НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Спосіб аналізу програмного коду, згенерованого нейромережею ChatGPT

Виконав (-ла): студент (-ка) 4 курсу, групи ІВ-93
(шифр групи)

Манчук Максим Вячеславович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент Міщенко Л.Д. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ст. викладач Виноградов Ю.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н Шимкович В. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2023 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Манчука Максима Вячеславовича

1. Тема проєкту *Спосіб аналізу програмного коду, згенерованого нейромережею ChatGPT*
керівник проєкту _____асистент_Міщенко Людмила Дмитрівна_____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 31 травня 2023 року №2101
2. Термін здачі студентом закінченого проєкту _____*червня 2023 р.*
3. Вихідні дані до проєкту *технічна документація. теоретичні дані*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Опис предметної області, дослідження методики генерації та аналізу програмного коду за допомогою нейромереж
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) *функціональна діаграма застосунку (принципова схема), діаграма класів (функціональна схема), структура програмного забезпечення (структурна схема)*

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормконтроль	Виноградов Ю. М.		

7. Дата видачі завдання_____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*21.01.2023*	
2.	*Вивчення та аналіз завдання*	*05.05.2023*	
3.	*Розробка архітектури та загальної структури системи*	*12.05.2023*	
4.	*Розробка структур окремих підсистем*	*12.05.2023*	
5.	*Програмна реалізація системи*	*19.05.2023*	
6.	*Оформлення пояснювальної записки*	*5.06.2023*	
7.	*Захист програмного продукту*		
8.	*Передзахист*		
9.	*Захист*	*23.06.2023*	

Студент-дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

АНОТАЦІЯ

В бакалаврському дипломному проєкті реалізовано спосіб генерації програмного коду, використовуючи нейромережу ChatGPT та його аналіз і оцінка, використовуючи відповідні інструменти для аналізу програмного коду. Схема розробленої системи ілюструється. Наведені теоретичні та експериментальні оцінки ефективності реалізованої системи

Ця програма є ефективним інструментом для створення запитів, які призначені для генерації специфічного коду на мові програмування Python. Вона також забезпечує автоматичну перевірку та оцінку якості згенерованого коду.

ANNOTATION

This bachelor's degree project has implemented a method for generating program code using the ChatGPT neural network and its analysis and evaluation using relevant tools for code analysis. The scheme of the developed system is illustrated. Theoretical and experimental evaluations of the effectiveness of the implemented system are presented.

This program is an efficient tool for creating requests aimed at generating specific code in the Python programming language. It also provides automatic analysis and evaluation of the quality of the generated code.

справки	Форма	Значення	Найменування	Кіл. листів	№ екземпляра	Додаток
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Спосіб аналізу програмного	3		
			коду, згенерованого			
			нейромережею ChatGPT			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Спосіб аналізу програмного	64		
			коду, згенерованого			
			нейромережею ChatGPT			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Спосіб аналізу програмного	1		
			коду, згенерованого			
			нейромережею ChatGPT			
			Функціональна діаграма			
			застосунку (Принципова			
			схема)			
	А4	***ІАЛЦ.467200.005 Д2***	Спосіб аналізу програмного	1		
			коду, згенерованого			
			нейромережею ChatGPT			
			Діаграма класів			
			(Функціональна схема)			
	А4	***ІАЛЦ.467200.006 Д3***	Спосіб аналізу програмного	1		
			коду, згенерованого			
			нейромережею ChatGPT			
			Структура програмного			
			забезпечення(Структурна			
			схема)			
	А4	***ІАЛЦ.467200.007 Д4***	Спосіб аналізу програмного	3		
			коду, згенерованого			
			нейромережею ChatGPT			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА
Зм.	***Лист***	**№ докум.**	***Підп***	***Дата***	
Розроб		Манчук М.В			*Спосіб аналізу програмного коду, згенерованого нейромережею ChatGPT* Опис альбому
Перев		Міщенко Л. Д.			

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського
ФІОТ
ІВ-93

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Спосіб аналізу програмного коду, згенерованого нейромережею ChatGPT*»

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання для обраного рішення полягає в розробці системи генерації програмного коду, використовуючи нейромережі, та аналіз і оцінка якості згенерованого коду.

Областю застосування цієї системи є програмування та розробка програмного забезпечення у більшості сферах, які використовують системи, написані на мові програмування Python.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», який був затверджений факультетом "Інформатики та обчислювальної техніки" кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут імені Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою дипломної роботи є розробка системи аналізу згенерованого коду, дослідження ефективного способу рішення існуючих проблем.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий і інтуїтивно-зрозумілий інтерфейс системи.
- Надати можливість користувачам створювати запити для генерації коду на мові програмування Python.
- Надати можливість користувачам власноруч конфігурувати специфічні параметри генерації програмного коду.
- Надати можливість зберегти згенерований програмний код та його аналіз.
- Надати вичерпну та зрозумілу документацію для розробленого продукту.

5.2. Вимоги до програмного забезпечення

- ОС Windows 7 або вище.
- PyCharm IDE 2022 або вище.

5.3. Вимоги до апаратної частини

- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми проєкту	16.01.2023-21.01.2023
Вивчення та аналіз завдання	23.01.2023-05.05.2023
Розробка архітектури та загальної структури системи	01.05.2023-12.05.2023
Розробка структур окремих частин системи	08.05.2023-12.05.2023
Програмна реалізація системи	15.05.2023-19.05.2023
Виправлення помилок	22.05.2023-5.06.2023
Оформлення пояснювальної записки	01.05.2023-12.05.2023

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Спосіб аналізу програмного коду, згенерованого нейромережею ChatGPT*»

Київ – 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CI	(Continuous integration) Неперервна інтеграція
CD	(Continuous delivery) Неперервна доставка
HTML	(HyperText Markup Language) Мова розмітки гіпертексту
GPT	(Generative Pre-Trained Transformer) Генеративний попередньо навчений перетворювач
API	(Application Programming Interface) Прикладний програмний інтерфейс
IDE	(Integrated Development Environment) Інтегроване середовище розробки
PEP	(Python Enhancement Proposal) Пропозиції по покращенню Python
HTTP	(Hypertext Transfer Protocol) Протокол передачі гіпертекстових документів

ВСТУП

Кожен програміст щодня зіштовхується з різноманітними завданнями, пов'язаними з програмним кодом. Під час розробки програмного забезпечення важливо не тільки створити працездатну програму, але й забезпечити її якість, безпеку та ефективність. Але чим більший програмний продукт, тим більше часу та ресурсів витрачається на його вдосконалення.

У зв'язку з цим виникає потреба у нових підходах, які можуть автоматизувати та спростити процес розробки програмного забезпечення. Нейромережі, що базуються на штучному інтелекті, мають великий потенціал у вирішенні цих завдань. Застосування нейромереж для генерації коду може допомогти зменшити зусилля розробників, скоротити час на розробку нового функціоналу та забезпечити високу якість створеного коду.

Однак у наш час такий варіант не є досконалим. Нейромережі, хоч і можуть розробляти програмний код, не завжди виконують своє завдання правильно. Проблеми можуть виникати навіть на початкових стадіях — при створенні простих алгоритмів. Тому весь код, згенерований нейромережею, потрібно детально аналізувати.

В традиційних підходах аналізу коду розробники мають велику кількість ручних завдань, таких як перевірка наявності помилок, забезпечення відповідності стандартам кодування та оцінка його ефективності. Цей процес може бути часо- та ресурсозатратним, особливо при роботі з великими проєктами. Тому актуальним рішенням буде створення скрипту, що буде автоматично аналізувати програмний код, згенерований нейромережею.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ СПОСОБІВ АНАЛІЗУ ТА ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ

1.1 Способи аналізу програмного коду

1.1.1 Критерії аналізу програмного коду

Код можна аналізувати за декількома критеріями, зокрема:

1.Читабельність: Чи зрозуміло і легко читати код? Чи використовуються зрозумілі імена змінних, функцій і класів? Чи є відповідні коментарі і форматування?

2.Ефективність: Чи оптимізований код для досягнення максимальної продуктивності? Чи використовуються ефективні алгоритми і структури даних? Чи існують зайві операції або повільні запити до бази даних?

3.Масштабованість: Чи може код легко масштабуватися для розширення функціональності або обробки більшого обсягу даних? Чи є в коді повторюваність або тісна залежність між компонентами?

4.Надійність: Чи є в коді помилки або вразливості, що можуть призвести до некоректної роботи або зламу системи? Чи існують в коді достатні перевірки на помилки та обробка виключень?

5.Можливість тестування: Чи є в коді достатні тести для перевірки правильності його роботи? Чи може код бути легко тестований і автоматизований?

6.Переносність: Чи може код працювати на різних платформах і середовищах? Чи є в коді залежності від конкретних характеристик апаратного забезпечення або операційної системи?

7.Документація: Чи існує достатня документація, яка описує функціональність і особливості коду? Чи є в коді коментарі, докстрінги або довідкові файли?

Ці критерії допомагають оцінити якість коду і визначити його сильні та слабкі сторони. Однак, важливо враховувати контекст і вимоги проєкту при аналізі коду.

1.1.2 Сервіс SonarQube

SonarQube - це відкрите програмне забезпечення для автоматичної оцінки якості коду і аналізу технічного боргу. Він надає засоби для виявлення потенційних проблем, помилок, вразливостей безпеки, дублікатів коду та інших недоліків в програмному коді.

Рисунок 1.1. - Логотип сервісу Sonar – видавця продукту SonarQube - Самокерованого інструменту статичного аналізу для безперервної перевірки кодової бази [1]

Основні функції та можливості SonarQube [1] включають:

1.Аналіз якості коду: SonarQube проводить статичний аналіз програмного коду для виявлення потенційних проблем і відображає їх в зручному інтерфейсі.

Це допомагає командам розробників виявляти і виправляти проблеми в коді, що сприяє покращенню його якості.

2.Дашборди та звіти: SonarQube надає гнучкі дашборди та звіти, які демонструють стан якості коду проєкту. Це дозволяє командам візуалізувати дані аналізу, відстежувати покращення та моніторити технічний борг.

3.Виявлення дублікатів: SonarQube здатний виявляти дублікати коду, тобто фрагменти коду, які повторюються в проєкті. Це допомагає зменшити повторення коду та покращити його підтримку і читабельність.

4.Перевірка вразливостей безпеки: SonarQube здатний виявляти потенційні вразливості безпеки в програмному коді, такі як недостатня обробка виключних ситуацій, вразливості XSS або SQL-ін'єкції. Це допомагає забезпечити безпеку програмного продукту.

5.Інтеграція з інструментами CI/CD: SonarQube може бути інтегрований з різними інструментами для забезпечення неперервної інтеграції та постачання (CI/CD), такими як Jenkins, GitLab, Azure DevOps тощо. Це дозволяє автоматично виконувати аналіз коду на кожному етапі розробки і сприяє швидкому виявленню проблем.

SonarQube підтримує багато мов програмування, включаючи Java, C#, JavaScript, Python, PHP та інші. Він є популярним інструментом серед розробників та команд, орієнтованих на підвищення якості коду та забезпечення стабільного розвитку проєктів.

1.1.3 Сервіс CodeClimate

CodeClimate - це інструмент для аналізу якості коду, який надає розробникам засоби для виявлення проблем, недоліків і незгод у програмному коді. Він допомагає командам розробників покращити якість коду, зменшити технічний борг і підтримувати стабільний розвиток проєктів.

Основні особливості та можливості CodeClimate [2] включають:

1.Статичний аналіз коду: CodeClimate виконує статичний аналіз програмного коду для виявлення проблем, таких як неправильне використання синтаксису, недотримання стандартів коду, вразливості безпеки, дублікати коду та інші потенційні проблеми. Він застосовує різні правила аналізу, які допомагають знайти і покращити якість коду.

2.Інтеграція з системами контролю версій: CodeClimate може бути інтегрований з системами контролю версій, такими як GitHub, Bitbucket, GitLab тощо. Це дозволяє автоматично виконувати аналіз коду при кожному коміті або пул-реквесті і отримувати повідомлення про виявлені проблеми безпосередньо в системі контролю версій.

3.Розширені метрики та статистика: CodeClimate надає розширені метрики та статистику про якість коду. Це включає такі показники, як покриття тестами, складність коду, завантаженість технічного боргу, тривалість аналізу тощо. Ці дані допомагають командам візуалізувати та моніторити якість коду в проєкті.

4.Рекомендації щодо виправлення: CodeClimate надає рекомендації щодо виправлення виявлених проблем. Він може автоматично генерувати коментарі або замітки з пропозиціями щодо поліпшення коду. Це допомагає розробникам швидко знайти та виправити проблеми в коді.

Рисунок 1.2. - Логотип компанії та продукту CodeClimate [2]

5.Перегляд коду та спільна робота: CodeClimate забезпечує можливість перегляду коду, коментування та обговорення проблем безпосередньо в інтерфейсі. Це сприяє спільній роботі команди розробників над поліпшенням якості коду.

CodeClimate підтримує різні мови програмування та інтегрується з багатьма інструментами розробки.

Це дозволяє використовувати його в різних проєктах та отримувати корисну інформацію про якість коду.

1.1.4 Сервіс Coverity

Coverity - це комерційний інструмент для статичного аналізу програмного коду, який спеціалізується на виявленні помилок, вразливостей та дефектів в програмах. Він використовується для поліпшення якості коду, зменшення ризиків і підвищення надійності програмних продуктів.

Рисунок 1.3. - Логотип інструменту аналізу коду Coverity [3]

Основні особливості та можливості Coverity [3] включають :

1.Статичний аналіз коду: Coverity виконує глибокий статичний аналіз програмного коду, що охоплює різні мови програмування, такі як C, C++, Java, C#, JavaScript тощо. Він виявляє потенційні проблеми, такі як невизначені вказівники, некоректне керування пам'яттю, вразливості безпеки, некоректні логічні операції, недостатню обробку помилок та інші.

2.Виявлення дефектів та вразливостей: Coverity ідентифікує дефекти, помилки програмування та вразливості безпеки, які можуть призвести до відмов системи або порушення безпеки. Це включає такі проблеми, як переповнення буфера, неправильна перевірка введених даних, використання невідповідних функцій тощо.

3.Інтеграція з інструментами розробки: Coverity може бути інтегрований з іншими інструментами розробки, такими як системи контролю версій і засоби автоматизації збирання та тестування. Це дозволяє автоматично виконувати аналіз коду при зміні або побудові проєкту та отримувати повідомлення про виявлені проблеми.

4.Графічне представлення результатів: Coverity надає графічне середовище для візуалізації результатів аналізу. Це дозволяє розробникам швидко переглянути та аналізувати знайдені проблеми, перевірити їх контекст та зв'язки з кодом.

5.Автоматична підтримка керування проблемами: Coverity надає можливості для керування проблемами, включаючи призначення, відстежування та контроль стану виправлень. Це допомагає командам розробників ефективно вирішувати знайдені проблеми та забезпечувати їх виправлення.

Coverity є потужним інструментом для аналізу програмного коду, який допомагає виявляти та виправляти потенційні проблеми в програмах, забезпечуючи підвищену якість та надійність розроблюваних продуктів.

1.1.5 Розширення PyLint

PyLint є інструментом статичного аналізу коду для мови програмування Python. Він призначений для виявлення потенційних проблем, стилістичних недоліків та вразливостей в програмному коді.

Основні особливості та можливості PyLint [4] включають:

1.Виконання статичного аналізу: PyLint аналізує програмний код Python, перевіряючи його на відповідність встановленим правилам і стандартам. Він оцінює якість коду, ідентифікує потенційні проблеми та недоліки, такі як синтаксичні помилки, неправильне використання змінних, некоректні імена функцій, несумісність типів, недостатні коментарі тощо.

2.Доступність правил аналізу: PyLint надає широкий набір правил аналізу, які можна налаштувати та вибрати в залежності від потреб проєкту. Це дозволяє визначити конкретні аспекти коду, на які слід звернути увагу під час аналізу.

3.Генерація звіту: PyLint генерує звіт, який містить виявлені проблеми, їх тип, місце в коді та рекомендації щодо виправлення. Звіт може бути представлений у різних форматах, включаючи текстовий формат, формат HTML або формат зрозумілий для інших інструментів аналізу коду.

4.Інтеграція з редакторами та іншими інструментами: PyLint може бути легко інтегрований з редакторами коду, такими як Visual Studio Code, PyCharm та інші. Це дозволяє розробникам отримувати наглядні підказки та попередження щодо потенційних проблем прямо у процесі редагування коду.

5.Налаштування та контроль якості коду: PyLint дозволяє налаштувати рівень строгості аналізу, включно з винятками певних правил або встановлення власних правил. Він також може бути використаний у процесі контролю якості коду, допомагаючи забезпечити стандартизований підхід до написання Python-коду.

Рисунок 1.4. - Логотип розширення для Python – PyLint

PyLint є потужним інструментом для аналізу коду Python, який допомагає забезпечити якість, читабельність та надійність програмного коду.

1.2 Генерація програмного коду нейромережами

1.2.1 Плагін Kite

Kite - це інтегрована розширення для редакторів коду, яка надає автоматичне доповнення коду, допоміжні підказки та функціональні можливості під час розробки програмного коду.

Рисунок 1.5. - Логотип плагіну Kite – розширення для редакторів коду[5]

Основні особливості та можливості Kite [5] включають:

1.Авто-доповнення коду: Kite аналізує контекст коду та надає інтелектуальні підказки для авто-доповнення коду. Він пропонує варіанти завершення функцій, методів, змінних та інших елементів коду, що полегшує та прискорює процес написання коду.

2.Допоміжні підказки: Kite відображає допоміжні підказки, які містять деталі про функції, методи, класи та інші елементи коду. Вони містять опис, приклади використання та іншу корисну інформацію, яка допомагає розробникам краще розуміти та використовувати різні частини API та бібліотек.

3.Аналіз коду: Kite використовує статичний аналіз коду, щоб зрозуміти його структуру та залежності. Це дозволяє забезпечити точне авто-доповнення та надавати відповідні підказки на основі контексту.

4.Підтримка багатьох мов: Kite підтримує багато мов програмування, включаючи Python, JavaScript, Go, Java, C++, C#, Ruby та інші. Це робить його корисним інструментом для розробки у різних середовищах програмування.

5.Інтеграція з редакторами коду: Kite інтегрується з популярними редакторами коду, такими як Visual Studio Code, PyCharm, Atom, Sublime Text та інші. Він надає плагіни або розширення для цих редакторів, що дозволяє зручно використовувати його функціональні можливості безпосередньо у редакторі.

Kite полегшує процес розробки програмного коду, надаючи автоматичне доповнення, допоміжні підказки та інші корисні функції, що сприяють покращенню продуктивності розробника.

1.2.2 Інструмент GitHub Copilot

Рисунок 1.6. - Логотип інструменту-помічника GitHub Copilot [6]

GitHub Copilot - це інтелектуальний помічник для програмістів, який розроблений GitHub у співпраці з OpenAI. Він використовує штучний інтелект, зокрема модель глибокого навчання GPT-3, для автоматичної генерації коду на основі контексту та інформації з відкритих джерел, включаючи публічні репозиторії GitHub.

Основні особливості та можливості GitHub Copilot [6] :

1.Авто-доповнення коду: Copilot надає швидке та інтуїтивно зрозуміле авто-доповнення коду, пропонуючи варіанти завершення рядків коду, функцій, класів та інших елементів на основі аналізу контексту.

2.Генерація коду: Copilot може автоматично створювати частини коду або навіть цілі функції на основі коментарів, описів або просто прикладів використання. Він намагається передбачити наміри розробника та забезпечити релевантний та коректний код.

3.Підтримка різних мов програмування: Copilot підтримує різні мови програмування, включаючи Python, JavaScript, TypeScript, Ruby, Go, Java, C++ та інші. Це робить його корисним інструментом для розробки в різних середовищах.

4.Інтеграція з редакторами коду: Copilot інтегрується з популярними редакторами коду, такими як Visual Studio Code, JetBrains IDEs, Atom та інші. Він працює як розширення або плагін, що дозволяє зручно використовувати його функціонал безпосередньо в редакторі.

5.Зворотній зв'язок та покращення: GitHub Copilot використовується в режимі "Technical Preview", що означає, що користувачі можуть надавати зворотній зв'язок щодо його роботи та якості генерації коду. Це допомагає вдосконалювати сервіс та робити його більш ефективним у майбутньому.

1.2.3 Модель глибокого навчання OpenAI Codex

Рисунок 1.7. - Логотип моделі глибокого навчання OpenAI Codex[7]

OpenAI Codex є нейронною мережею, розробленою компанією OpenAI, яка базується на моделі глибокого навчання GPT-3. Codex спеціалізується на генерації програмного коду та пов'язаних задач розробки програмного забезпечення.

Особливості та можливості OpenAI Codex [7] :

1.Генерація коду: Codex може створювати програмний код на основі вхідних запитів, коментарів, описів функцій або прикладів використання. Він розуміє різні мови програмування та вміє створювати код у відповідності до синтаксичних правил та популярних шаблонів.

2.Автодоповнення та підказки: Codex може пропонувати автодоповнення коду, шаблони, фрагменти та підказки під час розробки, що полегшує написання програмного коду та прискорює процес розробки.

3.Рефакторинг коду: Codex може допомагати з рефакторингом коду, запропоновуючи оптимізації, виправлення помилок, зміни структури та інші покращення для існуючого коду.

4.Взаємодія з редакторами коду: OpenAI Codex інтегрований з різними редакторами коду та інтегрованими середовищами розробки (IDEs). Це дозволяє розробникам використовувати його функціонал безпосередньо у своєму улюбленому редакторі.

5.Застосування в інших областях: Крім розробки програмного коду, OpenAI Codex може бути використаний у різних інших областях, таких як написання документації, створення відповідей на запитання, генерація тексту та інше.

OpenAI Codex є потужним інструментом, що спрощує процес розробки програмного коду та надає програмістам додаткові можливості та швидкість у їхніх завданнях.

1.2.4 Чат-бот ChatGPT

ChatGPT є моделлю штучного інтелекту розробки OpenAI, яка натренована на широкому спектрі текстових даних. Вона здатна відповідати на різноманітні запитання, надавати пояснення, виконувати завдання та надавати загальну інформацію [8].

ChatGPT працює на основі послідовностей слів і намагається зрозуміти контекст запиту, щоб забезпечити належну відповідь. Вона використовує свою внутрішню модель, яка складається з навчених патернів, правил та знань, щоб генерувати відповіді, які вона вважає належними.

Рисунок 1.8. - Логотип чат-боту ChatGPT[8]

Ця модель заснована на глибокому навчанні і навчалась на великому обсязі текстових даних, що дозволяє їй розуміти і узагальнювати інформацію з наведених даних, але важливо зазначити, що вона не має свідомості, розуміння або світосприйняття, як у людей.

1.2.5 Недоліки нейромереж

Хоча нейромережі можуть бути корисними для генерації коду, вони також мають свої обмеження та недоліки:

1.Проблеми в синтаксисі: Нейромережі можуть генерувати код, який не відповідає синтаксичним правилам конкретної мови програмування. Це може вимагати додаткового редагування та виправлення згенерованого коду, щоб зробити його виконавчим.

2.Обмежене розуміння семантики: Нейромережі можуть мати обмежене розуміння семантики коду [9]. Вони можуть генерувати код, який здавалося правильним з синтаксичної точки зору, але не відповідає очікуваному результату або не враховує специфічні особливості контексту використання.

3.Відсутність бізнес-логіки: Нейромережі не мають внутрішнього розуміння бізнес-логіки або специфічних потреб додатка. Вони можуть генерувати код, який технічно правильний, але не відповідає бізнес-логіці проєкту.

4.Низька якість коду: Згенерований код може бути низької якості з погляду читабельності, ефективності та інших аспектів. Нейромережі можуть не враховувати кращі практики програмування та оптимізації.

5.Залежність від навчальних даних: Нейромережі базуються на навчальних даних, які вони отримали під час процесу навчання. Якщо ці дані містять помилки або некоректності, це може вплинути на якість згенерованого коду.

Враховуючи ці недоліки, важливо мати на увазі, що нейромережі слід використовувати як інструмент для надання початкових рекомендацій або ідей, а не як єдине джерело коду. Розробники повинні залишатися критичними, перевіряти та валідувати згенерований код та забезпечувати його відповідність вимогам проєкту та кращим практикам програмування [10].

1.3 Порівняння альтернатив

Розглянуто 4 альтернативи аналізу програмного коду: SonarQube, Coverity, CodeClimate та PyLint. У таблиці 1.1. наведено порівняльну характеристику обраних альтернатив, в якій наведений короткий опис засобу, ефективність, переваги та недоліки.

Таблиця 1.1 – Порівняльна характеристика

Засіб	**Опис**	**Ефектив-ність**	**Переваги**	**Недоліки**
SonarQube	Автоматичний аналіз коду для виявлення помилок, дублікатів, недоліків безпеки та інших проблем.	Висока	- Підтримка багатьох мов програмування. - Великий набір правил та метрик для аналізу. - Інтеграція з іншими інструментами розробки.	- Вимагає конфігурації та налаштування. - Потребує додаткового часу для запуску аналізу на великих проєктах.
CodeClimate	Аналізує код на основі набору правил та метрик для виявлення якості коду та потенційних проблем.	Середня	- Простий інтерфейс та легка інтеграція. - Можливість налаштування правил аналізу. - Показники якості коду та технічний борг.	- Обмежена підтримка мов програмування порівняно з іншими інструментами. - Обмежена функціональність порівняно з більш продвинутими рішеннями.
Coverity	Виявляє помилки, недоліки та вразливості у коді шляхом аналізу потоків даних та статичного аналізу.	Висока	- Виявлення потенційних проблем у великих проєктах. - Мінімізація хибнопозитивних та хибнонегативних результатів. - Інтеграція з різними системами контролю версій.	- Вимагає налаштування та підготовки перед аналізом. - Висока вартість ліцензій та обмежена підтримка безкоштовних версій.

Кінець таблиці 1.1

Засіб	**Опис**	**Ефектив-ність**	**Переваги**	**Недоліки**
PyLint	Інструмент для аналізу коду Python, який виявляє стильові проблеми, помилки та інші недоліки.	Середня	- Проста установка та використання. - Виявлення широкого спектру проблем у коді. - Налаштування за допомогою конфігураційних файлів.	- Обмежена підтримка інших мов програмування, крім Python. - Низька гнучкість порівняно з деякими іншими інструментами.

Як видно із таблиці 1.1, кожний із засобів аналізу програмного коду має свої переваги та недоліки. Ефективність, вказана в таблиці є загальною та може різнитись в залежності від поставленого типу задачі.

SonarQube має високу ефективність, оскільки підтримує багато мов програмування. Також він має великий набір правил та метрик, за допомогою яких може аналізувати код, а інтеграція з іншими інструментами розробки є не складною. Але SonarQube є складним засобом, оскільки потрібно багато часу на конфігурацію параметрів та налаштування системи, а для запуску на великих проєктах витрачається додатковий час, оскільки йде велике навантаження.

CodeClimate має середню ефективність, оскільки має обмежену підтримку мов програмування та обмежену функціональність в порівняні з іншими інструментами. Але простий інтерфейс та легка інтеграція допомагають початківцям ознайомитись із засобом аналізу програмного коду, разом із можливістю налаштування правил аналізу і показником якості коду.

Coverity має більшу ефективність у великих проєктах, що допомагає виявити потенційні проблеми. Але засіб вимагає глибокої підготовки та налаштування перед аналізом. Підтримка безкоштовної версії обмежена.

PyLint, очевидно, має обмежену підтримку інших мов програмування, окрім Python, та має низьку гнучкість в порівнянні з іншими інструментами.

Але, як і інші засоби для Python, має просту установку та використання, виявляє проблеми коду в широкому спектрі і може налаштовуватись за допомогою конфігураційних файлів.

Також розглянуто 4 альтернативи генерації програмного коду за допомогою нейромереж: Kite, Copilot, OpenAI Codex, ChatGPT У таблиці 1.2. наведено порівняльну характеристику обраних альтернатив, в якій наведений короткий опис засобу, переваги та недоліки.

Засіб	Опис	Переваги	Недоліки
Kite	Інтегроване розширення редактора, яке надає автодоповнення та поради під час написання коду.	- Швидке та точне авто-доповнення коду. - Підтримка багатьох редакторів та IDE. - Покращує продуктивність розробника.	- Обмежена підтримка мов програмування. - Потребує підключення до хмарної платформи Kite. - Підтримка припинилась у 2021 році
GitHub Copilot	Розширення для IDE, що використовує нейромережу для генерації коду та надання рекомендацій.	- Швидке створення коду та автодоповнення. - Інтеграція з популярними редакторами. - Заснований на великому обсязі вихідних даних та навчанні.	- Можливість генерації некоректного або незахищеного коду. - Залежність від якості та розмаху навчальних даних.
OpenAI Codex	Модель глибокого навчання для генерації коду, яка заснована на навчанні на широкому спектрі даних.	- Велика кількість навчальних даних та широкий охоплюючий контекст. - Висока якість та різноманітність генерованого коду. - Здатність працювати з різними мовами програмування.	- Можливість генерації неточного або неочікуваного коду. - Залежність від доступності вхідних даних під час навчання.
ChatGPT	Модель глибокого навчання для генерації тексту та коду на основі заданого контексту.	- Широкий спектр генерації тексту та коду. - Здатність взаємодіяти з користувачами у формі чату. - Налаштування відповідей за допомогою контексту та інструкцій.	- Можливість генерації неточного або неочікуваного коду. - Залежність від доступності вхідних даних під час навчання.

Як видно з таблиці 1.2, кожний із засобів використовує нейромережі для генерації коду, і кожний із них має свої переваги, недоліки та сфери в яких їх можна використовувати.

Kite є швидким та точним інструментом авто-доповнення коду, що підтримує багато IDE, але потребує прямого підключення до хмарного середовища Kite і має обмежену підтримку різних мов програмування.

GitHub Copilot може інтегруватися з популярними редакторами IDE, навчаний на великому обсязі даних, але, як і всі інші нейромережі, може генерувати неправильний або незахищений код.

OpenAI Codex здатний працювати з різними мовами програмування, та має велику кількість навчальних даних. Має можливість працювати в контексті, тому має велику різноманітність та якість коду.

ChatGPT наразі має більшу ефективність у генеруванні програмного коду і не тільки. Він здатний взаємодіяти з користувачем у формі чату, тому може підлаштовувати свої відповіді відповідно до контексту.

ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі розглянуто різні способи генерації коду за допомогою нейромереж, що спрощує життя розробникам, оскільки процес програмування є часо- та ресурсоємним.

Засоби генерації коду з використанням нейромереж мають великий потенціал для автоматичної генерації коду, полегшення розробки та підвищення продуктивності розробників. Вони можуть запропонувати швидкі та точні шаблони коду, заповнювати пропущені деталі та навіть пропонувати альтернативні реалізації.

Розглянуто такі сервіси, як Kite, GitHub Copilot, OpenAI Codex та ChatGPT. Kite відрізняється високою швидкістю та можливістю автозаповнення коду, GitHub Copilot пропонує широкий спектр функціональності та відомостей з відкритих джерел, OpenAI Codex має широкий спектр мов програмування та великий обсяг навчальних даних, а ChatGPT може бути використаний для генерації коду та відповідей на загальні питання.

Проте, важливо пам'ятати, що нейромережеві засоби генерації коду мають свої обмеження та ризики. Вони можуть генерувати неточний або неочікуваний код, що вимагає ретельної перевірки та тестування. Залежно від якості вхідних даних та навчання моделей, їхні результати можуть бути різними.

Для запобігання помилок можна використовувати аналізатори програмного коду. У цьому розділі порівняно кілька популярних інструментів аналізу коду, таких як SonarQube, Coverity, CodeClimate та PyLint. Кожен з цих інструментів має свої переваги та недоліки.

SonarQube відрізняється широким спектром правил та метрик для аналізу коду, а Coverity пропонує виявлення потенційних проблем у великих проєктах.

CodeClimate є простим у використанні та налаштуванні, а PyLint допомагає виявляти широкий спектр проблем у коді.

Найкращий вибір інструменту аналізу коду залежить від конкретних вимог та контексту проєкту. Важливо оцінити потреби, масштаб та складність проєкту, підтримку мов програмування, інтеграційні можливості та інші фактори.

Загалом, аналіз коду та використання аналізаторів програмного коду є важливою складовою процесу розробки програмного забезпечення, що допомагає підвищувати якість, надійність та безпеку програмного продукту. Використання правильних інструментів аналізу коду може значно спростити та покращити цей процес, але вимагає розуміння їхніх можливостей та обмежень для досягнення найкращих результатів.

РОЗДІЛ 2
АНАЛІЗ НАЙПОШИРЕНІШИХ СПОСОБІВ ОБРОБКИ ТА ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ

2.1 Інструменти в мові програмування Python для аналізу програмного коду

Мова програмування Python має безліч інструментів для аналізу програмного коду. Найпоширенішим з них є PyLint – розширення статичного аналізу коду для мови Python, який допомагає виявляти потенційні помилки, проблеми стилістики та порушення конвенцій програмування. PyLint аналізує програмний код Python, використовуючи ряд правил і евристик для виявлення потенційних проблем, стилістичних недоліків та вразливостей. Основний процес аналізу включає наступні кроки:

1.Створення дерева синтаксичного аналізу: PyLint використовує вбудований синтаксичний аналізатор для створення дерева синтаксичного аналізу коду Python. Це дерево представляє структуру програми та залежності між її елементами, такими як модулі, класи, функції, змінні тощо.

2.Перевірка правил аналізу: PyLint застосовує набір правил аналізу до дерева синтаксичного аналізу. Ці правила перевіряють код на різні аспекти, включаючи синтаксичні помилки, стилістичні недоліки, невикористані змінні, неправильне використання функцій та модулів, несумісність типів та інші проблеми, що можуть вплинути на якість коду.

3.Генерація результатів аналізу: PyLint генерує звіт, який містить виявлені проблеми, їх тип, місце в коді та рекомендації щодо виправлення. Звіт може містити інформацію про рівень серйозності проблеми та її вплив на якість коду.

4.Налаштування правил аналізу: PyLint надає можливість налаштувати правила аналізу відповідно до потреб проєкту. Користувач може включати або виключати певні правила, налаштовувати рівень строгості аналізу та встановлювати свої власні правила.

5.Інтеграція з іншими інструментами: PyLint може бути легко інтегрований з редакторами коду, такими як PyCharm, Visual Studio Code, або використовуватися в зв'язці з іншими інструментами автоматичного тестування та збірки.

PyLint допомагає розробникам виявляти та виправляти потенційні проблеми в програмному коді Python, сприяючи поліпшенню якості, стилістики та надійності кодової бази. Далі наведено основні критерії, за якими PyLint аналізує програмний код.

2.1.1 Можливості PyLint

PyLint є інструментом статичного аналізу коду для мови програмування Python. Він призначений для виявлення потенційних проблем, помилок та неузгодженостей в коді з метою поліпшення якості та стилю програмного забезпечення. В функції PyLint входить [11]:

1. Перевірка стилю коду.
2. Виявлення помилок.
3. Аналіз комплексності коду.
4. Перевірка використання модулів та пакетів.
5. Надання рекомендацій та статистики.

2.1.2 Переваги та можливі недоліки використання

PyLint відрізняється від інших схожих засобів аналізу коду наступними особливостями:

1. PyLint активно використовує PEP 8, який є стандартом оформлення та стилю коду для мови Python [13]. Він перевіряє відповідність коду цим стандартам, надаючи рекомендації щодо поліпшення стилю та структури коду. Це допомагає забезпечити однорідність та читабельність коду у всьому проєкті.

2. PyLint надає широкі можливості налаштування, які дозволяють користувачам вибрати конкретні правила аналізу, які вони хочуть застосовувати до свого коду. Це дозволяє налаштувати аналіз таким чином, щоб відповідати потребам конкретного проєкту або команди розробників.

3. PyLint підтримує не тільки статичний, але й динамічний аналіз коду. Він виявляє потенційні помилки та проблеми, що можуть виникнути під час виконання програми. Це дозволяє виявити недоліки, які можуть бути недоступні для статичного аналізу.

4. PyLint має широкий набір правил та перевірок, які охоплюють багато аспектів програмного коду. Він перевіряє стиль коду, виявляє синтаксичні помилки, перевіряє правильність використання змінних та функцій, оцінює складність коду та багато іншого. Це дозволяє виявляти широкий спектр проблем та покращувати якість коду.

5. PyLint є розширюваним інструментом, який можна розширити за допомогою плагінів та розширень. Це дає можливість додавати власні правила та перевірки, а також налаштовувати поведінку інструменту під потреби конкретного проєкту.

Хоча існують інші засоби для генерації коду, всі наведені вище особливості PyLint роблять його найкращим вибором інструменту для аналізу програмного коду для даної роботи.

Хоч PyLint і є потужним інструментом для аналізу коду Python, він також має деякі недоліки, серед яких можуть бути наступні [4]:

1. Суворість: PyLint може бути досить суворим щодо вимог до стилю коду та дотримання правил. Це може вимагати значних зусиль для внесення змін в існуючий код або вирішення помилок, особливо якщо він був написаний без

урахування таких правил. Деякі правила також можуть виглядати суб'єктивними або неадекватними для конкретних проєктів.

2. Велика кількість помилок та попереджень: Зазвичай PyLint генерує значну кількість помилок та попереджень, особливо при аналізі великих проєктів. Це може призвести до перенасиченості звіту, ускладнити виявлення дійсно серйозних проблем та затримати процес розробки. Важливо аналізувати повідомлення, концентруючись на найбільш критичних аспектах коду.

3. Швидкодія: PyLint може бути відомий своєю відносно повільною роботою, особливо при аналізі великих проєктів або кодових баз. Це може призвести до затримок у процесі розробки та виконання аналізу. Для великих проєктів можливо необхідно розглянути альтернативи з більшою швидкодією.

4. Обмеження на деякі типи помилок: PyLint покриває багато аспектів коду, але може бути обмежений у виявленні деяких типів помилок або некоректного використання деяких бібліотек або фреймворків. Деякі специфічні помилки можуть потребувати ручної перевірки.

5. Складність конфігурації: Початкове налаштування PyLint та визначення правил аналізу може бути важким завданням. Вимоги проєкту можуть вимагати індивідуального налаштування, що вимагає часу та досвіду.

Не зважаючи на ці недоліки, PyLint залишається потужним інструментом для аналізу коду Python, який може допомогти поліпшити якість та стиль коду в проєкті.

2.2 Генерація програмного коду, використовуючи ChatGPT

2.2.1 Концепція prompt engineering

На час виконання дипломної роботи, ChatGPT ще немає можливості інтегруватись із додатками користувачів [12]*, тому замість цієї моделі буде*

використано модель OpenAI Codex, що є дуже схожою за функціоналом для генерації коду, оскільки базований на тій самій моделі.

OpenAI Codex, базований на GPT-3.5 моделі(як і чат-бот ChatGPT), використовує техніку «prompt engineering» (дослівно з англ. «Запитова інженерія») для генерації коду.

Далі наведено огляд того, як це працює [14] :

1.Аналіз вхідного тексту: При наданні тексту OpenAI Codex обробляє його та аналізує контекст і намір запиту. Проходить аналіз ключових слів та структури запиту, щоб визначити програмоване завдання і/або проблему, яку описано.

2.Розуміння мови: Codex навчений на широкому спектрі даних, пов'язаних з програмуванням(документації, репозиторії коду, приклади, тощо). Ці знання використовуються для розуміння концепцій програмування, синтаксису та бібліотек, що відносяться до запиту.

3.Контекстне розуміння: запит аналізується, враховуючи попередній контекст. Увага звертається на деталі та залежності в поередньому тексті, щоб генерувати точний та контекстно-орієнтований код.

4.Розпізнавання шаблонів: Codex розпізнає шаблони та загальні програмні прийоми під час навчання. Він впізнає схожі шаблони в тексті та генерує відповідний код.

5.Генерація коду: генерується код, що має виконувати запит. Враховується мова програмування, яку вказано в запиті. За її відсутності – визначення проходить автоматично.

6.Безпека та коректність: Хоча OpenAI Codex був навчений на великому обсязі даних – важливо зазначити що згенерований код може бути некоректним, або навіть небезпечним. Тому згенерований код важливо аналізувати для забезпечення його коректності та безпеки.

2.2.2 Генерація коду

Генерація коду за допомогою OpenAI Codex відбувається наступним чином:

1.Користувач надає вхідні дані у вигляді текстового запиту або коментарів, які описують намір або завдання, для якого потрібно згенерувати код. Наприклад, це може бути опис функції, коментар з описом функціональності або приклад вхідних та вихідних даних.

2.OpenAI Codex аналізує контекст вхідних даних, враховуючи текстові деталі, які вказують на мову програмування, змінні, класи, функції та інші елементи, що використовуються у коді.

3.З використанням моделі глибокого навчання, OpenAI Codex генерує код, відповідний вхідним даним та контексту. Він використовує свої знання про мови програмування, популярні бібліотеки, шаблони та практики розробки для створення відповідного та синтаксично правильного коду.

4.OpenAI Codex може надати кілька варіантів згенерованого коду, які відповідають вхідним даним та контексту. Він також може пропонувати авто-доповнення, підказки та шаблони, які полегшують процес розробки та допомагають уникнути синтаксичних помилок.

5.Користувач може взаємодіяти з OpenAI Codex, коригувати згенерований код, вибирати певні варіанти або надавати додаткові вказівки, щоб поліпшити якість генерації коду. Це допомагає Codex вчитися та вдосконалюватися з кожною ітерацією.

2.2.2 Codex Playground

Codex Playground – це інтерактивний веб-сервіс, який надається OpenAI для використання і випробовування OpenAI Codex. Він дозволяє експериментувати з Codex, вводити текстові запити та отримувати

згенерований код в режимі реального часу. Далі наведено кілька ключових особливостей Codex Playground [15] :

1.Зручний інтерфейс: Codex Playground має інтуїтивно зрозумілий та простий інтерфейс, що дозволяє користувачам вводити текстові запити та переглядати результат безпосередньо на сторінці.

2.Мови програмування: Codex підтримує багато різних мов програмування, такі як JavaScript, Python, Ruby, Go та багато інших.

3.Режим real time: під час введення запиту видно, як модель негайно генерує код, що дозволяє швидко експериментувати з ідеями та бачити результат на місці.

4.Документація: Codex Playground надає підсказки та документацію, що допомагають користувачам вивчити можливості Codex. Можна знайти приклади коду, посилання на розділи документації та вказівки щодо ефективного використання.

5.Збереження коду: Після генерації коду, результатом запиту можна поділитись, скопіювати згенерований код або зберегти його для подальшого використання.

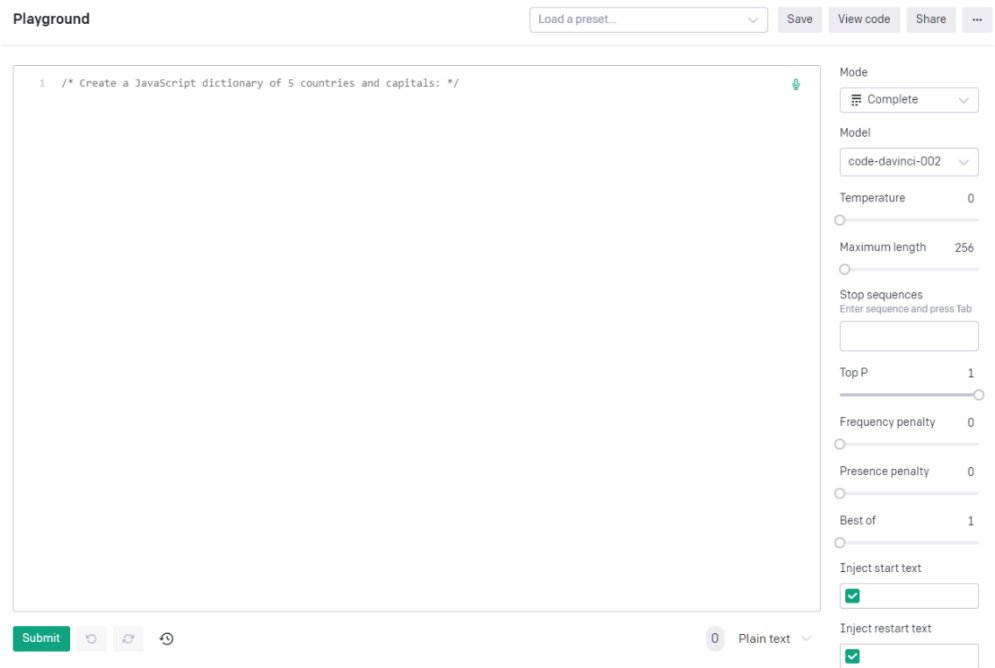


Рисунок 2.1 – Інтерфейс Codex Playground

Codex Playground є потужним інструментом для дослідженнята випробовування можливостей OpenAI Codex безпосередньо в веб-браузері. Але для більш ефективного використання потрібно використовувати OpenAI API, оскільки інтерфейс Codex Playground не має інтеграції з іншими продуктами.

2.2.3 OpenAI API

OpenAI API дозволяє розробникам інтегрувати OpenAI моделі в власні застосунки, інструменти чи сервіси. Далі наведено аналіз того, як працює API OpenAI [7] :

1.Аутентифікація: Для використання OpenAI API, потрібно отримати API ключ, який отримується на платформі OpenAI. Цей ключ використовується для аутентифікації і повинен бути в хедері кожного API запиту.

2.Точки доступу API: API OpenAI надає різні точки доступу для конкретних завдань. Для текстових моделей, таких як GPT-3.5, зазвичай використовується точка доступу "completions" для генерації тексту на основі запиту.

3.Формування запитів до API: Щоб зробити запит до API, потрібно сформувати HTTP POST-запит до відповідної точки доступу. В тілі запиту включаються необхідні параметри і дані для конкретного виклику API.

4.Налаштування моделі: Під час виконання запиту вказується модель, яку необхідно використовувати, наприклад, "text-davinci-003" для GPT-3.5. Також можна налаштувати різні параметри, такі як температура для контролю випадковості виводу або максимальну кількість токенів для обмеження довжини відповіді.

5.Обробка токенів: Моделі OpenAI обробляють текст у невеликі частини, які називаються "токенами", де токен може бути один символ або, наприклад, ціле слово. Відповідь API включає згенерований текст у вигляді послідовності токенів. Потрібно декодувати токени, щоб отримати читабельний текст.

6.Аналіз відповідей API: Відповідь API містить згенерований текст або результат запиту. Проводити аналіз та обробляти цю відповідь можна у програмі або відображати її користувачам.

2.2.4 Переваги та можливі недоліки використання OpenAI Codex

OpenAI Codex є потужним інструментом для допомоги із програмним кодом, оскільки створений саме для цієї задачі. Переваги Codex над іншими подібними засобами генерації коду [7]:

1. Масштаб та навчальні дані: OpenAI Codex ґрунтується на архітектурі GPT-3.5, яка є високотехнологічною та потужною мовною моделлю. Він навчений на величезному наборі даних, що включає широкий спектр мов програмування, документацію, репозиторії коду та інші відповідні джерела. Ці обширні навчальні дані дозволяють Codex мати ширше розуміння концепцій програмування та більш комплексну базу знань.

2. Підтримка багатьох мов: Codex підтримує декілька мов програмування, включаючи популярні, такі як Python, JavaScript, Java, C++ та багато інших. Ця можливість підтримки декількох мов буде корисною при подальшому розвитку системи для генерації і аналізу програмного коду.

3. Розуміння природної мови: Codex проявляє сильну здатність розуміти текст природною мовою. Він може інтерпретувати та обробляти інструкції, написані людиною, що дозволяє користувачам виражати свої наміри та вимоги більш інтуїтивно. Це розуміння природної мови значно покращує користувацький досвід і зменшує когнітивне навантаження при взаємодії з моделлю.

4. Контекстуальне розуміння: Codex проявляє вражаюче контекстуальне розуміння заданого фрагмента коду або програмувальних проблем. Він може розуміти контекст коду, типи змінних, сигнатури функцій та залежності, щоб надавати більш точні та контекстно відповідні пропозиції щодо коду. Це

контекстуальне усвідомлення дозволяє Codex генерувати код, який відповідає наміру користувача та дає бажані результати.

5. Якість та зв'язність коду: Codex відомий своєю здатністю генерувати високоякісний код, який відповідає найкращим практикам та зберігає правильну синтаксичну структуру. Він може створювати добре структурований, читабельний та зручний для підтримки код, що зменшує необхідність в ручній переробці. Крім того, Codex здатний генерувати зв'язний код, що робить його більш послідовним та зрозумілим.

Хоча існують інші моделі для генерації коду, широкі навчальні дані, підтримка багатьох мов, розуміння природної мови, контекстуальне усвідомлення, якість коду OpenAI Codex роблять його провідним вибором для даної роботи.

Звісно, як і будь яка інша модель, OpenAI Codex має свої недоліки. Хоча OpenAI Codex є потужною мовною моделлю, яка може допомагати у різних завданнях, використання її також може супроводжуватися деякими проблемами та викликами. Серед них можуть бути такі:

1. Розуміння контексту: Codex іноді може генерувати відповіді, які не мають контексту або зв'язку. Якщо вхідні дані є неоднозначними або погано сформульованими, модель може надавати технічно правильні, але незв'язні або несуттєві відповіді.

2. Надмірне покладання на Codex: Можна спокуситися надто сильно покладатися на Codex для складних або критичних завдань без подвійної перевірки вихідних даних. Хоча Codex може надавати корисні рекомендації, він не повинен замінювати людський розсуд або експертизу.

3. Обмеження підтримки: В квітні 2023 року OpenAI повідомили що підтрика Codex буде мінімальною та з часом перестане підтримуватися взагалі. Хоча на сьогодення OpenAI Codex все ще є актуальною і потужною моделю – в майбутньому потрібно буде виконувати пошук іншої моделі для генерації коду.

4. Проблеми з приватністю та безпекою: Використання Codex може включати передачу чутливої або конфіденційної інформації. Але цього можна укникнути, якщо слідувати рекомендаціям, які OpenAI надає по Codex.

5. Питання інтелектуальної власності: Генерація коду або контенту за допомогою Codex може викликати питання щодо інтелектуальної власності, особливо якщо результат нагадує існуючий авторський матеріал або порушує патенти.

6. Обмеження продуктивності: Незважаючи на потужність Codex, він може мати деякі обмеження при виконанні певних завдань, особливо тих, що вимагають специфічних знань у певній області або складного мислення. Він може не завжди генерувати бажаний результат або не відповідати очікуванням у певних сценаріях.

OpenAI активно працює над вирішенням цих проблем та поліпшенням можливостей системи з часом. Тому для того щоб користуватись їх продуктом ефективно, необхідно стежити за змінами та оновленнями.

ВИСНОВКИ ДО РОЗДІЛУ 2

У цьому розділі розглянуто OpenAI Codex і PyLint як два інструменти для генерації та аналізу програмного коду. OpenAI Codex є потужною мовною моделлю, яка може допомагати у генерації коду та виконанні різних завдань програмування. Він має багато переваг, такі як швидка реакція, велика база знань, можливість генерування коду зі стислих описів та підтримка різних мов програмування. Крім того, він може бути особливо корисним у випадках, коли розробник має обмежений досвід у програмуванні або не має достатньо часу для написання коду вручну.

PyLint є інструментом для аналізу коду Python. Він допомагає виявляти потенційні проблеми в коді, такі як синтаксичні помилки, стилістичні недоліки, невикористаний код та інші. Він забезпечує можливість покращити якість та стиль коду, а також знизити кількість помилок, що можуть виникнути при виконанні програми.

Обидва інструменти мають свої переваги і обмеження. OpenAI Codex надає широкий спектр можливостей, але може виникати проблема розуміння контексту, виникнення біасованих або неприйнятних відповідей, відсутності перевірки джерел та інших питань, пов'язаних з використанням мовної моделі. PyLint, з іншого боку, може допомогти виявити проблеми в коді, але може вимагати складної конфігурації та мати деякі недоліки, такі як ложнопозитивні помилки або обмежену підтримку нових функцій мови.

Важливо враховувати ці переваги і недоліки при використанні цих інструментів і знаходити баланс між автоматизованою допомогою та ручною перевіркою. Комбінація Codex і PyLint, разом з розсудливістю та експертизою розробників, може допомогти забезпечити якість, ефективність та безпеку коду в проєктах програмування. Наприклад, розробник може скористатися Codex для створення заготовки коду, а потім використовувати PyLint для виявлення та

виправлення помилок. Крім того, розробник може використовувати Codex для вирішення складних завдань, таких як генерація коду для машинного навчання, а PyLint допоможе забезпечити якість та стиль цього коду. Таким чином, використання обох інструментів може допомогти розробникам забезпечити ефективне та безпечне програмування, що відповідає вимогам сучасного ринку.

РОЗДІЛ 3
РОЗРОБКА ТА РЕАЛІЗАЦІЯ АЛГОРИТМУ ГЕНЕРАЦІЇ ТА АНАЛІЗУ ПРОГРАМНОГО КОДУ

3.1 Загальний опис алгоритму

Основним завданням є розробка способу генерації та аналізу програмного коду, використовуючи нейромережі та аналізатори. Це означає, що запропоноване рішення має за запитом генерувати та одразу аналізувати згенерований код для оцінки його якості.

Загальна структура реалізації:

1) під’єднання до моделі глибокого навчання OpenAI Codex через API;
2) створення запитів для генерації коду;
3) підключення розширення PyLint;
4) аналіз згенерованого коду та його оцінка;
5) формування результатів аналізу програмного коду.

3.2 Деталізований опис розробки алгоритму та його архітектури

3.2.1 Під’єднання до моделі глибокого навчання OpenAI Codex через API

Для того щоб під’єднатись до моделі OpenAI Codex потрібно отримати власний API ключ на платформі OpenAI.

```python
import openai

# Встановлення ключа API
openai.api_key = 'API_KEY'

# Продовження коду...
```

У цій частині коду використано бібліотеку openai, яка надає зручні функції для взаємодії з API OpenAI. Власний ключ потрібно підставити замість 'API_KEY'.

3.2.2 Створення запитів для генерації коду

Після успішної аутентифікації, можна продовжити використовувати API OpenAI для здійснення запитів та отримання результатів. Далі наведено базовий алгоритм, як використати API OpenAI для здійснення запиту до моделі GPT-3.5, що є основою і Codex, і ChatGPT, для генерації тексту [7]:

```python
import openai

# Встановлення ключа API
openai.api_key = 'YOUR_API_KEY'

# Параметри запиту
prompt = "Generate code to calculate the factorial of a number in Python."
max_tokens = 100
temperature = 0.8

# Запит до API
response = openai.Completion.create(
    engine="davinci-codex",  # Використовування Codex для генерації коду
    prompt=prompt,
    max_tokens=max_tokens,
    temperature=temperature
)

# Отриманий згенерований код занесено в змінну
generated_code = response.choices[0].text.strip()

# Виведення згенерованого коду
print(generated_code)
```

У цьому прикладі використано метод openai.Completion.create() для створення запиту до API. Передаються параметри запиту, такі як engine (двигун, в даному випадку "davinci-codex"), prompt (текстовий запит, який описує завдання), max_tokens (максимальна кількість токенів відповіді) і temperature - значення від 0.1 до 1.0, де більше температури (наприклад, 0.8) робить згенерований вивід більш різноманітним та випадковим, тоді як менше значення (наприклад, 0.1) робить його більш детермінованим та збалансованим.

Після здійснення запиту, відповідь містить згенерований текст або код у властивості response.choices[0].text. У прикладі, згенерований код витягується та виводиться на екран.

У цьому прикладі показано базовий алгоритм. Для задоволення умов проєкту, необхідно зробити декілька модифікацій:

- Параметри запиту prompt, max_tokens, temperature зробити доступними для зміни користувачу.
- Згенерований код виводити не тільки на екран, а й зберігати його у текстовий файл (чи файл з розришенням .py) для подальшого використання.

Зберегти код у файл можна за допомогою методу with open:

```
# Збереження згенерованого коду у файл
with open("generated_code.py", "w") as file:
    file.write(generated_code)
```

3.2.3 Підключення розширення PyLint

Підключення розширення PyLint відбувається шляхом підключення бібліотеки pylint:

```
pip install pylint
```

Далі існує кілька способів початку аналізу коду:

- За допомогою консолі. Необхідно в консолі написати “pylint <назва файлу.py>
- За допомогою виклику функції:

```
import pylint

# Запуск аналізу через pylint
pylint.run_pylint(["generated_code.py"])
```

Функція run_pylint може приймати багато параметрів. Найчастіше використованими є:

1) --disable: Дозволяє вимкнути певні перевірки PyLint. Можна передати список ідентифікаторів модулів, класів або функцій, які потрібно вимкнути.

2) --enable: Дозволяє ввімкнути певні перевірки PyLint. Можна передати список ідентифікаторів модулів, класів або функцій, які потрібно ввімкнути.

3) --ignore: Дозволяє вказати певні файли, папки або шаблони файлів, які необхідно ігнорувати під час аналізу.

4) --rcfile: Дозволяє вказати файл конфігурації PyLint, який містить додаткові налаштування для аналізу.

5) --output-format: Дозволяє вибрати формат виводу результатів аналізу.

6) --reports: Дозволяє зберігати результати аналізу у файл.

7) --load-plugins: Дозволяє завантажити додаткові плагіни для PyLint.

3.2.4 Приклади аналізу коду

Далі наведено критерії та приклад аналізу коду розширенням PyLint [11]:

1) PyLint перевіряє, чи відповідає код вказівкам та конвенціям, визначеним у стилістичному посібнику Python Enhancement Proposals (PEP 8).

Він перевіряє такі аспекти, як відступи, довжину рядка, правила найменування, імпорт модулів, використання пробілів тощо.

Приклад:

```
import time, random

print("Text with a really long line so pylint can detect it as a problem so you can deal with it as you want Text with a really long line so pylint can detect it as a problem so you can deal with it as you want")

text = "Plain text"

def FUNC(parameters):
    return parameters * random.randint(1, 5)

start = time.time()
print(FUNC(4))
```

При аналізі цього коду, PyLint видає таке повідомлення:

```
********** Module simple
simple.py:3:0: C0301: Line too long (212/100) (line-too-long)
simple.py:11:0: C0304: Final newline missing (missing-final-newline)
simple.py:1:0: C0410: Multiple imports on one line (time, random, io) (multiple-imports)
simple.py:5:0: C0103: Constant name "text" doesn't conform to UPPER_CASE naming style (invalid-name)
simple.py:7:0: C0103: Function name "FUNC" doesn't conform to snake_case naming style (invalid-name)

-----------------------------------------------------------------
Your code has been rated at 0.00/10
```

Розберемо кожну знайдену проблему окремо:

- Line too long (212/100) (line-too-long) – стрічка коду є занадто довгою(212 символів із 100 рекомендовануиз);
- Final newline missing (missing-final-newline) – відсутній новий рядок в кінці файлу;
- Multiple imports on one line (time, random, io) (multiple-imports) – бібліотеки time, random імортовано в одному рядку;
- Constant name "text" doesn't conform to UPPER_CASE naming style (invalid-name) – константна змінна text не відповідає стилю найменування UPPER_CASE, тобто великими літерами;
- Function name "FUNC" doesn't conform to snake_case naming style (invalid-name) – назва функції FUNC не відповідає стилістиці snake_case, де перша літера кожного слова пишеться з маленької, а кожний пробіл замінюється на нижнє підкреслення.

Після виправлення коду:

```
import time
import random

print("Text with a really long line so pylint can detect it as a problem so you can deal with it"
      "Text with a really long line so pylint can detect it as a problem so you can deal with it")

TEXT = "Plain text"


def func(parameters):
```

```
    return parameters * random.randint(1, 5)

start = time.time()

print(func(4))
```

Отримано такий результат аналізу:

```
************* Module simple

-------------------------------------------------------------------
Your code has been rated at 10/10 (previous run: 0/10, +10.00)
```

2) PyLint виявляє синтаксичні помилки, такі як неправильні або неправильно розміщені оператори, відсутні дужки або лапки, некоректне використання операторів тощо. Він також визначає семантичні помилки, включаючи невизначені змінні, не використані імпорти та непідтримуванні присвоєння типів.

Приклад:

```
import time

X, A = 5, [1, 2, 3]
B
if (5 + 5 == 10):
    Y = X + 3
```

Проаналізувавши такий код, отримано результат:

```
************* Module simple
```

```
simple.py:4:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)
simple.py:3:0: W0104: Statement seems to have no effect (pointless-statement)
simple.py:3:0: E0602: Undefined variable 'B' (undefined-variable)
simple.py:6:0: W0104: Statement seems to have no effect (pointless-statement)
simple.py:1:0: W0611: Unused import time (unused-import)

-------------------------------------------------------------------
Your code has been rated at 0.00/10
```

Розберемо кожну знайдену проблему окремо:

- Unnecessary parens after 'if' keyword (superfluous-parens) – зайві(необов'язкові) дужки після if;
- Statement seems to have no effect (pointless-statement) – строка не має ефекту, строка не має сенсу
- Undefined variable 'B' (undefined-variable) – Невизначена змінна B;
- Unused import time (unused-import) – невикористаний імпорт.

Після виправлення коду:

```
X, A = 5, [1, 2, 3]

if 5 + 5 == 10:
    Y = X + 3
```

Отримано такий результат аналізу:

```
************* Module simple

-------------------------------------------------------------------
Your code has been rated at 10/10 (previous run: 0/10, +10.00)
```

3) PyLint оцінює складність коду, аналізуючи такі фактори, як цикломатична складність, яка вимірює кількість лінійно незалежних шляхів у функції або методі. Він виявляє надмірно складний код, який може бути складнішим для читання, розуміння та підтримки. Крім того, PyLint може виявляти проблеми з дизайном, такі як невикористаний код, дублювання коду або неправильне використання структур управління.

4) PyLint перевіряє, як обробляються виключення у коді. Він виявляє випадки, коли виключення перехоплюються, але не обробляються належним чином, або ситуації, коли повідомлення про виняток не використовуються ефективно для налагодження або повідомлення про помилку.

5) PyLint оцінює наявність та якість документації в кодовій базі. Він перевіряє відсутність або неповність докстрінгів (інлайнової документації) та дотримання вказівок щодо документування модулів, функцій, класів та їх параметрів [16].

Приклад:

```
import random

def random_sum(a_num):
    b_num = random.randint(0, 10)
    return a_num + b_num


print(random_sum(2))
```

Проаналізувавши такий код, отримано результат:

```
************* Module simple
simple.py:1:0: C0114: Missing module docstring (missing-module-
docstring)
```

```
simple.py:4:0: C0116: Missing function or method docstring (missing-
function-docstring)

-------------------------------------------------------------------
Your code has been rated at 6.00/10
```

Розберемо кожну знайдену проблему окремо:

- Missing module docstring (missing-module-docstring) – відсутній докстрінг модуля.
- Missing function or method docstring (missing-function-docstring) – відсутній докстрінг функції

Після виправлення коду:

```
"""Module for finding sum of 2 numbers"""
import random


def random_sum(a_num):
    '''Returned argument is a sum of 2 numbers.'''
    b_num = random.randint(0, 10)
    return a_num + b_num


print(random_sum(2))
```

Отримано такий результат аналізу:

```
************* Module simple

-------------------------------------------------------------------
Your code has been rated at 10/10 (previous run: 6/10, +4.00)
```

3.2.5 Формування результатів аналізу програмного коду

Як згадано вище, результат аналізу можна зберігати у файл.

Так, параметр --reports в PyLint дозволяє зберігати результати аналізу у файл. За допомогою цього параметра можна вказати різні формати файлів для збереження результатів. Ось кілька доступних форматів файлів для параметра –reports [11]:

1. «text»: Це формат за замовчуванням, в якому результати аналізу зберігаються у текстовому файлі.

2. «colorized»: Результати аналізу зберігаються у текстовому файлі з кольоровим виділенням для полегшення сприйняття.

3. «parseable»: Результати аналізу зберігаються у файлі у форматі, який можна легко обробляти іншими інструментами.

4. «html»: Результати аналізу зберігаються у файлі HTML, який можна відкрити у веб-браузері для зручного перегляду.

5. «json»: Результати аналізу зберігаються у файлі JSON, який містить структуровану інформацію про помилки, попередження та інші аспекти аналізу.

6. «junit»: Результати аналізу зберігаються у файлі у форматі JUnit XML, який є стандартним форматом для звітів про тестування в багатьох інструментах.

7. «html-inline»: Результати аналізу зберігаються у файлі HTML, але всі стилі та сценарії вбудовуються безпосередньо в HTML-файл. Це може бути корисним, якщо потрібно мати один самодостатній файл для перегляду результатів.

8. «mako»: Результати аналізу зберігаються у файлі у форматі Mako Template, який є шаблонним двигуном для генерації динамічних HTML-сторінок.

9. «msvs»: Результати аналізу зберігаються у файлі у форматі, який може бути прочитаний іншими інструментами розробки, наприклад, в середовищі Microsoft Visual Studio.

10. «null»: Результати аналізу не зберігаються у файл, а просто відображаються на екрані. Це корисно, якщо немає необхідності зберігати результати аналізу у файл.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі описано процес розробки та реалізації алгоритму, що поєднує генерацію програмного коду за допомогою OpenAI Codex та аналіз коду за допомогою PyLint. У цьому розділі надано загальний опис алгоритму та деталізований опис його розробки та архітектури.

У розробці алгоритму використовувалось під'єднання до моделі глибокого навчання OpenAI Codex через API, створення запитів для генерації коду, підключення розширення PyLint для аналізу коду та формування результатів аналізу програмного коду.

Запропонований спосіб генерації та аналізу програмного коду є ефективним в контексті простоти використання та забезпечення швидкого засвоєння новими розробниками. Генерація програмного коду за допомогою OpenAI Codex дозволяє швидко отримувати початковий код для подальшої роботи, що зменшує зусилля, потрібні для початку проєкту.

Аналіз програмного коду за допомогою PyLint надає можливість виявляти потенційні помилки, дотримуватись стандартів коду та отримувати рекомендації щодо покращення. Це допомагає забезпечити якість та стабільність програмного забезпечення.

Продукт, що поєднує генерацію коду за допомогою OpenAI Codex та аналіз за допомогою PyLint, може забезпечити зручне та ефективне середовище розробки для команди розробників. Він може допомогти зекономити час та зусилля, спростити процес розробки програмного забезпечення і забезпечити високу якість коду.

РОЗДІЛ 4
РЕЗУЛЬТАТ РОЗРОБКИ АЛГОРИТМУ ГЕНЕРАЦІЇ ТА АНАЛІЗУ ПРОГРАМНОГО КОДУ

4.1 Есперименталь ні дані та результати

Проведено тестування програмного продукту. Процес тестування складався з таких етапів:

- створення запиту для генерації програмного коду;
- аналіз згенерованого коду;
- перевірка результатів людським фактором.

Для тестування створено кілька запитів для генерації програмного коду. Для прикладу на рис 4.1. зображено один із них:

```
/* Create a Python function that calculates square root of a number. Include checking whether the input data is a number */
```

Рисунок 4.1. – Приклад запиту для генерації коду

Для уникнення можливих непорозумінь, використано англійську мову, оскільки модель початково створена саме для цієї мови. Переклад запиту:

«Створи Python функцію, яка обраховує корінь квадратний числа. Включи перевірку чи є вхідні дані числом»

Після відправлення запиту, отримано згенерований файл “generated.py” з таким вмістом:

```python
import math

def calculate_square_root(number):
    if not isinstance(number, (int, float)):
        raise ValueError("Input must be a number")
    if number < 0:
        raise ValueError("Cannot calculate square root of a negative number")

    return math.sqrt(number)
try:
    result = calculate_square_root(12)
    print(result)  # Output: 4.0
except ValueError as e:
    print(e)
```

Рисунок 4.2. – вміст згенерованого файлу “generated.py”

Працездатність згенерованого перевірено, результат є задовільним. Результати можна побачити в таблиці 4.1.

Таблиця 4.1 – результати тестування згенерованого програмного коду

Вхідні дані	**Результат програми**	**Обрахування на калькуляторі**
16	4	4
65025	255	255
4785	69.1736944	69,1736944
8559687	2925.694276	2 925,694276

Як видно із таблиці 4.1, програма знаходить корінь квадратний правильно, отже, програма працює правильно.

Далі, згенерований код проаналізовано інструментом PyLint.

Параметри інструменту залишені за замовчуванням.

```
D:\PythonProjects\OND_Lab1\Codex\Scripts\python.exe D:\PythonProjects\Codex\pylintTest.py
************* Module generated
generated.py:14:0: C0304: Final newline missing (missing-final-newline)
generated.py:1:0: C0114: Missing module docstring (missing-module-docstring)
generated.py:3:0: C0116: Missing function or method docstring (missing-function-docstring)

-----------------------------------
Your code has been rated at 7.50/10
```

Рисунок 4.3. – Результат аналізу згенерованого файлу з програмним кодом

Із результату видно, що згенерованому програмному коду не достає нового рядку в кінці файлу, а також відсутні докстрінги для модуля та функції, яка знаходиться в цьому модулі.

Окрім цього запиту, створено декілька інших, різної складності та деталізації. Загальні результати експерименту наведені в наступному підрозділі та висновках до розділу.

4.2 Переваги та недоліки розробленого продукту

Розроблений продукт має кілька переваг. Однією з них є велика швидкість генерації коду, що є особливо важливим для проєктів з короткими дедлайнами. OpenAI Codex, на якому базується розроблений продукт, виконує завдання генерації коду зі швидкістю, недоступною для людей.

Генерація коду за допомогою Codex показала декілька переваг. По-перше, це швидкість генерації простих скриптів. Тому це зменшує час, який витрачає програміст на написання коду, а також допоможе зменшити кількість помилок, які можуть виникнути при ручному написанні коду.

Після проведення аналізу коду за допомогою PyLint, можна зробити висновок про високу точність та швидкість аналізу згенерованого коду легкої складності. Більш того, завдяки використанню цього інструменту, можна виявити проблеми та не витрачати час на їй пошук вручну, що також зменшує час, який витрачає програміст на написання і перевірку програмного коду.

Однією з головних переваг розробленого продукту є його унікальність, оскільки генерація та аналіз згенерованого коду за допомогою нейромереж є новинкою на території України. Це означає, що розроблений продукт може стати лідером на ринку. Важливо зазначити, що з розвитком штучного інтелекту та машинного навчання, популярність генерації коду за допомогою нейромереж буде зростати.

Проте, варто зазначити, що розроблений продукт має свої недоліки. Один з них полягає у складнощах з настройкою OpenAI Codex і PyLint. Крім того, Codex може виходити за межі своєї компетенції, що може привести до неправильної генерації коду.

OpenAI Codex досить чітко та ефективно генерує програмний код для не складних алгоритмів, але його точність падає при запитах середньої складності та вище. Згенерований код стає не ефективним, швидкість генерації падає, а сам код інколи вже не виконує поставленої задачі.

Подібно Codex, PyLint також показав себе як ефективний інструмент тільки для простих та середніх алгоритмів. Аналізуючи велику кількість коду, PyLint втрачає свою точність, а час аналізу експоненційно зростає.

Таким чином, розроблений продукт має великий потенціал і може стати важливим інструментом для програмістів, які хочуть прискорити свою роботу і отримати більш точний та ефективний код. Проте варто враховувати, що використання продукту поточної версії для великих проєктів є неефективним. В наступному підрозділі наведено пропозиції щодо подальшого розвитку продукту.

4.3 Пропозиції щодо подальшого розвитку продукту

Запропонований спосіб генерації та аналізу програмного коду, який був розроблений у рамках дипломної роботи, є досить ефективним. Однак, як і будь-який інший продукт, він потребує подальшого розвитку. Далі наведено основні напрями розвитку програмного продукту:

4.3.1 Розширення мовної підтримки

Оскільки наразі засіб генерації та аналізу програмного коду працює лише із мовою програмування Python – можна розширити мовну підтримку та використовувати інші мови для генерації коду.

Наразі OpenAI Codex підтримує певний набір програмних мов, таких як Python, JavaScript, Java, C++, Ruby та інші. Можна розширити мовну підтримку і почати підтримувати ці мови також.

4.3.2 Оптимізація генерації коду

Основна мета оптимізації полягає у підвищенні якості та точності згенерованого коду, щоб забезпечити користувачам ще кращий досвід та зменшити час і зусилля, необхідні для внесення правок або виправлення згенерованого коду.

Ось кілька способів оптимізації генерації коду:

1.Врахування контексту: Можна оформити моделі OpenAI Codex доступ до достатньої кількості контексту, щоб вона могла адекватно розуміти та враховувати попередні рядки коду або вказівки, що допомагають зрозуміти потреби користувача. Це дозволить моделі краще розуміти ідею користувача та генерувати відповідний код.

2.Покращення обробки виняткових ситуацій: Можливо розглянути можливості для покращення обробки виняткових ситуацій або неочікуваних вхідних даних. Наприклад, модель може повертати помилку або пропонувати альтернативний код, якщо виявлено несумісність або потенційні проблеми в згенерованому коді.

3.Використання контекстної інформації: Врахувати деталізовану інформацію про потреби та вимоги користувача, яка може бути надана через попередні вказівки, коментарі або анотації. Ця додаткова контекстна інформація може допомогти моделі точніше розуміти та відтворювати інтенцію користувача у згенерованому коді.

4.Створення шаблонів: Створити можливість обирання шаблонів для генерації коду, щоб економити час користувачів та спрощувати роботу для системи.

4.3.3 Розширення аналізу коду та його звітності

Наразі PyLint аналізує лише код, згенерований мовою програмування Python. Але після недавніх оновлень інструменту, PyLint може на ефективному рівні аналізувати також C/C++, Java, JavaScript. Його функціональність обмежена в цих мовах, але перевірка синтаксичних помилок, використанних стандартів оформлення коду, дотримання рекомендацій стилістики та виявленні потенційних проблем у логіці програми можливе.

Однак, щоб максимально ефективно використовувати Pylint, можна врахувати деякі додаткові параметри та можливості, які він надає. Ось деякі з них:

- Конфігураційний файл: Можна створити окремий файл конфігурації для Pylint, де можна налаштувати правила, рівень суворості та інші параметри аналізу. Це дозволяє налаштувати Pylint під свої потреби та вимоги проєкту.

- Включення/виключення правил: Pylint надає можливість включати та виключати конкретні правила аналізу. Це корисно, якщо потрібно зосередитися на конкретних аспектах коду або ігнорувати деякі помилки, які не є потрібні.
- Рівень суворості: Pylint має рівні суворості, які можна налаштувати, вибрати більш суворий рівень, щоб виявити більше проблем, або менш суворий рівень, щоб уникнути зайвих попереджень.

Також буде актуальним рішенням підключити більш ефективні засоби для аналізу коду, які можуть аналізувати на більш детальному рівні, використовуючи навчені моделі.

4.3.4 Безпека та захист

Врахування безпеки є важливим аспектом подальшого розвитку будь-якого продукту.

1.Створити відповідні механізми аутентифікації API для взаємодії з OpenAI Codex та Pylint, авторизації та захисту від несанкціонованого доступу. Використати протоколи шифрування та інші методи безпеки для захисту передачі даних.

2.Захист результатів аналізу: При збереженні результатів аналізу коду та його звітності з Pylint, потрібно врахувати безпеку цих даних. Можна обмежити доступ до збережених файлів з результатами аналізу, обмежуючи права доступу та використовуючи шифрування, якщо необхідно.

4.3.5 Створення вебсайту

Створення вебсайту для продукту може бути ефективною ідеєю з кількох причин:

1. Вебсайт надасть можливість ефективно представити продукт користувачам та зацікавленим сторонам. Буде можливо пояснити, як програма, що використовує OpenAI Codex та аналізатор PyLint, працює і які переваги вона надає.

2. На вебсайті можна розмістити документацію, приклади використання, ресурси для розробників та пояснення щодо інтеграції продукту з OpenAI Codex та PyLint. Це допоможе користувачам краще зрозуміти, як використовувати продукт та отримати максимальну користь з нього.

3. Вебсайт може стати центром для підтримки користувачів, де вони зможуть знайти відповіді на свої питання, звернутися до команди розробників за допомогою форми зв'язку або форуму, а також дізнатися про оновлення та нові функції продукту.

4. Маркетинг та просування: Вебсайт є потужним інструментом для маркетингу та просування продукту. Буде можливість створити привабливий дизайн, розмістити огляди та рекомендації, запропонувати демонстраційні відео та інші матеріали, що сприятимуть привернення уваги потенційних користувачів та клієнтів.

5. Розширення функціональності: Через вебсайт можна розширити функціональність продукту, наприклад, надати можливість реєстрації та авторизації користувачів, зберігання їх налаштувань, статистику використання та інші функції, які поліпшать користувацький досвід.

4.3.6 Масштабованість

Масштабованість є важливим аспектом розвитку продукту, оскільки вона дозволяє продукту ефективно функціонувати при зростанні обсягів даних, користувачів або складності завдань. Ось декілька пропозицій щодо масштабованості, які можуть сприяти подальшому розвитку продукту:

1. Створити гнучкість архітектури продукту, щоб вона легко адаптувалась до змінних потреб і можливих зростань обсягів даних чи користувачів. Можна розглянути використання мікросервісної архітектури або інших модульних підходів, які дозволяють горизонтально масштабувати окремі компоненти.

2. Гнучкість інфраструктури: Можна розглянути можливість використання хмарних сервісів або контейнеризації, що дозволить легко масштабувати інфраструктуру продукту за потреби.

4.3.7 Відгуки користувачів

Отримання відгуків користувачів є надзвичайно важливим етапом для подальшого розвитку продукту. Ці відгуки можуть надати безцінну інформацію щодо сильних сторін продукту та проблем, з якими стикаються користувачі. Крім того, відгуки можуть містити пропозиції щодо покращень, що допоможе у покращенні якості та функціональності продукту.

Залучення користувачів до процесу розвитку продукту є дуже важливим кроком, оскільки це дозволяє створити продукт, який точніше відповідає їхнім потребам та очікуванням. Відгуки користувачів можуть допомогти виявити ті аспекти продукту, які можуть бути поліпшені та вдосконалені. Це може збільшити задоволення користувачів від продукту та покращити їхнє сприйняття його як цілісної системи.

Крім того, залучення користувачів до процесу розвитку продукту може допомогти забезпечити зворотній зв'язок від користувачів, що сприятиме створенню більш ефективної комунікації між розробниками та користувачами. Це допоможе у забезпеченні співпраці та задоволеності користувачів.

Отже, залучення користувачів до процесу розвитку продукту є дуже важливим кроком, який допоможе покращити якість та функціональність продукту, а також забезпечити більш ефективну комунікацію між розробниками та користувачами.

ВИСНОВКИ ДО РОЗДІЛУ 4

У четвертому розділі проведено аналіз ефективності розробленої системи генерації програмного коду за допомогою моделі глибокого навчання OpenAI Codex та аналізу згенерованого програмного коду інструментом PyLint.

Розроблений продукт, який базується на OpenAI Codex, має кілька переваг, зокрема велику швидкість генерації коду, зменшення часу написання коду і виявлення помилок, а також його унікальність на ринку України. Однак, продукт також має недоліки, зокрема складнощі з настройкою Codex і PyLint, можливість неправильної генерації коду та зниження точності та швидкості генерації для складних алгоритмів. Крім того, PyLint втрачає точність і збільшує час аналізу при обробці великих обсягів коду. Загалом, розроблений продукт має потенціал стати важливим інструментом для прискорення роботи програмістів, проте для великих проєктів потрібно удосконалювати його ефективність.

Для подальшого розвитку продукту запропоновано кілька варіантів, зокрема розширення мовної підтримки, оптимізацію генерації коду, розширення аналізу коду та його звітності, забезпечення безпеки та захисту, створення вебсайту, масштабованість та збір відгуків користувачів. Пропозиції щодо подальшого розвитку продукту спрямовані на покращення функціональності, розширення можливостей та забезпечення безпеки та зручності використання.

У сучасному світі конкурентів у сфері автоматичної генерації та аналізу програмного коду немає. Тому така система є досить актуальною та її розробка є ефективним рішенням для спрощення життя розробникам ПЗ.

ВИСНОВКИ

Під час виконання дипломної роботи створено систему аналізу згенерованого програмного коду, використовуючи нейромережу ChatGPT.

У першому розділі цієї роботи проведено теоретичний огляд та аналіз особливостей різних засобів для генерації програмного коду та його аналізу. Розглянуто багато сервісів та інструментів, в кожного з яких свої переваги та недоліки. Обрано такі інструменти як модель глибокого навчання OpenAI Codex для генерації програмного коду, та розширення PyLint для аналізу згенерованого програмного коду.

У рамках другого розділу проведено дослідження обраних інтсрументів, зокрема їх особливостей, можливостей та способів під'єднання. Обрано мову програмування Python, на якій написана система. Також поставлено декілька задач, які необхідно виконати для досягнення мети дипломної роботи.

У третьому розділі дипломної роботи в якому описана розробка та реалізація алгоритму генерації та аналізу програмного коду, згенерованого нейромережею, викладено докладний опис розробки та реалізації запропонованого методу, який дозволяє генерувати програмний код за допомогою OpenAI Codex та аналізувати його за допомогою PyLint. Цей метод забезпечує швидке та ефективне створення початкового коду для подальшої роботи, що зменшує зусилля, необхідні для початку проєкту.

Четвертий розділ стосується тестування розробленого продукту та його можливостей. Результати показали, що система генерує програмний код правильно в 80% запитів. Аналіз цього коду показав понад 95% проблем, які перед цим знайдені власноруч. Таким чином запропонований спосіб показав себе як ефективний для генерації та аналізу програмного коду із доволі високою точністю. Точність та ефективність падає при збільшені складності запитів, тому ситсему потрібно покарщувати для досягненя необхідних цілей.

В розділі розглянуто пропозиції для подальшого розвитку продукту, що спрямовані на покращення функціональності, розширення можливостей та забезпечення безпеки й зручності використання. Такі пропозиції є важливим етапом у розвитку продукту, оскільки дозволяють підвищити ефективність та зручність використання методу, а також забезпечити високу якість програмного забезпечення.

Таким чином, метод генерації та аналізу програмного коду, розроблений у рамках дипломної роботи, може стати ефективним інструментом для команди розробників. Він дозволяє зекономити час та зусилля, спростити процес розробки програмного забезпечення і забезпечити високу якість коду.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SonarQube документація [Електронний ресурс] – Режим доступу до ресурсу: https://docs.sonarqube.org/latest/.
2. Блог CodeClimate [Електронний ресурс] – Режим доступу до ресурсу: https://codeclimate.com/blog.
3. Coverity SAST [Електронний ресурс] – Режим доступу до ресурсу: https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html.
4. Можливості PyLint [Електронний ресурс] – Режим доступу до ресурсу:https://ubunlog.com/uk/pylint-herramienta-analisis-codigo-python/.
5. Блог Kite [Електронний ресурс] – Режим доступу до ресурсу: https://www.kite.com/blog/.
6. Партнер GitHub Copilot [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/features/copilot.
7. OpenAI Codex [Електронний ресурс] – Режим доступу до ресурсу: https://openai.com/blog/openai-codex.
8. Introducing ChatGPT [Електронний ресурс] – Режим доступу до ресурсу: https://openai.com/blog/chatgpt.
9. Coding Made AI—Now, How Will AI Unmake Coding? [Електронний ресурс] – Режим доступу до ресурсу: https://spectrum.ieee.org/ai-code-generation-language-models.
10. What developers need to know about generative AI [Електронний ресурс] – Режим доступу до ресурсу: https://github.blog/2023-04-07-what-developers-need-to-know-about-generative-ai/.
11. Документація PyLint [Електронний ресурс] – Режим доступу до ресурсу: https://pylint.readthedocs.io/en/latest/.

12. Gewirtz D. How to use ChatGPT to write code [Електронний ресурс] / Dawid Gewirtz // ZDNET. – 2023. – Режим доступу до ресурсу: https://www.zdnet.com/article/how-to-use-chatgpt-to-write-code/.
13. Style Guide for Python Code [Електронний ресурс] – Режим доступу до ресурсу: https://peps.python.org/pep-0008/.
14. Promt Engineering [Електронний ресурс] – Режим доступу до ресурсу: https://www.promptingguide.ai/.
15. Code Completion [Електронний ресурс] – Режим доступу до ресурсу: https://platform.openai.com/docs/guides/code.
16. Документування коду [Електронний ресурс] – Режим доступу до ресурсу: https://highload.today/uk/blogs/navishho-i-yak-dokumentuvati-kod-na-python-osnovni-kroki-ta-poradi-rozrobnika/.
17. The Nine Steps of Software Product Development Life Cycle [Електронний ресурс] – Режим доступу до ресурсу: https://www.turing.com/blog/software-product-development-life-cycle/.
18. Kennedy M. Effective PyCharm / M. Kennedy, M. Harrison., 2019. – 221 с.

Додаток 1

Спосіб аналізу програмного коду,
згенерованого нейромережею ChatGPT

Діаграма застосунку
(Функціональна схема)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2023 р

					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Манчук М. В.			Спосіб аналізу програмного коду, згенерованого нейромережею ChatGPT Діаграма застосунку (Функціональна схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Міщенко Л. Д.					
Н. Контр.		Виноградов Ю.М.				КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-93	
Затвердив							

Додаток 2

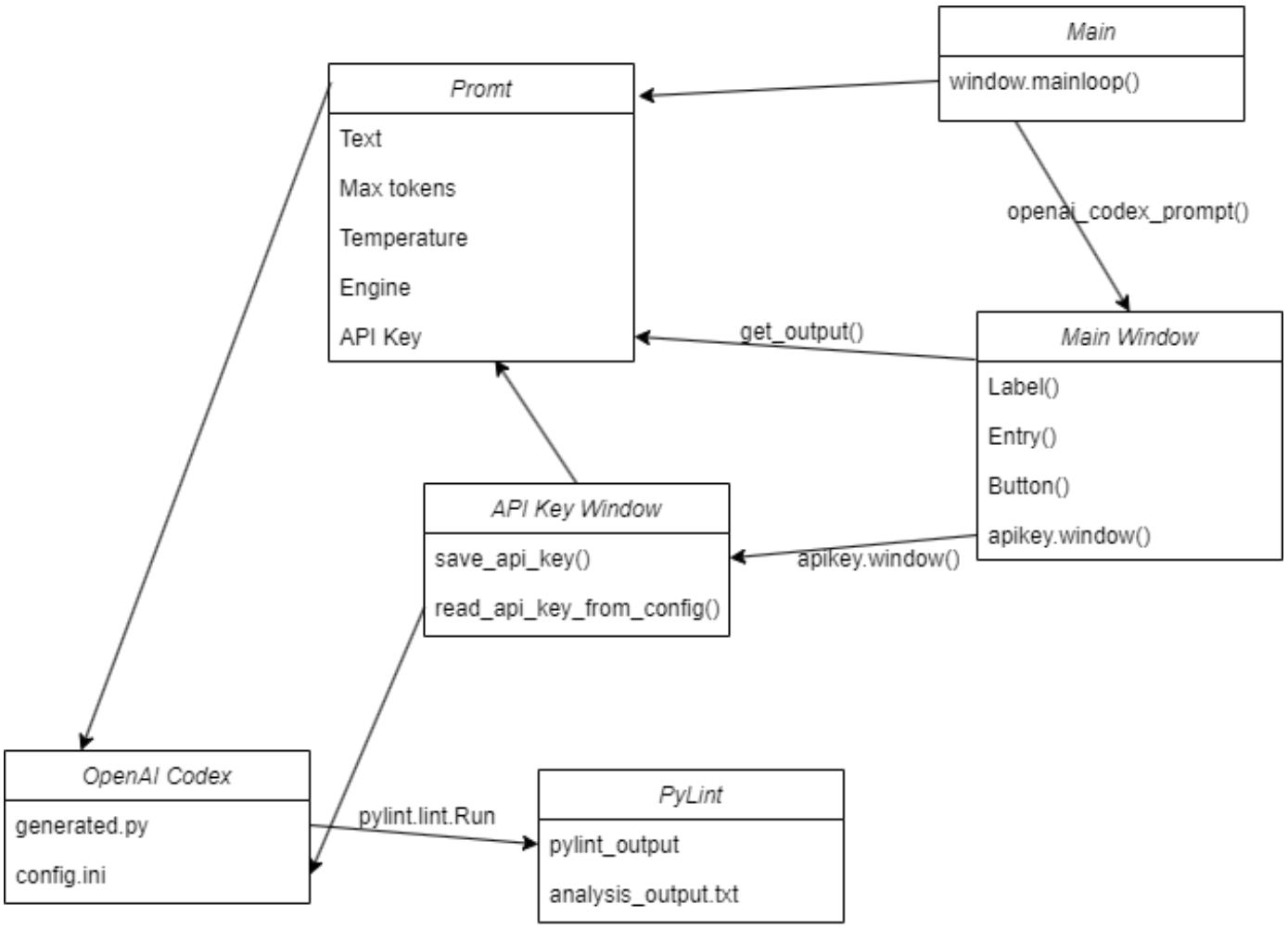
Спосіб аналізу програмного коду,
згенерованого нейромережею ChatGPT

Діаграма класів
(Функціональна схема)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.005 Д2		
		№ докум.	Підпис	Дата			
Розробив		Манчук М.В.			Спосіб аналізу програмного коду, згенерованого нейромережею ChatGPT Діаграма класів (Функціональна схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Міщенко Л.Д.					
Н. Контр.		Виноградов Ю.М.				КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-93	
Затвердив							

Додаток З

Спосіб аналізу програмного коду,
згенерованого нейромережею ChatGPT

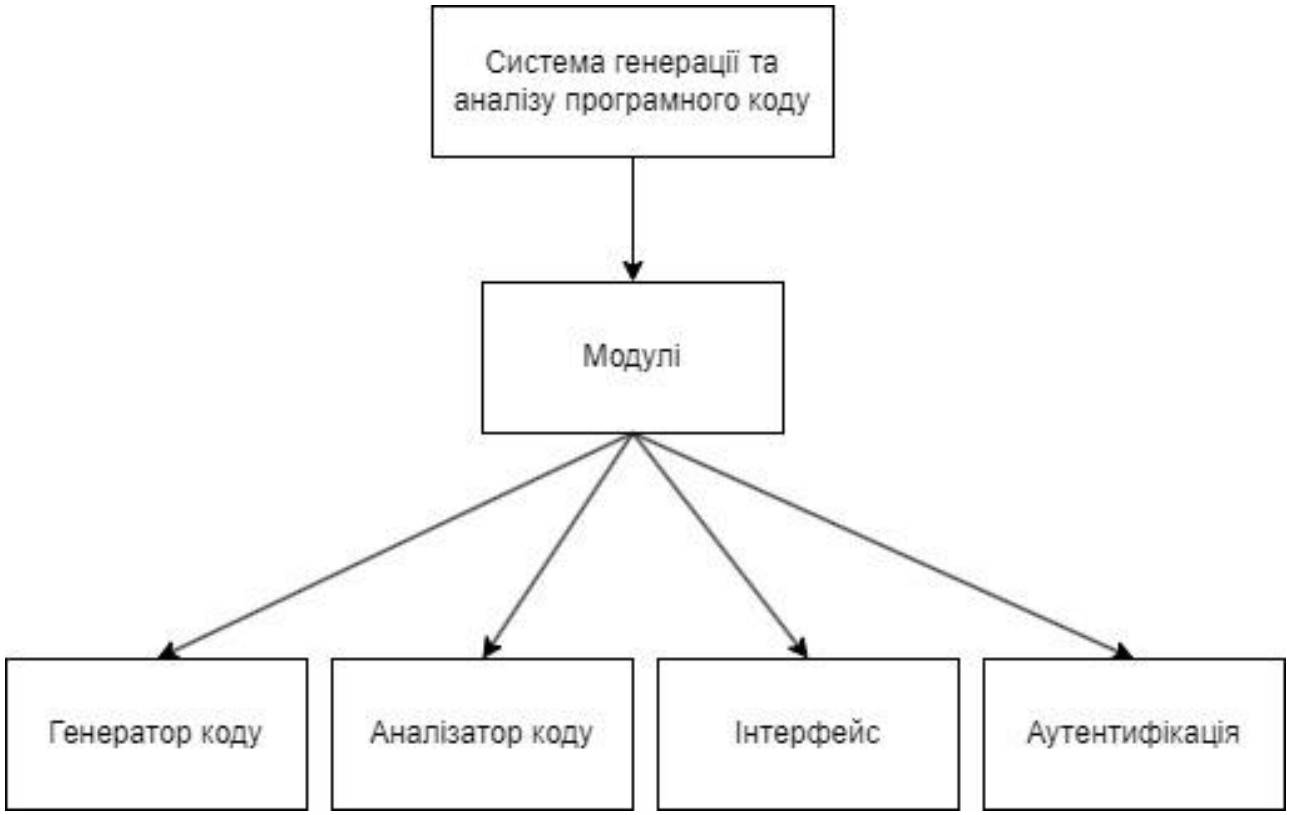
Структура програмного забезпечення

(Структурна схема)

ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ 2023 р



					ІАЛЦ.467200.006 ДЗ		
		№ докум.	Підпис	Дата			
Розробив		Манчук М.В.			Спосіб аналізу програмного коду, згенерованого нейромережею ChatGPT Структура ПЗ (Структура схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Міщенко Л.Д.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-93	
Н. Контр.		Виноградов Ю.М.					
Затвердив							

Додаток 4

Спосіб аналізу програмного коду,
згенерованого нейромережею ChatGPT

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 3

Київ 2023 р

```python
import tkinter as tk
import openai
import pylint.lint
import apikey

API_KEY = apikey.read_api_key_from_config()
openai.api_key = API_KEY


def openai_codex_prompt(prompt, max_tokens, temperature):
    response = openai.Completion.create(
        engine='davinci-codex',
        prompt=prompt,
        max_tokens=max_tokens,
        temperature=temperature
    )

    if response and response.choices:
        return response.choices[0].text
    else:
        return 'Помилка при виконанні запиту до OpenAI Codex'


def get_output():
    prompt = entry.get()
    max_tokens = int(max_tokens_entry.get())
    temperature = float(temperature_entry.get())

    output = openai_codex_prompt(prompt, max_tokens, temperature)
    label['text'] = output


    with open('generated.py', 'w') as file:
        file.write(output)


    pylint_output = pylint.lint.Run(['generated.py'], exit=False).linter.stats


    with open('analysis_result.txt', 'w') as file:
        file.write(str(pylint_output))


window = tk.Tk()
window.title("OpenAI Codex Інтерфейс")
window.geometry("400x400")
window.configure(bg="lightgray")


prompt_label = tk.Label(window, text="Введіть текст запиту для генерації коду:",
font=("Arial", 12), bg="lightgray")
prompt_label.pack(pady=10)


entry = tk.Entry(window, font=("Arial", 10), width=40)
entry.pack()
```

```
max_tokens_label = tk.Label(window, text="Максимальна кількість токенів:",
font=("Arial", 12), bg="lightgray")
max_tokens_label.pack(pady=10)


max_tokens_entry = tk.Entry(window, font=("Arial", 10), width=10)
max_tokens_entry.pack()


temperature_label = tk.Label(window, text="Температура:", font=("Arial", 12),
bg="lightgray")
temperature_label.pack(pady=10)


temperature_entry = tk.Entry(window, font=("Arial", 10), width=10)
temperature_entry.pack()


generate_button = tk.Button(window, text="Генерувати", font=("Arial", 12),
command=get_output)
generate_button.pack(pady=10)


apikey_button = tk.Button(window, text="Налштувати ключ API", font=("Arial",
12), command=apikey.apikey_window)
apikey_button.pack(pady=10)


label = tk.Label(window, text="", font=("Arial", 12), bg="lightgray")
label.pack()


window.mainloop()

import tkinter as tk
import configparser


def read_api_key_from_config():
    config = configparser.ConfigParser()
    config.read('config.ini')


    if 'DEFAULT' in config and 'API_KEY' in config['DEFAULT']:
        api_key = config['DEFAULT']['API_KEY']
        return api_key
    else:
        return None


def apikey_window():


    window = tk.Tk()
    window.title("API-ключ OpenAI")
    window.geometry("400x150")
    window.configure(bg="lightgray")

    def save_api_key():
        api_key = entry.get()
```

```
        config = configparser.ConfigParser()
        config['DEFAULT'] = {'API_KEY': api_key}

        with open('config.ini', 'w') as file:
            config.write(file)


        window.destroy()

    label = tk.Label(window, text="Введіть ваш API-ключ OpenAI:", font=("Arial",
12), bg="lightgray")
    label.pack(pady=20)


    entry = tk.Entry(window, font=("Arial", 10), width=40)
    entry.pack()


    save_button = tk.Button(window, text="Зберегти", font=("Arial", 12),
command=save_api_key)
    save_button.pack(pady=10)


    window.mainloop()
```