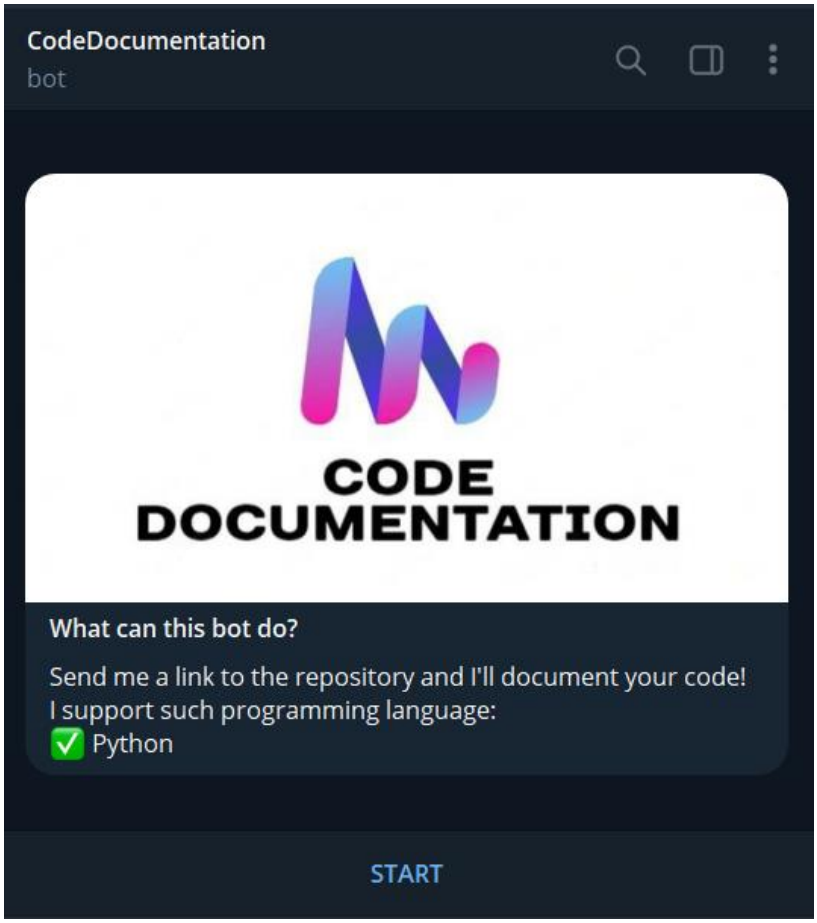

Рисунок 4 – Початок роботи з телеграм ботом

Після цього бот привітається і можна або одразу надсилати посилання на репозиторій, або натиснути на рекомендовану кнопку «Start Code

Documentation», як показано на рис. 5. Потім бот попросить надіслати йому посилання на репозиторій, що можемо бачити на рис. 6.

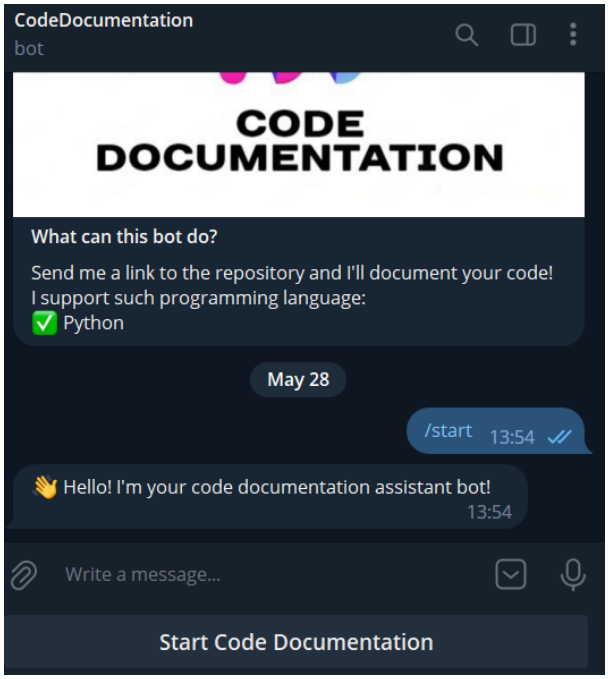

Рисунок 5 – Відповідь чат-боту після натискання кнопки «Start»

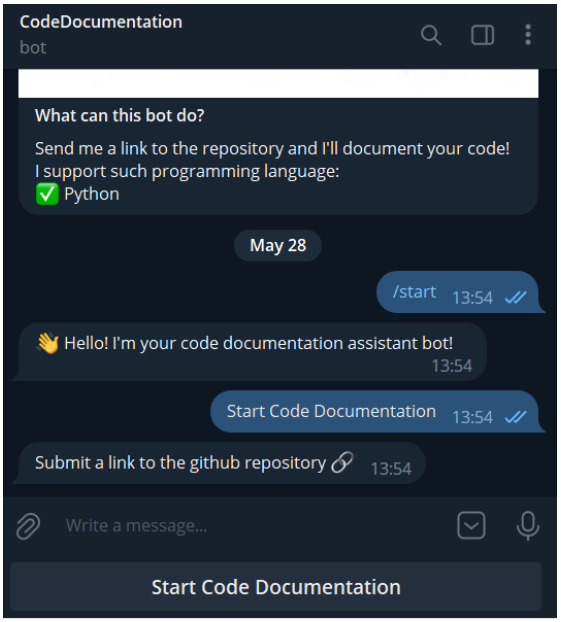

Рисунок 6 – Відповідь чат-боту після натискання кнопки «Start Code Documentation»

Тепер можемо надіслати боту посилання на репозиторій та чекати на результат. Результат роботи показаний на рис. 7. Бот успішно

задокументував код та надіслав користувачу два посилання: на архів з задокументованим кодом та на архів з документацію doxygen.

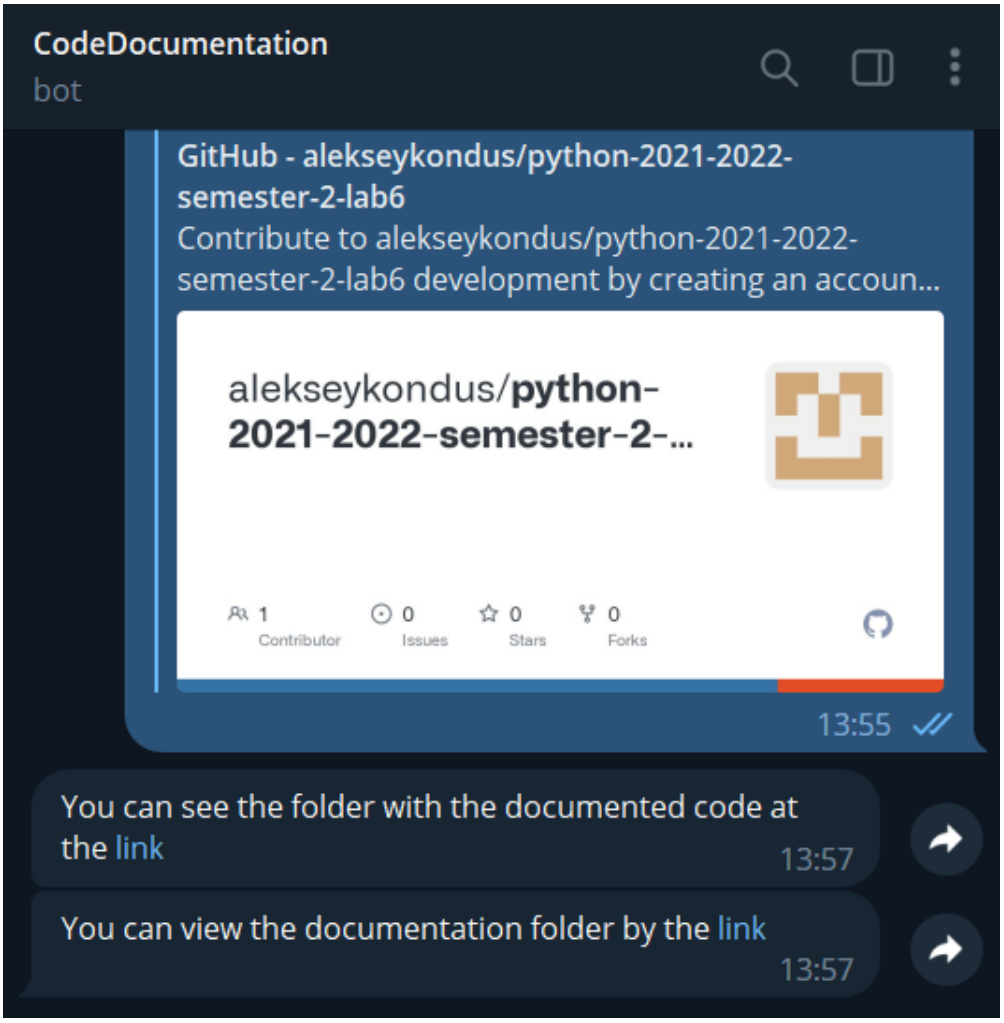

Рисунок 7 – Успішна робота чат-боту

Окрім цього, бот може обробляти некоректні посилання:

– Якщо користувач надсилає посилання, яке не відноситься до GitHub або GitLab, він отримує повідомлення «Please send a link to the GitHub repository»;
– Якщо користувач надсилає посилання з неправильним ім'ям власника (owner) або неправильною назвою користувача, бот відправляє повідомлення «Check if the link is correct»;
– Якщо користувач надсилає репозиторій, в якому відсутній код мовою Python, то він отримує повідомлення: «I can only document Python code. The repository has no Python code». Варто зазначити, що для перевірки

наявності коду на мові Python у репозиторії використовується модуль requests для виконання HTTP-запитів до API GitHub.

Побачити обробку некоректних посилань можемо на рис. 8.

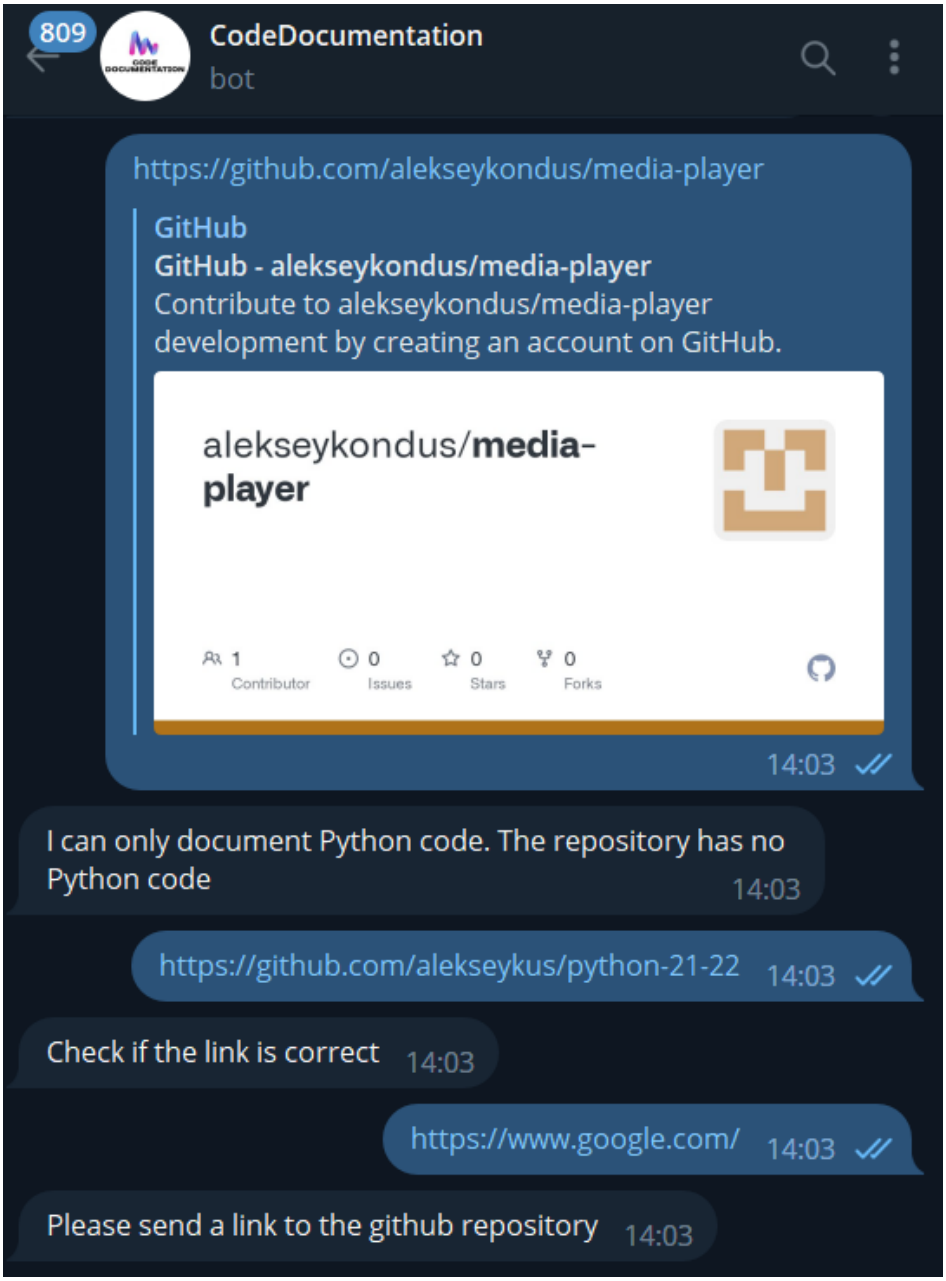

Рисунок 8 – Обробка ботом некоректних посилань

3.4 Аналіз згенерованої документації

Приклад згенерованої документації для класу DocumentationGenerator, який був описаний в 3.2.1, можемо побачити у

додатку А на рисунку А.1. Повну згенеровану документацію проекту разом з кодом для CLI засобу можемо побачити за посиланням: https://github.com/alekseykondus/GraduateWork

Розглянемо декілька методів цього класу, які були задокументовані CLI засобом. На рис. 10 можемо побачити приклад згенерованої документації для методу _get_prompt. Вона надає короткий опис функції _get_prompt, вказує на параметр code, який представляє собою блок коду, що потребує документування. Також, згідно з документацією, метод повертає повідомлення (prompt message) із заданим кодом.

```
def _get_prompt(self, code):
    """Generates prompt message and returns it with given code block for OpenAI API.

    Args:
    code (str): The code block to be documented

        Returns:
        (str): The prompt message with the given code.
        """
```

Рисунок 9 – Приклад згенерованої документації для методу _get_prompt

На рис. 10 можемо побачити приклад згенерованої документації для методу is_ignored_directory. Наведена документація пояснює, що цей метод, призначений для перевірки, чи збігається аргумент directory_name з іменем директорії зі списку ignored_dirs. Якщо збігається, то метод повертає True, що означає, що дана директорія повинна бути проігнорована. В іншому випадку, метод повертає False, що означає, що директорія не повинна бути ігнорована.

```
def is_ignored_directory(self, directory_name):
    """Returns True if the directory_name argument matches a directory name in the ignored_dirs list.

    Args:
    directory_name (str): The name of the directory in question.

    Returns:
    bool: True if the directory is to be ignored, False otherwise.
    """
```

Рисунок 10 – Приклад згенерованої документації для методу is_ignored_directory

Проаналізувавши наведені приклади, можемо зробити висновок, що згенерована документація виглядає зрозумілою та інформативною. Вона надає користувачеві необхідну інформацію щодо призначення та функціональності методів. Завдяки цьому, користувач може легко зрозуміти, як правильно їх використовувати та яку роль вони відіграють у функціонуванні програми.

Проаналізуємо часові затрати на генерацію документації та кількість API ресурсів, які використовуються під час роботи.

Варто зазначити, що під час генерації використовувалась модель gpt-3.5-turbo, вартість якої складає $0.002 за кожні 1000 токенів (токени поділяють текст на окремі частини). На таблиці 4 можемо побачити час, який був витрачений на документування коду та створення doxygen документації для CLI засобу, розробленого у роботі, а також собівартість документування. Середній час роботи для документування коду CLI засобу складає приблизно 4 хвилини, а середня собівартість документування дорівнює 0.63 грн. Можемо зробити висновок, що використання бота значно прискорює цей процес та знижує витрати порівняно з ручним написанням документації розробником.

Таблиця 4 – Аналіз згенерованої документації

Тест №	Час виконання (секунди)	Кількість використаних токенів	Ціна документування (центи)	Ціна документування (грн)
1	219.99	6675	1.3674	0.5
2	256.75	6412	1.335	0.49
3	464.94	12615	2.3488	0.87
4	184.66	6181	1.4566	0.54
5	188.66	6207	1.2362	0.46
6	324.08	10117	2.0318	0.75
7	343.22	10296	1.233	0.46
8	203.2	6619	2.0592	0.76
9	205.68	6421	2.0878	0.77
10	192.92	6266	1.7734	0.65

3.4 Впровадження моделі в розробку програмного продукту

Розроблений CLI засіб можна легко інтегрувати в процес автоматичної побудови проекту. Це дозволить забезпечити автоматичну

генерацію документації під час кожної побудови проекту, що вкрай полегшить процес розробки для всієї команди, адже в процесі розробки вже буде документація, яку можна використовувати для кращого розуміння коду, раннього виявлення помилок в архітектурі, ознайомлення нових учасників команди, тощо. В результаті отримуємо завжди актуальну документацію, бо автоматично підтримується синхронізованою з кодом.

Для впровадження моделі можна, наприклад, створити скрипт або налаштувати систему побудови (build system), яка виконує необхідні кроки для побудови проекту, а в останній частині додає команду для генерації документації за допомогою розробленого CLI засобу. Це дозволить автоматично створювати актуальну документацію кожного разу, коли проект будується.

Наведемо приклад вбудови CLI засобу по документації коду в PyCharm, популярне інтегроване середовище розробки для мови програмування Python.

1. Відкрийте проект у PyCharm і перейдіть до налаштувань проекту;
2. У розділі «Run/Debug Configuration» оберіть Before Launch;
3. Клікніть на плюс (+) та оберіть "Run External Tools", щоб додати новий зовнішній інструмент;
4. У новому вікні знову клікніть на плюс (+) та введіть назву для інструменту, наприклад "Generate Documentation";
5. У полі «Program» введіть шлях до файлу з CLI засобом documentation_generator.exe;
6. У полі «Arguments» треба вказати --path $ContentRoot$ --doxygen. Це означає, що CLI засобу буде переданий параметром шлях до вашого проекту. Параметр --doxygen є необов'язковим, його треба вказувати, якщо потрібно генерувати ще й doxygen документацію;
7. Натисніть «OK», щоб зберегти налаштування зовнішнього інструменту.

Таким чином, можна легко інтегрувати CLI засіб у своє середовище розробки і використовувати його для автоматичної генерації документації під час роботи проектів, наприклад, у PyCharm. Аналогічним чином можна інтегрувати документування коду і у інші системи побудови проектів та IDE.

3.5 Порівняння з існуючими засобами документації коду

Варто зазначити, що повних аналогів даного засобу немає, адже будь-які інші інструменти лише допомагають створювати документацію коду, але не створюю її за розробників. Розглянемо деякі з таких інструментів:

1. AutoDocstring – доступний для деяких інтегрованих середовищ розробки (IDE), таких як PyCharm або Visual Studio Code. AutoDocstring дозволяє швидко створювати шаблони docstrings для функцій, методів, класів тощо, використовуючи вхідні параметри та типи даних. Він автоматично генерує заголовки, описи параметрів та повернутих значень на основі вхідного коду. Тобто цей інструмент створює шаблон документації, але цей шаблон потрібно заповнити розробникам. Детальніше AutoDocstring розглядається у [20].

2. Sphinx-apidoc – популярний генератор документації для проектів на Python для створення гарної документації з різними вихідними форматами (HTML, LaTeX тощо). Він аналізує модулі, класи та функції, отримує інформацію про їх структуру та описи, і створює вихідний файл документації у вигляді HTML, PDF або інших форматів. Але розробник все ще повинен забезпечити наявність належної документації та детально описати складові свого коду. Детальніше Sphinx-apidoc висвітлено у [21].

Ці інструменти можуть спростити процес написання docstrings, але варто зазначити, що для їх правильної роботи розробнику потрібно самостійно написати вичерпну документацію в коді, забезпечити її точність

та повноту. Якщо порівнювати з розробленим CLI засобом чи телеграм ботом, то в них використовується штучний інтелект та нейромережі, щоб автоматично генерувати документацію коду без необхідності написання коментарів вручну або документаційних рядків. Засоби аналізують структуру коду, розуміють його функціональність і здатні згенерувати вичерпну документацію, яка відповідає стандартам PEP 257, що економить час і зусилля розробників.

ВИСНОВКИ

У роботі було розроблено два застосунки (CLI засіб та чат-бот) для автоматизованого створення документації коду з використанням функціоналу ChatGPT. Ціль полягала в розробці інструменту, який допоможе програмістам швидко та ефективно створювати документацію для свого коду, зменшуючи витрати часу та зусиль.

У ході роботи було проведено аналіз існуючих методів документування коду та порівняно їх зі створеним інструментом. Було підтверджено, що традиційні методи, такі як використання коментарів або спеціальних розміток, вимагають від програміста великої кількості ручної роботи і можуть бути часозатратними.

Застосування розробленого застосунку з використанням ChatGPT дозволяє автоматизувати процес створення документації. Так як розроблені застосунки використовують ШІ у своїй основі, то можуть аналізувати код і генерувати відповідні описи, приклади використання та іншу необхідну документацію. Однією з переваг цього підходу є те, що ШІ може розуміти код і зв'язки між різними його частинами. Це дозволяє отримати більш точну та контекстуальну документацію, яка відповідає функціональності та особливостям конкретного коду. Це полегшує завдання програмістам та допомагає зберегти час, який може бути використаний можна на більш пріоритетні завдання розробки.

Також у роботі було зазначено, що розроблений CLI засіб можна легко інтегрувати в системи побудови проекту, що показує гнучкість та розширюваність розробленого застосунку. Така інтеграція дозволяє забезпечити актуальність документації на кожному етапі розробки проекту. Кожного разу, коли виконується процес побудови проекту, засіб буде аналізувати код і оновлювати документацію з урахуванням поточного стану проекту.

Однак, варто пам’ятати, що використання моделі ChatGPT також має свої обмеження, що вимагає незначної перевірки документації з боку програміста.

Також важливо пам'ятати про забезпечення безпеки та конфіденційності документації. Оскільки весь код, який буде надісланий до моделі ChatGPT, стає доступним для аналізу та зберігається у системі, потрібно бути обережним щодо надання конфіденційної або чутливої інформації.

Загалом, розроблені інструменти для автоматизованого створення документації коду з використанням функціоналу ChatGPT можуть бути цінним доповненням для роботи фахівців у галузі розробки програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Schmidhuber J. Annotated History of Modern AI and Deep Learning / Jürgen Schmidhuber., 2022. – 75 с.

2. Deep Learning – A Tutorial for Data Scientists. [Електронний ресурс] – Режим доступу до ресурсу: https://www.datacamp.com/tutorial/tutorial-deep-learning-tutorial

3. Introducing ChatGPT. [Електронний ресурс] – Режим доступу до ресурсу: https://openai.com/blog/chatgpt

4. A.M. Jasmine Hashana, P. Brundha, Mohamed Uvaze Ahamed Ayoobkhan, Fazila S, Deep Learning in ChatGPT – A Survey, in: 2023 7th International Conference on Trends in Electronics and Informatics (ICOEI), IEEE, 2023

5. How does ChatGPT work? [Електронний ресурс] – Режим доступу до ресурсу: https://zapier.com/blog/how-does-chatgpt-work/

6. Xiao F., X C. Inner Life of an AI: A Memoir by ChatGPT. Independently Published, 2022.

7. M. Abdullah, A. Madain, Y. Jararweh ChatGPT: fundamentals, applications and social impacts, 2022 Ninth International Conference on Social Networks Analysis, Management and Security (SNAMS), IEEE (2022, November)

8. Understanding the Transformer Architecture that runs ChatGPT. [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/pulse/understanding-transformer-architecture-nlp-evan-dunbar

9. A. Gillioz, J. Casas, E. Mugellini, O. Abou Khaled, Overview of the transformer-based models for nlp tasks, in: 2020 15th Conference on Computer Science and Information Systems (FedCSIS), IEEE, 2020

10. 15 Rules For Crafting Effective GPT Chat Prompts, [Електронний ресурс] – Режим доступу до ресурсу: https://expandi.io/blog/chat-gpt-rules/

11. М. Бровінська Google відкрив чат-бот Bard AI для всіх охочих. [Електронний ресурс] – Режим доступу до ресурсу: https://dev.ua/news/venom-1683041675

12. Chat GPT VS. Other Language Models: Who Do You Think Wins? [Електронний ресурс] – Режим доступу до ресурсу: https://unstop.com/blog/chat-gpt-vs-other-language-models-a-comparison

13. Code documentation: benefits, challenges, and tips for success. [Електронний ресурс] – Режим доступу до ресурсу: https://swimm.io/learn/code-documentatio.n/code-documentation-benefits-challenges-and-tips-for-success/

14. PEP 257 – Docstring Conventions, 2001, [Електронний ресурс] – Режим доступу до ресурсу: https://peps.python.org/pep-0257/

15. Офіційний сайт Sphinx, [Електронний ресурс] – Режим доступу до ресурсу: https://www.sphinx-doc.org/en/master/

16. Java Platform, Standard Edition Javadoc Guide, [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/en/java/javase/13/javadoc/javadoc.html

17. Doxygen. Generate documentation from source code, [Електронний ресурс] – Режим доступу до ресурсу: https://www.doxygen.nl/index.html

18. SOURCE CODE DOCUMENTATION BEST PRACTICES. [Електронний ресурс] – Режим доступу до ресурсу: https://easternpeak.com/blog/source-code-documentation-best-practices/

19. Spinellis D, Code Documentation, IEEE Software, 2010. Vol. 27, no. 4.

20. autoDocstring – Python Docstring Generator, [Електронний ресурс] – Режим доступу до ресурсу: https://marketplace.visualstudio.com/items?itemName=njpwerner.autodocstring

21. sphinx-apidoc, [Електронний ресурс] – Режим доступу до ресурсу: https://www.sphinx-doc.org/en/master/man/sphinx-apidoc.html

ДОДАТОК А

Приклад згенерованої документації

```
class DocumentationGenerator:
    """This class is responsible for generating documentation for Python code.
    Its methods can be used for individual code blocks as well as entire directories of code.

    Args:
    language (str): The programming language of the code to be documented ('Python').
    OpenAI API_key: The API key for authenticating OpenAI account.

    Attributes:
    language (str): The programming language of the code to be documented ('Python').
    ignored_dirs (list): A list of directory names to be ignored during documentation generation.

    Methods:
    get_ignored_dirs: Returns the list of directories to be ignored.
    generate_docs: Generates the documentation for a given code block.
        - code (str): The code block to be documented.
        return (str): The generated documentation.
    _get_prompt: Generate prompt message necessary for OpenAI chat completion API for generating docs
        - code (str): The code block we're generating a prompt for.
        return (str): The complete prompt message.
    is_method: Returns True if the function_node is a method within a class.
        - function_node: The function node in the AST.
        - ast_tree: The full AST of the code.
    generate_docs_for_block_and_change_node: Generates docstrings for a block of code and replaces the existing node with new node.
        - tree: The full AST of the code.
        - node: The code block node to generate documentation for.
        - debug (bool, optional): If True, gives verbose output during execution.
    generate_docs_for_code_from_file: Generates documentation for code in a file.
        - file_path (str): The path to the file containing the code to be documented.
        - debug (bool, optional): If True, gives verbose output during execution.
    is_ignored_directory: Returns True if the directory_name argument matches any directory names in the ignored_dirs attribute.
        - directory_name (str): The name of the directory in question.
    generate_docs_for_code_from_dir: Recursively generates documentation for code in all files in a given directory.
        - dir_path (str): The path to the top level dir in the directory structure containing the code to be documented.
    """
```

Рисунок А.1 – Приклад згенерованої документації для класу DocumentationGenerator