ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Штучний інтелект у процесах діагностики технічних систем»

на здобуття освітнього ступеня бакалавра
зі спеціальності 124 «Системний аналіз»
(код, найменування спеціальності)

освітньо-професійної програми Інтелектуальні системи управління
(назва)

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

_____ Кирило ФІЛІПОВ_____
(підпис)

Виконав: здобувач вищої освіти гр. САД-41
Кирило ФІЛІПОВ
_____

Керівник: Михайло КУЗЬМІЧ
_____
к.т.н., доцент Ім’я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім’я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра «Інформаційних систем та технологій»

Ступінь вищої освіти Бакалавр

Спеціальність 124«Системний аналіз»

Освітньо-професійна програма Інтелектуальні системи управління

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК

«____» _____ 20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Філіпову Кирилу Олександровичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Штучний інтелект у процесах діагностики технічних систем»

керівник кваліфікаційної роботи Михайло КУЗЬМІЧ, Доктор філософських наук, доцент
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «___» _____ 20__ р. № ___

2. Строк подання кваліфікаційної роботи «___» _____ 20__ р.

3. Вихідні дані до кваліфікаційної роботи:

- використання мови програмування Python;
- розробка програми з використанням додатку Windows Notepad.

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

1. Проаналізувати проблеми використання штучного інтелекту в діагностиці технічних систем, виявити напрямки існуючих досліджень.
2. Розробити метод для діагностики програмного забезпечення за допомогою штучного інтелекту.
3. Створити та протестувати прототип інтелектуальної системи для діагностики помилок в логах комп'ютерних програмах.

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «___» _____ 20__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз існуючих процесів діагностики інформаційних систем	27.02-11.03.25	Виконано
2	Загальна характеристика використання штучного інтелекту в діагностиці систем	12.03-18.03.25	Виконано
3	Створення програми для виявлення помилок в технічних системах на базі штучного інтелекту	19.03-24.03.25	Виконано
4	Тестування програми	24.03.-25.03.25	Виконано
5	Опрацювання в програмі різних даних для виявлення помилок	26.03.-27.03.25	Виконано
6	Використання програми для аналізу помилок в логах	27.03.-28.03.25	Виконано
7	Підсумок отриманих результатів використання програми	05.05.-26.05.25	Виконано
8	Підготовка заключних висновків та презентації до кваліфікаційної роботи	26.05.-03.06.25	Виконано

Здобувач (ка) вищої освіти _____ Кирило ФІЛІПОВ
(підпис) *Ім'я, ПРІЗВИЩЕ*

Керівник
кваліфікаційної роботи _____ Михайло КУЗЬМІЧ
(підпис) *Ім'я, ПРІЗВИЩЕ*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 78 стор., 8 рис., 6 табл., 20 джерел.

Мета роботи – визначення пріоритетних напрямків і завдань для діагностики технічних систем з використанням штучного інтелекту.

Об’єкт дослідження – дослідження використання штучного інтелекту в діагностиці технічних систем.

Предмет дослідження – використання штучного інтелекту для виявлення помилок при діагностиці технічних систем.

При проведенні дослідження було використано AI на базі MLPClassifier та системи інтеграції API від OpenAI.

КЛЮЧОВІ СЛОВА: ТЕХНІЧНІ СИСТЕМИ, ШТУЧНИЙ ІНТЕЛЕКТ, НЕЙРОННА МЕРЕЖА, ДІАГНОСТИКА СИСТЕМИ, ЛОГИ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ТЕСТУВАННЯ, ЕКСПЕРНІ СИСТЕМИ, МАШИННЕ НАВЧАННЯ.

Зміст

ВСТУП

Діагностика важлива для визначення стану різних технічних систем в процесі всього циклу їх існування та експлуатації. Саме вона визначає технічний стан та можливості використання об'єкта. Процес діагностики складається зі збору та аналізу інформації про роботу технічної системи для виявлення і виправлення несправності та помилок.

Одним із перспективних напрямків діагностики стану технічних систем є застосування штучного інтелекту. Використання штучного інтелекту допоможе аналізувати прогнозне виявлення поломок обладнання та енергетичних систем, оцінювати стан медичного обладнання та інтерпретувати результати медичної діагностики, визначати стан зносу архітектурних будівель та мостів, підвищити кібербезпеку шляхом аналізу мережевого трафіку та виявлення шкідливого програмного забезпечення.

Важливою складовою діагностики програмного забезпечення є процес виявлення помилок. Для цього використовують лог-файли. Це тестові документи, в яких зафіксовані всі події, що були виконані у комп'ютерній системі. Вони допомагають ідентифікувати та усунути помилку на сайті або в програмному забезпеченні.

Сучасне програмне забезпечення має високий рівень складності, розвинену архітектуру та великою кількістю компонентів, що значно ускладнює процес виявлення та виправлення помилок.

Актуальність теми полягає в потенціалі використання штучного інтелекту для аналізу великого обсягу даних для прогнозування потенційних проблем у технічних системах. Великі технологічні компанії, такі як Google, Microsoft, Meta, Apple, Amazon багато інвестують в розробку та впровадження штучного інтелекту, бо вважають його ключовою технологією для зростання і розвитку інновацій у майбутньому. Українські компанії все активніше використовують впровадження штучного інтелекту в своїй діяльності для створення чат-ботів, консультацій клієнтів та навіть вибору товарів.

Основною метою кваліфікаційної роботи є визначення пріоритетних напрямів і основних завдань використання штучного інтелекту для діагностики програмного забезпечення та підвищення їх ефективності.

Об’єктом дослідження є процес використання штучного інтелекту для діагностики технічних систем в роботі однієї з найдавніших технічних систем, а саме АТС. Дослідження засноване на досвіді роботи в провідній телекомунікаційній компанії України.

Предметом мого дослідження є використання штучного інтелекту для виявлення помилок в логах за допомогою застосування програми AI на базі MLPClassifier та системи інтеграції API від OpenAI.

Дані дослідження можуть використовуватися на практиці в будь-яких компаніях, що використовують АТС та веб-сайти у своїй роботі. Результати дослідження також можуть бути інтегровані в будь-які системні розробки програмного забезпечення, підвищуючи швидкість їх роботи та стабільність функціонування з можливістю доопрацювання.

Для виконання роботи потрібно вирішити такі завдання:

1. Проаналізувати с сучасні проблеми використання штучного інтелекту в процесах діагностики технічних систем, виявити напрямки існуючих досліджень.
2. Провести дослідження застосування існуючих методів використання штучного інтелекту в роботі сучасних технічних систем.
3. Розробити метод для діагностики програмного забезпечення за допомогою штучного інтелекту.
4. Створити прототип інтелектуальної системи для діагностики помилок в логах комп’ютерних програмах.
5. Протестувати розроблену систему на реальних даних.
6. Надати практичні рекомендації для впровадження штучного інтелекту в розробці та підтримці комп’ютерних програм.

Для вирішення поставлених завдань буде використано комплекс із методів дослідження, що доповнюють один одного.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ШТУЧНИЙ ІНТЕЛЕКТ ТА ДІАГНОСТИКУ ТЕХНІЧНИХ СИСТЕМ

1.1 Методи використання штучного інтелекту в діагностиці технічних систем

В основі процесу діагностики технічної системи є оцінка технічного стану об’єкту. В залежності від методів впливу на об’єкт розрізняють тестову та функціональну діагностику. Тестова діагностика передбачає подачу до компонентів системи тестових сигналів для виявлення технічних несправності в роботі системи, а робоча діагностика проводиться під час роботи технічної системи в реальних умовах навантаження. Тестове діагностування в основному проводиться коли об’єкт не використовується за призначенням, тоді як функціональне діагностування проводиться під час його роботи.

Діагностика відрізняється відносно часу її виконання. Так періодична діагностика виконується у запланований час з певними інтервалами або коли виникають проблеми в роботі системи. Безперервна ж діагностика використовується при високих ризиках у технічних системах.

Способів виявлення несправності технічних систем є кілька : органолептичний, при використанні органів чуттів людини; інструментальний, з використанням обладнання для виявлення недоліків; статистичний, що заснований на аналізі накопиченої інформації про технічну систему.

Щоб обрати метод діагностики технічної системи, важливо встановити до якого типу вона відноситься, розуміти її призначення, умови експлуатації та фактори ризику.

Штучний інтелект є сукупністю інформаційних технологій, що дозволяє виконувати складні комплексні задачі за допомогою наукових методів досліджень і алгоритмів обробки інформації. З його допомогою можна створювати власні бази знань, складні моделі прийняття рішень та алгоритми роботи з великими обсягами інформації.

Застосування штучного інтелекту в діагностиці технічних систем має велике значення, оскільки він здатний виконувати завдання, які виконують люди. Штучний інтелект подібний до людини своєю здатністю навчатися та покращувати набуті навички в управлінні та обробці великих обсягів даних.

На сучасному етапі розвитку штучного інтелекту можна виділити символьний і коненціоністський напрямки. Символьний метод, або семіотичний базується на логічних міркуваннях та математичних обчисленнях. Відповідно до цього методу інтелектуальні системи використовують символи та правила, які відображають знання про предметну область. Цей метод дає змогу формувати системи, що вміють робити логічні висновки та приймати рішення на основі структурованих даних, необхідних для розробки експертних систем визначення дефектів програмного забезпечення.

Конекціоністський метод, або висхідний базується на моделюванні нейронної мережі і передбачає, що інтелектуальна поведінка виникає внаслідок паралельної обробки інформації численними штучними нейронами, що моделюють поведінку біологічних нейронів.

Нейронні мережі досягли чудових результатів у розпізнаванні зображень, класифікації та прогнозуванні і саме це робить їх цінним інструментом для виявлення поведінкових аномалій програмних систем.

Машинне навчання, як окрема дисципліна вивчення штучного інтелекту, зосереджена на розробці алгоритмів для прогнозування виявлення дефектів та класифікації помилок для покращення процесів тестування. До машинного навчання штучного інтелекту належать супервізований і несупервізований підхід. Причому супервізований підхід використовують для виявлення знайомих типів помилок, а несупервізований для виявлення не типових помилок для структуризації даних.

Розділом машинного навчання є глибоке навчання, засноване на глибоких нейронних мережах з численними прихованими шарами, які виконують лінійні або нелінійні перетворення. Нейронні мережі використовують штучні нейрони, які виконують математичні обчислення подібні до роботи нейронів мозку

людини. Навчання нейронних мереж успішно проводять з використанням великих обсягів інформації і їх використання дуже поширене в діагностиці. Згорткові нейронні мережі адаптовані до обробки зображень і застосовуються для аналізу та поліпшення якості візуальних представлень систем програми, таких як сегментація зображення та поліпшення його чіткості. Рекурентні нейронні мережі та трансформери використовують зворотній зв'язок для аналізу послідовностей даних, таких як логи програми і послідовності викликів функцій.

Особливе місце в галузі штучного інтелекту має обробка природної мови, що дозволяє аналізувати текст в документах і коментарях коду та визначати знайдені помилки. Методи векторного представлення слів та розмовні моделі оцінюють семантичну схожість між різними програмними артефактами, що сприяє більш ефективному пошуку та категоризації дефектів.

Еволюційні обчислення належать до розділу еволюційного моделювання і побудовані на принципах природного відбору та генетичних механізмах для пошуку оптимального вирішення складних проблем моделюючи процеси природного відбору. Генетичне програмування використовують при оптимізації програмного коду та автоматичному виправленні помилок. В контексті еволюційних обчислень розрізняють три напрямки, такі як генетичні алгоритми, еволюційна стратегія та еволюційне програмування. Ці методи дозволяють генерувати й оцінювати різноманітні варіанти рішень на шляху до оптимального результату.

Експертні системи, як програмні комплекси, поєднують в собі бази знань різних експертів та механізми логічного виведення, що є важливим інструментом діагностики програмного забезпечення. Експертні бази знань містять дані про типові помилки та методи їх усунення, що дозволяє автоматизувати процес діагностики та надати рекомендації розробникам програмного забезпечення.

Визначення моделей є важливим аспектом використання штучного інтелекту. Розробники та тестувальники програмного забезпечення мають критично оцінювати прийняття рішень штучним інтелектом і при необхідності вміти змінити або доопрацювати хибні чи сумнівні результати.

Етичні аспекти застосування штучного інтелекту мають особливу увагу, адже його застосування тягне за собою права, обов’язки та відповідальність. Питання відповідальності за помилки, які допустив штучний інтелект, конфіденційність використовуваних даних для навчання моделі та упередженість алгоритмів є актуальними в зв’язку з зростанням сфер застосування інтелектуальних систем. Розробка етичних принципів та стандартів використання штучного інтелекту є важливим кроком для забезпечення правового врегулювання цієї галузі в Україні та світі в цілому.

Отже, сучасні концепції та методи штучного інтелекту мають величезний потенціал для вирішення таких складних завдань, як діагностика та тестування програмного забезпечення. Інтеграція різних технологій та підходів штучного інтелекту дозоляє створювати ефективні системи автоматизованої діагностики, що мають здатність до навчання, самовдосконалення та адаптації. Штучний інтелект відкриває нові перспективи для підвищення якості та надійності технічних систем.

1.2 Сучасні методи діагностики та тестування програмного забезпечення

Діагностика та тестування програмного забезпечення відіграють важливе значення в розробці програмного забезпечення, вони спрямовані на виявлення, локалізацію та усунення помилок, перевірку відповідності систем встановленим вимогам. З сучасним технологічним розвитком програмні системи стають все складнішими, тому традиційні підходи до діагностики стають менш ефективними, а це викликає потребу в розробці і впровадженні нових інтелектуальних методів для автоматизації програмного продукту.

Виділяють два типи тестування програмного забезпечення: статичне та динамічне тестування. Статичне тестування аналізує вихідний код програми без виконання тестової програми. При цьому підході виявляють синтаксичні помилки, порушення стандартів кодування, потенційну вразливість та інші дефекти на ранніх етапах розробки. Динамічне тестування використовують з

метою виконання програми для виявлення помилок у її роботі. Такий підхід має такі рівні, як модульне або компонентне, інтеграційне та системне тестування, тестування продуктивності або приймальне тестування. Модульне тестування зосереджене на пошуках дефектів в окремих модулях програми, таких як класи та функції, щоб переконатися, що вони працюють відповідно специфікацій. Інтеграційне тестування перевіряє правильність взаємодії між різними компонентами системи та взаємодію програми з середовищем виконання, а системне тестування оцінює роботу програми в цілому. Методи та можливості різних рівнів тестування наведені в Таблиці 1.1.

Назва рівню тестування	Об’єкт	Метод	Можливості
Модульне (компонентне)	Модулі Класи Функції Процедури	«Біла скринька», «Чорна скринька»	Виявлення дефектів на ранній стадії розробки. Підвищення розуміння коду та його структури. Спрощення інтеграції окремих компонентів. Підвищення продуктивності роботи окремих компонентів системи.
Інтеграційне	Групи модулів Підсистеми Зовнішні інтерфейси (API, бази даних та інше)	Знизу вгору, Зверху вниз, Спрямоване, Сендвіч-тестування	Виявлення дефектів у взаємодії компонентів з середовищем. Перевірка правильності передачі даних та обміну інформацією. Оцінка сумісності різних частин системи.
Системне	Інтегрована система	«Чорна скринька»	Оцінка якості системи в цілому. Перевірка системи на відповідність вимогам специфікації. Виявлення проблем при взаємодії всіх компонентів системи. Перевірка зручності та функціональності системи.
Приймальне (продуктивне)	Завершена система в робочому середовищі	Кероване, Операційне, Альфа і бета-тестування	Підтвердження готовності системи до використання. Внесення остаточних виправлень в систему. Підвищення рівня впевненості

			в успішності впровадження системи.

Таблиця 1.1 Методи та можливості різних рівнів тестування.

Відповідно до ступеня автоматизації розрізняють ручне та автоматизоване тестування. Ручне тестування виконує людина. В останні роки значно розвинулося саме автоматизоване тестування через низку переваг. Автоматизоване тестування використовує спеціальні інструменти та фреймворки для створення, виконання та аналізу тестів. Цей підхід дозволяє заощадити час на тестування та підвищити точність результатів. За потреби, при автоматизованому тестуванні можливе регулярне проведення регресійного тестування для виявлення помилок, що виникають внаслідок змін коду. На практиці часто використовується поєднання ручного та автоматизованого тестування, що значно підвищує ефективність роботи системи та має гнучкість в застосуванні.

Залежно від фази тестування розрізняють тестування чорного, сірого та білого ящика, які розрізняються рівнем знання внутрішньої будови системи. Тестування методом чорного ящика передбачає використання тільки зовнішнього інтерфейсу програми, тобто без використання початкового коду програми і успішно використовується в бета-тестуванні. Тестування методом білого ящика засноване на знанні коду та аналізі внутрішньої структури системи і використовується на всіх рівнях тестування. Відповідно тестування методом сірого ящика комбінує метод чорного і білого ящика і часто використовується для тестування веб-додатків. Переваги і недоліки цих методів наведені в Таблиці 1.2.

Назва методу тестування	Переваги	Недоліки
«Чорна скринька»	Тестування без заглиблення в код значно економить час. Фокусування на проблемах інтерфейсу користувача. Швидке виявлення дефектів інтеграції.	Складність визначити походження виявлених помилок. Великі затрати на виявлення конкретного місця помилки в системі через незнання коду. Неможливість проведення повного тестування.
«Біла скринька»	Є доступ до специфікації і вихідного коду. Висока	Потрібні навички програмування. Аналіз

	точність визначення причини та місця помилки. Можливість виявлення прихованих помилок та дефектів.	великих обсягів коду затратне по часу. Застосування для невеликих за обсягом програмних систем. Можна знаходити і витрачати час на помилки, які не впливають на функціонування системи.
«Сіра скринька»	Використання специфікації та архітектурної побудови системи. Рівновага між ефективністю та деталізацією. Краще розуміння помилок ніж в «Чорній скриньці». Менша трудомісткість порівняно з «Білою скринькою».	Немає доступу до вихідного коду

Таблиця 1.2 Переваги та недоліки методів тестування чорної, білої та сірої скриньки.

Ефективне тестування програмного забезпечення включає позитивний та негативний сценарії для перевірки функціональності системи. В першу чергу необхідно провести тестування позитивного сценарію, метою якого є перевірка функціональних можливостей програми при коректному введені даних в стандартних умовах використання системи. В результаті тестування програма має виконувати операції, що передбачені специфікацією без помилок і відображати коректні результати. Так як позитивне тестування перевіряє більшу частину роботи програми, то часто розробники нехтують негативним сценарієм. Але для ефективної роботи програми необхідно проводити обидва сценарії. При негативному сценарії тестування проводиться методом вводу некоректних даних або імітує спробу злому програми. Саме негативний сценарій допомагає відкоригувати роботу програми при некоректному використанні та уникнути

таких помилок, як збій в роботі, припинення реагування програми або навіть доступ до конфіденційних даних системи. Тому не дивлячись на великі затрати часу та фінансів не треба нехтувати ефективністю програмного продукту.

Отже, сучасні підходи до діагностики та тестування програмного забезпечення характеризуються високим рівнем автоматизації, використанням різноманітних методів та технологій, а також інтеграцією в процесами розробки та впровадження програмного забезпечення. Розвиток цих підходів спрямований на підвищення ефективності виявлення помилок, забезпечення надійності та стабільності роботи програмних систем в умовах зростаючої складності сучасного технологічного середовища з високими вимогами до конкурентоспроможності програмного продукту.

1.3 Підготовка технічної документації для діагностики програмного забезпечення з використанням штучного інтелекту

Підготовка технічної документації є важливим етапом для діагностики програмного забезпечення і тому вимагає структурованого підходу. Саме якість підготовленої документації дозволяє штучному інтелекту вірно розуміти дані.

Роль технічної документації важко переоцінити, адже саме вона надає штучному інтелекту структуровану інформацію. Вона містить опис типових сценаріїв помилок, відмов системи та виникнення можливих аномалій роботи системи. Правильне розуміння логів, тобто повідомлень про помилки, також забезпечує технічна документація.

Основними вимогами для документації діагностики штучним інтелектом є її структура. Розділи повинні мати логічний розподіл. Для однакових помилок має бути застосована однакова термінологія, а опис має бути чітким і з мінімальною кількістю слів. Списки та таблиці також допоможуть структурувати технічну документацію.

На першому етапі підготовки необхідно провести аналіз вихідних даних. Зібрати існуючі повідомлення про помилки. Якщо вже проводилось тестування,

то необхідно додати історії тестів системи. На цьому етапі важливо визначити типові класи помилок, саме це заощадить багато часу при впровадженні програмного забезпечення. Нижче наведена таблиця 1.3, що містить приклад структури технічної документації.

Код помилки	Опис помилки	Причина виникнення	Рішення
ERR_001	Помилка з’єднання	Відсутність доступу до мережі	Перевірити підключення та налаштування мережі
ERR_002	Помилка Тайм-ауту відповіді	Тимчасова недоступність сервера	Спробувати пізніше
ERR_003	Некоректний запит	Некоректний формат даних	Перевірити формат даних
ERR_004	Послуга недоступна	Тимчасова недоступність сервера	Спробувати пізніше

Таблиця 1.3 Структура технічної документації.

Для стандартизації помилок важливо чітко сформулювати шаблони. Вони мають містити інформацію про код помилки для її ідентифікації, описувати саму помилку зрозумілим текстом. Важливо вказати причини помилки та які дії потрібні для її усунення. Знайдене штучним інтелектом автоматичне рішення про виправлення помилки напряму залежить від чіткого опису причин та рішень в технічній документації.

Обов’язково треба приділити увагу написанню бази знань для користувачів програмного продукту та зробити її доступною для доступу через API.

Для генерації технічної документації з коду успішно використовується SQL Phrase Index, Doxygen, MkDocs та інші системи. Програмами для створення описів API для зчитування є Swagger, Open API, які можна використовувати для аналізу за допомогою штучного інтелекту.

У процесі підготовки, тестування та використання програмного забезпечення його технічна документація повинна регулярно оновлюватись та доповнюватись актуальними даними.

Отже, якісно підготовлена і структурована технічна документація дуже важлива для успішного створення життєздатної та конкурентоспроможної технічної системи. Чим структурованою та стандартизованою буде документація, тим ефективніше буде аналіз та діагностика системи штучним інтелектом.

2 ЗАСТОСУВАННЯ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ В ДІАГНОСТИЦІ ТЕХНІЧНИХ СИСТЕМ

2.1 Використання експертних систем в діагностиці

Експертна система є інтелектуальною системою, що використовує знання та логіку експертів певних областей накопичену в спеціальних базах знань. Такі системи містять організовані експертні знання в певній області. Центром будь-якої експертної системи є база знань, в якій знання можуть бути представлені у вигляді правила «якщо (умова) – то (висновок)», логічних тверджень або семантичних мереж. Структура бази знань має певну ієрархію створення понять, які ідуть від загальних понять, до специфічних. Знання в базі розподіляються за чітко визначеними категоріями.

Застосування експертних систем в діагностиці мають ряд переваг. Вони мають обґрунтованість та доволі легку доказовість правильності прийнятих рішень. Експерті системи роблять знання висококваліфікованих фахівців доступними для широкого кола користувачів, а при постійному оновленні результатами нових досліджень і досвіду ще й підвищується їх ефективність з часом та високу довіру користувачів.

Експертні системи можуть бути застосовані для вирішення широкого ряду задач. Їх використовують для інтерпретації даних при багаторівневому аналізі, щоб їх узгодити, для діагностики виявлення неполадок в системі, для діагностики, проектування, планування та навчання.

Щодо сфер застосування, то експертні системи успішно застосовують в медицині для діагностики захворювань, обробки результатів лабораторних досліджень та призначення лікування, в автомобілях проводять діагностику помилок роботи, на виробництві виконують моніторинг технологічних процесів та стану обладнання, для виявлення проблем у комп’ютерних мережах та програмному забезпеченні.

Недоліками експертних систем можна назвати великі затрати часу та зусиль для внесення експертних даних до бази знань, постійне необхідність вносити оновлення, щоб база була актуальною. Також на відміну від сучасного машинного навчання експертні системи не мають здатності до самооновлення, тому не можуть бути повністю автоматизованими. Вони не можуть створювати нові ідеї або приймати рішення в нестандартних ситуаціях, як це може зробити кваліфікована людина. Але незважаючи на певні обмеження, експертні системи є потужним інструментом у діагностиці, який може значно покращити якість та ефективність процесу прийняття рішень у різних галузях. З розвитком технологій штучного інтелекту, зокрема машинного навчання, очікується подальший розвиток та розширення сфери застосування експертних систем у діагностиці.

2.2 Машинне навчання для класифікації та прогнозування помилок у програмному коді

Сучасний процес розробки програмного забезпечення характеризується високою складністю, що зумовлює закономірне виникнення різноманітних помилок програмного коду. Тому актуальними стають питання виявлення, класифікації та прогнозування потенційних недоліків програмного забезпечення на різних етапах його розробки та експлуатації. Особливої уваги заслуговують методи машинного навчання, які демонструють високу ефективність при вирішенні даних завдань.

Технології машинного навчання для класифікації та прогнозування помилок у програмному коді базуються на статистичному аналізі великого обсягу даних про попередні недоліки програмного забезпечення. Аналіз здійснюється з метою виявлення закономірностей, які надалі використовуються для розпізнавання потенційних проблем у програмних проектах. Застосування таких технологій дозволяє суттєво підвищити якість програмного продукту шляхом раннього виявлення та виправлення помилок.

Помилки програмного коду можна розділити на декілька груп залежно від методології їх функціонування. Перша група включає методи, засновані на контрольному навчанні, такі як дерева рішень, метод випадкових лісів, методи опорних векторів та нейронні мережі. Друга група охоплює алгоритми кластеризації, які використовують точки даних для класифікації. Ця група знаходить приховані шаблони у даних про помилки. До третьої групи відносять методи зниження розмірності, які дозволяють виділити найбільш інформативні ознаки помилок.

Варто відзначити, що ефективність застосування методів машинного навчання значною мірою залежить від якості та обсягу доступних даних для навчання моделей. У галузі розробки програмного забезпечення важливим джерелом таких даних є бази даних помилок, системи контролю версій та журнали змін. Аналіз даних із цих джерел дозволяє встановити зв’язки між помилками та певними патернами написання коду, які можуть служити індикаторами потенційних проблем у майбутньому.

Ефективність методів машинного навчання для прогнозування помилок у програмному коді визначається декількома факторами. По-перше, це здатність моделі визначити наявність дефектів в коді. По-друге, здатність моделі до коректного визначення коду без дефектів. І по-третє, це збалансованість цих показників, яка полягає в балансі між точністю та повнотою.

Особливої уваги заслуговує проблема дисбалансу класів при навчанні моделей класифікації помилок. Зазвичай у програмному коді кількість коректних фрагментів значно перевищує кількість фрагментів із помилками, що може призвести до зміщення моделі у бік класифікації всіх фрагментів як безпомилкових. Для вирішення цієї проблеми застосовують різні методи балансування класів, таких як SMOTE (Synthetic Minority Oversampling Technigue), RandomUnderSampler та інші.

Поширеним підходом до прогнозування помилок у коді є аналіз метрик коду. Метрики коду включають такі показники, як цикломатична складність, кількість рядків коду, коефіцієнт згуртованості та інші. Застосування методів

машинного навчання для аналізу зв’язку між метриками коду та наявністю помилок дозволяє виявляти потенційно проблемні ділянки коду на ранніх етапах розробки програмного забезпечення.

Ще одним напрямком застосування машинного навчання для класифікації помилок є аналіз патернів зміни коду. Цей підхід базується на припущенні, що певні патерни зміни коду можуть бути індикаторами підвищеного ризику помилок. До таких патернів можна віднести часті модифікації одного й того ж файлу, одночасні зміни в декількох файлах або зміни, що вносяться в нетипові години та інша інформація, що має постійне повторення. Застосування алгоритмів асоціативних правил та послідовного аналізу до даних із системи контролю версій, дозволяє виявляти такі патерни та використовувати їх для прогнозування потенційних проблем.

Окремої уваги заслуговує підхід до прогнозування помилок, заснований на аналізі історії змін програмного коду. Даний підхід базується на припущенні, що модулі програмного забезпечення, які часто змінюються, мають вищу ймовірність виникнення помилок. Застосування методів часових рядів та рекурентних нейронних мереж для аналізу історії змін дозволяє виявляти динаміку за певний проміжок часу в інтенсивності виникнення помилок, що є цінною інформацією для планування процесу тестування програмного забезпечення.

Ще одним напрямком розвитку методів машинного навчання для прогнозування помилок та класифікації є метод ансамблів. Цей метод полягає в комбінуванні прогнозів декількох моделей для отримання точнішого результату. Дослідження показують, що ансамблі, які поєднують прогнози різних моделей, таких як дерева рішень, нейронні мережі та методи опорних векторів, демонструють вищу точність у порівнянні з окремими моделями.

Слід зауважити, що поряд із перевагами застосування методів машинного навчання для прогнозування та класифікації помилок програмного коду існують і певні обмеження. Зокрема, це складність тлумачення результатів моделей нейронних мереж та методу ансамблів. Окрім того, існує проблема інтеграції

моделей в різні проекти та організації, оскільки кожний проект має свої особливості розробки та типи можливих помилок.

Підсумовуючи, можна стверджувати, що застосування машинного навчання для опрацювання помилок в коді є перспективним напрямком підвищення якості програмного забезпечення. Подальший розвиток цього напрямку пов’язаний із розробкою ефективніших методів виявлення ознак, підвищенням тлумачення моделей та їх адаптацією до особливостей конкретних проектів розробки програмного забезпечення.

2.3 Нейронні мережі та глибоке навчання в процесах автоматизації діагностики програм

Розвиток технологій штучного інтелекту, зокрема нейронних мереж та методів глибокого навчання, відкриває нові можливості для автоматизованої діагностики програмного забезпечення. Ці технології допомагають зменшити ризики людської помилки та значно покращити функціонування програмних систем.

Основними компонентами нейронних мереж є штучні нейрони, які організовані в шари та з’єднані між собою певними зв’язками. Навчання нейронної мережі полягає в підборі вагомих коефіцієнтів таким чином, щоб мережа виконувала коректне вирішення задач.

Глибоке навчання є підходом у машинному навчанні, що ґрунтується на використанні глибоких нейронних мереж. Ці мережі мають багато прихованих шарів між вхідним і вихідним шарами. Це дозволяє моделям автоматично виділяти ієрархічні ознаки з даних, починаючи від простих деталей на нижніх рівнях і завершуючи складними абстракціями на вищих. Така здатність робить глибокі нейронні мережі особливо ефективними для аналізу складних структурованих даних, таких як програмний код.

У контексті автоматизованої діагностики програмного забезпечення нейронні мережі та методи глибокого навчання застосовуються у вирішенні

різноманітних завдань. Одним із головних завдань є виявлення аномалій у поведінці програмного забезпечення. Аномалії можуть бути індикаторами наявності помилок у коді, вразливостей безпеки або інших проблем, які потребують уваги розробників. Нейронні мережі, зокрема автокодувальники та рекурентні нейронні мережі демонструють високу ефективність у виявленні аномалій шляхом навчання на зразках нормальної поведінки програми та подальшого виявлення відхилень від цієї норми.

Згідно з дослідженнями згорткові нейронні мережі виявляють високу ефективність при аналізі статичного коду програми. Ці мережі розглядають код як послідовність символів і здатні виявляти локальні патерни, які можуть бути індикаторами помилок. Однак застосування згорткових нейронних мереж для аналізу коду має свої особливості, пов’язані з необхідністю врахування синтаксичної структури програмного коду, яка відрізняється від структури природних мов і зображень, для яких традиційно використовуються згорткові мережі.

Рекурентні нейронні мережі, особливо їхні модифікації LSTM та GRU ефективно застосовуються для аналізу послідовностей, що робить їх придатними для аналізу динамічної поведінки програмного забезпечення. Ці мережі здатні виявляти аномалії у послідовностях викликів функцій, доступу до пам’яті та інших аспектах виконання програми. Крім того, рекурентні нейронні мережі високоефективні при прогнозуванні поведінки програми, що дозволяє виявляти потенційні проблеми ще до того, як вони призведуть до збоїв.

Окремої уваги заслуговують трансформерні моделі, які останнім часом демонструють вражаючі результати у вирішенні завдань, пов’язаних із обробкою послідовностей даних. Ці моделі засновані на механізмі уваги і застосовуються для аналізу програмного коду. Вони здатні враховувати зв’язки між елементами коду, навіть якщо вони знаходяться на значній відстані один від одного. Дослідження показують, що трансформерні мережі, такі як CodeBEST та GPT для коду, демонструють високу точність у виявленні помилок та пропозиції їх виправлення.

Одним з основних аспектів використання нейронних мереж та методів глибокого навчання в автоматизованій діагностиці програмного забезпечення є представлення програмного коду у форматі, придатному для обробки цими методами.

Існують різні підходи до такого представлення, включаючи представлення на рівні символів, токенів, абстрактних синтаксичних дерев (AST) та графів залежності програми (PDG). Кожен із цих підходів має свої переваги та недоліки, і вибір конкретного підходу залежить від особливостей вирішуваного завдання.

Особливістю застосування методів глибокого навчання для діагностики програмного забезпечення є необхідність наявності великих обсягів розмічених даних для навчання моделей. Це становить певну проблему, оскільки створення таких даних потребує значних зусиль експертів. Для вирішення цієї проблеми застосовуються методи навчання з підкріпленням, напівавтоматичного навчання та трансферного навчання, які дозволяють ефективно використовувати наявні дані та зменшити потребу в розмічених даних.

Однією з перспективних областей застосування нейронних мереж у діагностиці програмного забезпечення є виявлення вразливостей безпеки. Вразливості безпеки є особливим типом помилок програмного забезпечення, які можуть бути використані зловмисниками для несанкціонованого доступу до системи або порушення її роботи. Виявлення таких вразливостей є складним завданням, яке потребує глибокого розуміння як програмного коду, так і можливих способів його експлуатації. Нейронні мережі, особливо глибокі, демонструють високу ефективність у виявленні партернів коду, які можуть бути індикаторами наявності вразливостей.

У таблиці 2.1 наведено порівняльний аналіз різних архітектури нейронних мереж для вирішення завдань автоматизованої діагностики програмного забезпечення.

Архітектура	Основні задачі	Переваги	Недоліки	Обчислювальна складність
Згорткові нейронні мережі (CNN)	Аналіз статичного коду, виявлення локальних	Ефективність обробки локальних партернів, відносно	Обмеженість при аналізі довгострокових	Середня

	партернів	невисока обчислювальна складність	залежності у коді	
Рекурентні нейронні мережі (RNN/LSTM)	Аналіз послідовностей викликів, прогнозування поведінки	Здатність обробляти послідовності змінної довжини, врахування контексту	Високі обчислювальні вимоги, складність навчання	Висока
Трансформерні моделі	Аналіз коду з урахуванням далеких зв’язків, генерація виправлень	Паралельна обробка, висока точність, здатність враховувати далекі зв'язки	Надзвичайно високі вимоги до обчислювальних ресурсів	Дуже висока
Автокодувальники	Виявлення аномалій, зменшення розмірності	Навчання без учителя, ефективне стиснення даних	Складність інтерпретації результатів	Низька до середньої
Графові нейронні мережі (GNN)	Аналіз графів залежності програми, виявлення складних партернів	Природня обробка структурованих даних, врахування зв’язків між елементами коду	Складність представлення коду у вигляді графа	Висока

Таблиця 2.1 Порівняльний аналіз архітектури нейронних мереж для автоматизованої діагностики програмного забезпечення.

Важливим аспектом застосування нейронних мереж для діагностики програмного забезпечення є їхня здатність до інтерпретованих результатів. Часто нейронні мережі, особливо глибокі, розглядаються як "чорні скриньки", які видають результат без пояснення причин прийнятого рішення. Це може бути проблемою у контексті діагностики програмного забезпечення, оскільки розробникам важливо розуміти, чому певний фрагмент коду було визначено як проблемний. Для вирішення цієї проблеми розробляються методи інтерпретованого штучного інтелекту, які дозволяють отримувати пояснення рішень, прийнятих нейронними мережами.

Одним із напрямків застосування нейронних мереж у діагностиці програмного забезпечення є автоматичне виправлення помилок. Цей підхід ґрунтується на використанні генеративних моделей, таких як автокодувальники та генеративно-змагальні мережі (GAN), для генерації варіантів виправлення виявлених помилок. Хоча цей напрямок є відносно новим, він демонструє

перспективні результати, особливо для виправлення типових помилок, таких як перевірки порожнього поля, невірно оброблені виключення тощо.

Ще одним важливим аспектом застосування нейронних мереж у діагностиці програмного забезпечення є їхня здатність до адаптації та навчання на нових даних. Це особливо важливо у контексті розвитку програмного забезпечення, оскільки з часом з'являються нові технології, патерни програмування та типи помилок. Нейронні мережі здатні адаптуватися до таких змін і підтримувати свою ефективність в майбутньому.

Не можна не відзначити роль ансамблів нейронних мереж у підвищенні точності діагностики програмного забезпечення. Ансамблі, які поєднують результати декількох моделей, демонструють вищу стійкість до помилок та точність прогнозування у порівнянні з окремими моделями. Це особливо важливо в контексті діагностики програмного забезпечення, де хибні спрацьовування можуть призводити до значних витрат ресурсів на аналіз та виправлення неіснуючих проблем.

Підсумовуючи, можна стверджувати, що нейронні мережі та методи глибокого навчання є потужним інструментом для автоматизованої діагностики програмного забезпечення. Вони здатні ефективно вирішувати завдання виявлення помилок, прогнозування потенційних проблем та навіть пропонування варіантів виправлення. Подальший розвиток цього напрямку пов'язаний із розробкою більш ефективних методів представлення програмного коду, підвищенням тлумачення результатів та зменшенням потреби в розмічених даних для навчання моделей.

2.4 Методи обробки природної мови для аналізу звітів про помилки та технічної документації

Обробка природної мови є важливим напрямком штучного інтелекту, який фокусується на взаємодії між комп'ютерами та людською (природною) мовою. У контексті діагностики програмного забезпечення методи нейролінгвістичного

програмування відіграють критичну роль у аналізі текстової інформації, пов'язаної з програмним забезпеченням, зокрема повідомлень про помилки, описів проблем у системах відстеження помилок, технічної документації та коментарів у коді. Аналіз цієї інформації дозволяє отримати цінні відомості про характер проблем, їх причини та можливі шляхи вирішення.

Основними джерелами текстової інформації для аналізу в процесі діагностики програмного забезпечення є системи відстеження помилок, такі як JIRA, Bugzilla, GitHub Issues та інші. Ці системи містять описи проблем, звіти про помилки, коментарі розробників та користувачів, а також інформацію про статус та пріоритет проблем. Аналіз цієї інформації дозволяє виявляти тренди та патерни у виникненні проблем, класифікувати проблеми за типами та складністю, а також прогнозувати час, необхідний для їх вирішення.

Іншим важливим джерелом інформації є технічна документація програмного забезпечення, яка включає специфікації вимог, проектну документацію, посібники користувача та розробника, а також API-документацію. Аналіз документації дозволяє виявляти невідповідності між специфікаціями та реалізацією, неоднозначності у вимогах, а також прогалини у документації, які можуть бути джерелом проблем при розробці та використанні програмного забезпечення. Процес обробки природної мови для аналізу текстової інформації про програмне забезпечення зазвичай включає декілька етапів. Першим етапом є попередня обробка тексту, яка включає такі операції, як токенізація (розбиття тексту на слова або токени), видалення стоп-слів (слів, які не несуть смислового навантаження), стемінг (приведення слів до основи) та лематизація (приведення слів до словникової форми). Ці операції дозволяють зменшити розмірність даних та підвищити ефективність подальшого аналізу.

Другим етапом є вилучення ознак із тексту. Традиційними підходами до вилучення ознак є моделі на основі мішка слів (Bag of words) та TF-IDF. Однак, ці підходи мають суттєві обмеження, оскільки не враховують семантичні зв'язки між словами та контекст їх використання. Сучасні підходи ґрунтуються на

використанні векторних представлень слів, таких як Word2Vec, GloVe та FastText, які здатні захоплювати семантичні зв'язки між словами.

Особливого значення у аналізі текстової інформації про програмне забезпечення набули нейронні мережі, зокрема рекурентні нейронні мережі, згорткові нейронні мережі та Трансформерні моделі. Ці моделі демонструють високу ефективність у вирішенні завдань класифікації тексту, вилучення інформації, автоматичного реферування та інших завдань нейролінгвістичного програмування, які є актуальними в контексті аналізу повідомлень про помилки та документації.

Одним із найбільш перспективних напрямків застосування методів NLP у діагностиці програмного забезпечення є автоматична класифікація звітів про помилки. Класифікація може здійснюватися за різними критеріями, такими як тип помилки (функціональна, продуктивності, безпеки), компонент програмного забезпечення, який містить помилку, складність помилки та пріоритет її усунення. Автоматична класифікація дозволяє прискорити процес оброблення звітів про помилки, зменшити навантаження на розробників та забезпечити більш ефективний розподіл ресурсів.

Іншим важливим напрямком є автоматичне визначення дублікатів звітів про помилки. У великих програмних проектах часто виникає ситуація, коли різні користувачі повідомляють про одну і ту ж проблему різними словами. Для вирішення цього завдання застосовуються методи обчислення семантичної схожості текстів, які ґрунтуються на векторних представленнях та нейронних мереж. Виявлення дублікатів дозволяє уникнути дублювання зусиль на вирішення однієї і тієї ж проблеми та отримати більш повну інформацію про проблему шляхом об'єднання інформації з різних звітів.

Аналіз тональності повідомлень про помилки та коментарів користувачів дозволяє оцінити ступінь невдоволення користувачів певними аспектами програмного забезпечення та визначити найбільш критичні проблеми, які потребують негайного вирішення. Крім того, аналіз тональності може бути

корисним для оцінки сприйняття користувачами нових функцій та змін у програмному забезпеченні.

Важливим застосуванням методів NLP у діагностиці програмного забезпечення є автоматичне вилучення інформації про кроки відтворення помилок із звітів користувачів. Ця інформація є критичною для розробників, оскільки дозволяє їм відтворити проблему та зрозуміти її причини. Однак, користувачі часто надають цю інформацію у неструктурованому вигляді, що ускладнює її аналіз. Методи вилучення інформації на основі NLP дозволяють автоматично виділяти послідовність дій, які призводять до виникнення помилки, з текстового опису проблеми.

У таблиці 2.2 наведено порівняльний аналіз ефективності різних методів нейролінгвістичного програмування для аналізу повідомлень про помилки та документації програмного забезпечення.

Метод	Основні задачі	Переваги	Недоліки
TF-IDF + SVM	Класифікація повідомлень про помилки	Простота, швидкість	Ігнорування семантики
Word2Vec + CNN	Класифікація, визначення дублікатів	Врахування локального контексту	Обмеженість при аналізі далеких зв'язків
BERT	Класифікація, вилучення інформації	Висока точність, контекстні представлення	Висока обчислювальна складність
GPT	Генерація описів помилок, підсумків	Високоякісна генерація тексту	Надзвичайно висока обчислювальна складність
BiLSTM + Attention	Вилучення кроків відтворення	Ефективна обробка послідовностей	Складність навчання
FastText	Класифікація, аналіз тональності	Ефективність для рідкісних слів	Обмеженість при аналізі контексту

Таблиця 2.2 Порівняльний аналіз методології обробки природної мови для аналізу повідомлень про помилки.

Окремої уваги заслуговує застосування методів NLP для аналізу відповідності між документацією та реалізацією програмного забезпечення. Невідповідність між документацією та реалізацією є поширеною причиною помилок та проблем при використанні програмного забезпечення. Методи NLP дозволяють автоматично виявляти такі невідповідності шляхом порівняння тексту документації з кодом програми. Для цього застосовуються методи вилучення

інформації з тексту документації та статичного аналізу коду, які дозволяють встановити відповідність між описаними в документації вимогами та їх реалізацією у коді.

Суттєвим викликом при застосуванні методів NLP для аналізу повідомлень про помилки та документації є обробка специфічної термінології, характерної для галузі програмного забезпечення. Ця термінологія включає технічні терміни, назви технологій, фреймворків, бібліотек та інші специфічні для програмування поняття. Для ефективної обробки такої термінології застосовуються спеціалізовані словники, онтології та моделі векторних представлень, навчені на корпусах текстів, пов'язаних із програмним забезпеченням.

Важливим аспектом застосування методів NLP у діагностиці програмного забезпечення є їхня здатність до обробки текстів різними мовами. Програмне забезпечення часто розробляється міжнародними командами та використовується користувачами з різних країн, які можуть надавати звіти про помилки різними мовами. Для вирішення цього завдання застосовуються методи багатомовної обробки тексту, які дозволяють аналізувати тексти незалежно від мови, якою вони написані.

Цікавим напрямком застосування методів нейролінгвістичного програмування у діагностиці програмного забезпечення є автоматичне генерування описів помилок та рекомендацій щодо їх усунення. Цей підхід ґрунтується на використанні генеративних моделей мови, таких як GPT (Generative Pre-trained Transformer), які здатні генерувати зв'язний та інформативний текст на основі вхідних даних. Застосування таких моделей дозволяє автоматично створювати детальні описи виявлених проблем, їх можливих причин та рекомендацій щодо їх усунення, що суттєво полегшує роботу розробників.

Актуальною проблемою при застосуванні методів нейролінгвістичного програмування для аналізу повідомлень про помилки є обробка неповної та неточної інформації. Користувачі програмного забезпечення часто надають неповні або неточні описи проблем, які вони зустрічають, що ускладнює їх аналіз

та вирішення. Методи NLP, зокрема на основі нейронних мереж, здатні працювати з неповною та шумною інформацією, вилучаючи корисні відомості навіть із неякісних описів проблем.

Одним із перспективних напрямків застосування методів нейролінгвістичного програмування у діагностиці програмного забезпечення є аналіз зв’язків між різними джерелами інформації, такими як звіти про помилки, документація, код програми та історія змін. Інтеграція інформації з різних джерел дозволяє отримати більш повне розуміння проблем та їх причин, що сприяє більш ефективному їх вирішенню. Для цього застосовуються методи кросмодального аналізу та графові нейронні мережі, які здатні обробляти різнорідну інформацію та виявляти зв'язки між різними типами даних.

Варто зауважити, що ефективність застосування методів NLP у діагностиці програмного забезпечення значною мірою залежить від доступності якісних даних для навчання моделей. У галузі розробки програмного забезпечення існує певний дефіцит публічно доступних наборів даних для тренування моделей NLP, особливо для специфічних задач, таких як аналіз повідомлень про помилки. Для вирішення цієї проблеми застосовуються методи трансферного навчання, які дозволяють адаптувати моделі, попередньо навчені на загальних корпусах текстів, до специфічних завдань у галузі діагностики програмного забезпечення.

Слід відзначити роль тлумачення результатів при застосуванні методів NLP у діагностиці програмного забезпечення. Розробникам важливо розуміти, чому певне повідомлення про помилку було класифіковано певним чином або чому дві проблеми були визначені як дублікати. Для забезпечення вірної інтерпретації застосовуються методи пояснень на основі атрибуції ознак, візуалізації уваги та генерування текстових пояснень.

Підсумовуючи, можна стверджувати, що методи обробки природної мови відіграють ключову роль у автоматизації процесів аналізу текстової інформації, пов'язаної з програмним забезпеченням. Вони дозволяють ефективно обробляти великі обсяги неструктурованої текстової інформації, вилучаючи з неї цінні відомості для діагностики та вирішення проблем програмного забезпечення.

Подальший розвиток цього напрямку пов'язаний із розробкою більш ефективних методів обробки специфічної термінології, інтеграції інформації з різних джерел та забезпечення вірного тлумачення результатів.

Важливо зазначити, що штучний інтелект у діагностиці програмного забезпечення не замінює розробників, а є інструментом, який підвищує їхню продуктивність та ефективність. Автоматизація рутинних завдань, таких як класифікація повідомлень про помилки та виявлення дублікатів, дозволяє розробникам зосередитися на більш творчих та складних аспектах розробки програмного забезпечення.

3 РОЗРОБКА ТА ВПРОВАДЖЕННЯ СИСТЕМИ ДІАГНОСТИКИ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Архітектура інтелектуальної системи діагностики програмного забезпечення

Розробка архітектури інтелектуальної системи діагностики програмного забезпечення вимагає комплексного підходу, який враховує специфіку програмних систем, різноманітність методів аналізу та особливості застосування технологій штучного інтелекту. У даному підрозділі представлено концептуальну архітектуру такої системи, її основні компоненти та принципи взаємодії між ними.

Запропонована архітектура інтелектуальної системи діагностики програмного забезпечення ґрунтується на модульному принципі, який забезпечує гнучкість, масштабованість та можливість інтеграції з існуючими середовищами розробки та системами керування проектами.

Концептуальна архітектура інтелектуальної системи діагностики програмного забезпечення складається з шести основних модулів: збору даних, попередньої обробки інформації, аналізу коду, виявлення аномалій, класифікації помилок та формування рекомендацій та інтерфейсу користувача. Кожен із цих модулів виконує специфічні функції та взаємодіє з іншими компонентами системи через чітко визначені інтерфейси, що забезпечує можливість незалежного розвитку та вдосконалення окремих компонентів.

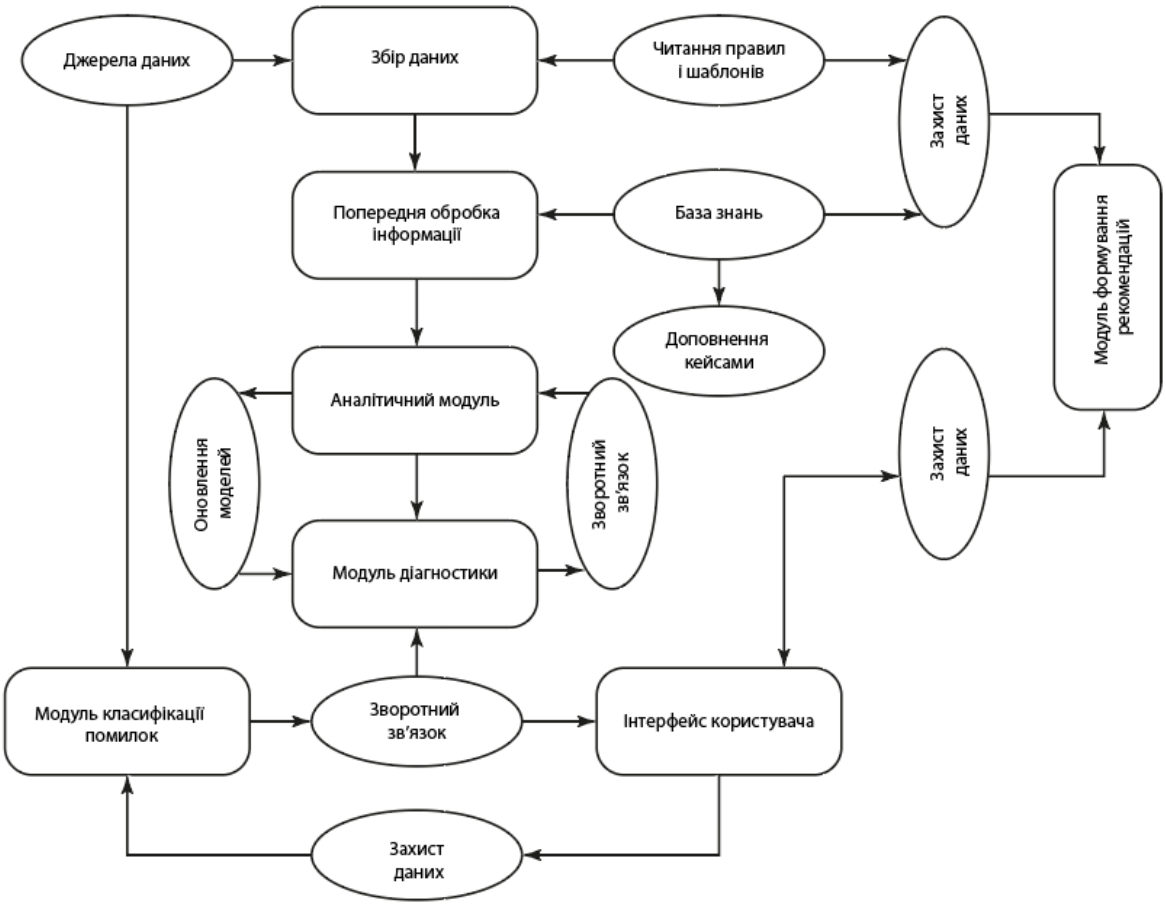

Рис. 3.1 Архітектура інтелектуальної системи діагностики програмного забезпечення.

Модуль збору даних забезпечує отримання вхідної інформації з різних джерел, включаючи вихідний код програмного забезпечення, історію змін, лог-журнали роботи програми, повідомлення про помилки та іншу релевантну інформацію. Цей модуль інтегрується з системами контролю версій (Git, SVN, Mercurial тощо), системами відстеження помилок (JIRA, Bugzilla, GitHub Issues тощо) та іншими інструментами розробки програмного забезпечення. Відповідно до досліджень, ефективність системи діагностики значною мірою залежить від повноти та якості вхідних даних, тому цей модуль є критично важливим для функціонування всієї системи. Адже недостовірні та неповні дані можуть призвести до хибних висновків.

Модуль попередньої обробки інформації відповідає за фільтрацію, нормалізацію та структурування даних, отриманих від модуля збору. Цей модуль виконує такі операції, як токенізація коду, видалення зайвої інформації та перетворення даних у формат, придатний для подальшого аналізу методами штучного інтелекту. Важливою функцією цього модуля є генерація ознак, які будуть використовуватися моделями машинного навчання для аналізу та класифікації помилок. Тому що якісно оброблені дані сприяють збільшенню точності моделей машинного навчання та експертних систем, що задіяні в наступних етапах обробки.

Аналітичний модуль поєднує методи статичного та динамічного аналізу для виявлення потенційних проблем у програмному коді. Статичний аналіз виконується без запуску програми та включає перевірку синтаксису, аналіз потоку даних та потоку керування, перевірку типів та інші методи аналізу вихідного коду. Динамічний аналіз, навпаки, передбачає виконання програми та моніторинг її поведінки, включаючи аналіз викликів функцій, використання пам'яті, продуктивності тощо. Цей модуль інтегрує традиційні методи аналізу коду з методами на основі штучного інтелекту, що дозволяє підвищити точність виявлення складних та нетривіальних помилок. А це важливо, так як автоматичний аналіз коду значно економить час на розробку і надає розробнику детальну інформацію про потенційні проблемні місця програмного забезпечення, а також рекомендації щодо їх покращення.

Модуль діагностики займається аналізом і класифікацією виявлених аномалій або помилок. Він проводить глибокий аналіз, встановлюючи причинно-наслідкові зв’язки, визначає критичність збоїв і формує гіпотези щодо джерел проблем. Тому на базі цього модуля часто генеруються рекомендації з виправлення аномалій або оптимізації програмного забезпечення. Це важливий компонент архітектури, оскільки він безпосередньо перетворює сирі дані та результати попереднього аналізу у конкретні висновки.

Модуль класифікації помилок відповідає за категоризацію виявлених проблем за типом, серйозністю, пріоритетом та іншими критеріями. Цей модуль

використовує методи супервізованого навчання, такі як дерева рішень, випадкові ліси, градієнтний бустинг та нейронні мережі, для класифікації помилок на основі їх характеристик. Важливою функцією цього модуля є також оцінка ризиків, пов'язаних з виявленими проблемами, та визначення їх пріоритетності для виправлення. Тому що правильна категоризація помилок сприяє накопиченню досвіду та поповненню бази знань, що дає можливість системі надавати точніші рекомендації.

Модуль формування рекомендацій генерує пропозиції щодо виправлення виявлених помилок та покращення якості програмного забезпечення. Цей модуль використовує генеративні моделі інтелектуальних систем для створення конкретних пропозицій щодо змін у коді, а також методи пояснюваного штучного інтелекту для надання обґрунтувань своїх рекомендацій. Цей модуль сприяє оптимізації процесу виправлення, що забезпечує стабільність та безпеку програмного забезпечення.

І нарешті, інтерфейс користувача містить досвід користувача та надає зворотній зв’язок, яким підтверджує або спростовує корисність отриманих рекомендацій. Користувач отримує корисну інформацію та підказки про усунення помилок. Цей модуль є ключовим для забезпечення прозорості, зручності та ефективності роботи всієї системи.

Важливим аспектом архітектури запропонованої системи є використання методів ансамблювання для підвищення точності та надійності прогнозів. Відповідно до досліджень, комбінування результатів різних моделей та методів аналізу дозволяє компенсувати недоліки окремих підходів та підвищити загальну ефективність системи. У запропонованій архітектурі ансамблювання застосовується на рівні модулів діагностики та класифікації помилок, а також при формуванні остаточних рекомендацій.

Особливістю запропонованої архітектури є також система зворотного зв'язку, яка дозволяє враховувати реакцію розробників на рекомендації системи та використовувати цю інформацію для поліпшення моделей машинного навчання. Ця система реалізує принцип навчання з підкріпленням, при якому модель

постійно вдосконалюється на основі інформації про успішність або неуспішність своїх прогнозів.

Важливим компонентом архітектури є також підсистема візуалізації результатів, яка представляє інформацію про виявлені проблеми та рекомендації в зрозумілому та інтуїтивно зрозумілому форматі. Ця підсистема використовує методи інформаційної візуалізації для представлення складної залежності та патернів у формі, яка полегшує їх сприйняття та інтерпретацію розробниками. Згідно з дослідженнями, ефективна візуалізація результатів аналізу є критично важливою для прийняття цих результатів розробниками та їх інтеграції в процес розробки програмного забезпечення.

Таким чином, запропонована архітектура інтелектуальної системи діагностики програмного забезпечення поєднує традиційні методи аналізу коду з сучасними технологіями штучного інтелекту, забезпечуючи високу точність виявлення та класифікації помилок, а також формування ефективних рекомендацій щодо їх усунення. Модульний принцип побудови системи забезпечує її гнучкість, масштабованість та можливість інтеграції з існуючими середовищами розробки програмного забезпечення.

3.2 Алгоритми та методи підвищення ефективності виявлення помилок за допомогою штучного інтелекту

Підвищення ефективності виявлення помилок у програмному забезпеченні є ключовим завданням у галузі розробки програмних систем. Інтеграція методів штучного інтелекту відкриває нові можливості для вирішення цього завдання, дозволяючи автоматизувати складні аналітичні процеси та виявляти патерни, які важко ідентифікувати традиційними методами. У цьому підрозділі розглянуто алгоритми та методи, які дозволяють підвищити ефективність виявлення помилок у програмному забезпеченні з використанням технологій штучного інтелекту.

Одним із ключових методів підвищення ефективності виявлення помилок є використання ансамблевих алгоритмів машинного навчання, які комбінують

результати декількох моделей для отримання більш точних прогнозів. Ансамблеві методи, такі як випадкові ліси (Random Forests) та градієнтний бустинг (Gradient Boosting), демонструють вищу точність у задачах класифікації та прогнозування помилок у порівнянні з окремими моделями. Особливістю ансамблевих методів є їхня здатність компенсувати недоліки окремих моделей та зменшувати ймовірність перенавчання, що є критично важливим при розробці систем діагностики програмного забезпечення.

Для ефективного виявлення помилок у програмному коді широко застосовуються методи глибокого навчання, зокрема згорткові нейронні мережі (CNN) та рекурентні нейронні мережі (RNN). Саме згорткові нейронні мережі ефективно виявляють локальні патерни в коді, які можуть бути індикаторами помилок. Рекурентні нейронні мережі, особливо їх модифікації, такі як LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit), здатні аналізувати послідовності операцій та виявляти довгострокові залежності в коді. Комбінування цих методів дозволяє врахувати як локальні, так і глобальні характеристики коду, підвищуючи точність виявлення помилок.

Трансформерні моделі, такі як BERT (Bidirectional Encoder Representations from Transformers) та GPT (Generative Pre-trained Transformer), адаптовані для аналізу програмного коду, демонструють вражаючі результати у виявленні складних патернів та помилок. Модель CodeBERT, яка є адаптацією BERT для програмного коду, показує високу точність у задачах аналізу коду, включаючи виявлення вразливостей та семантичних помилок. Ці моделі використовують механізм уваги (attention mechanism), який дозволяє їм ефективно обробляти довгі послідовності та врахувати контекстуальні зв'язки між різними частинами коду.

Важливим напрямком підвищення ефективності виявлення помилок є застосування методів навчання з підкріпленням (reinforcement learning) для оптимізації стратегій пошуку помилок. Алгоритми навчання з підкріпленням, такі як Q-learning та Policy Gradient, можуть бути використані для визначення оптимальної послідовності операцій аналізу, які максимізують ймовірність виявлення помилок при мінімальних витратах обчислювальних ресурсів. Цей

підхід дозволяє системі адаптуватися до специфіки конкретного проекту та зосередитися на аналізі найбільш критичних і схильних до помилок частин коду.

Для виявлення рідкісних та нетипових помилок ефективними є методи виявлення аномалій, які базуються на припущенні, що помилки представляють собою відхилення від нормальної поведінки програми або структури коду. Автокодувальники (autoencoders) та ізоляційні ліси (isolation forests) демонструють високу ефективність у виявленні нетипових патернів у коді та поведінці програмного забезпечення. Ці методи особливо цінні для виявлення нових типів помилок, які не були представлені в навчальних даних.

Методи обробки природної мови (NLP) відіграють важливу роль у аналізі текстової інформації, пов'язаної з програмним забезпеченням, такої як коментарі в коді, повідомлення про помилки та документація. Застосування методів векторування тексту, таких як Word2Vec та FastText, у поєднанні з методами класифікації, дозволяє ефективно аналізувати текстову інформацію та виявляти потенційні проблеми на основі її семантичного змісту. Цей підхід особливо корисний для аналізу повідомлень про помилки та виявлення зв’язків між різними проблемами.

Важливим аспектом підвищення ефективності виявлення помилок є використання методів активного навчання (active learning) для оптимізації процесу збору та розмітки даних для навчання моделей. Активне навчання дозволяє зменшити кількість необхідних розмічених прикладів, зосередившись на найбільш інформативних випадках. Цей підхід є особливо цінним у контексті діагностики програмного забезпечення, де розмітка даних потребує значних зусиль експертів.

Для підвищення інтерпретації результатів аналізу застосовуються методи пояснюваного штучного інтелекту (Explainable AI). Такі методи, як LIME (Local Interpretable Model-agnostic Explanations) та SHAP (SHapley Additive exPlanations), дозволяють визначити фактори, які найбільше впливають на прийняття рішення моделлю, і представити цю інформацію в зрозумілій для розробників формі. Це

сприяє більш ефективному використанню результатів аналізу та підвищує довіру до системи діагностики.

Графові нейронні мережі (Graph Neural Networks, GNN) є ефективним інструментом для аналізу структурованих даних, таких як графи залежності програми (Program Dependency Graphs, PDG) та абстрактні синтаксичні дерева (Abstract Syntax Trees, AST). Ці мережі здатні виявляти складні патерни залежності між різними частинами коду, які можуть бути індикаторами потенційних проблем. Особливо ефективними є моделі GNN для виявлення помилок, пов'язаних з неправильним використанням API та порушеннями їх залежності між компонентами програми.

Для ефективного використання обмежених обчислювальних ресурсів застосовуються методи розрідженого навчання (sparse learning) та квантизації моделей. Такі методи дозволяють зменшити розмір моделей та прискорити їх роботу без значної втрати точності, що є важливим для інтеграції систем діагностики в процес безперервної інтеграції (CI/CD) та забезпечення швидкого зворотного зв'язку для розробників.

Важливим компонентом системи діагностики є механізм ранжування виявлених проблем за їх критичністю та пріоритетом. Для цього ефективно використовуються методи багатокритеріального навчання (multi-objective learning), які враховують різні аспекти помилок, такі як їх вплив на функціональність, безпеку, продуктивність та зручність використання програмного забезпечення. Це дозволяє оптимізувати процес виправлення помилок та зосередитися на найбільш критичних проблемах.

Для ефективної роботи з неповними та шумними даними застосовуються методи робастного навчання (robust learning), які підвищують стійкість моделей до шуму та помилок у вхідних даних. Відповідно до досліджень Вони особливо важливі при аналізі даних моніторингу та журналів роботи програми, які можуть містити шум та артефакти, не пов'язані з реальними проблемами.

Алгоритми федеративного навчання (federated learning) дозволяють навчати моделі на розподілених даних без необхідності їх централізації, що вирішує

проблеми конфіденційності та безпеки. Саме ці алгоритми можуть бути ефективно використані для навчання моделей діагностики на даних з різних організацій та проектів, зберігаючи при цьому конфіденційність вихідного коду та бізнес-логіки.

Таким чином, поєднання різних методів та алгоритмів штучного інтелекту дозволяє значно підвищити ефективність виявлення помилок у програмному забезпеченні. Ансамблеві методи, глибоке навчання, трансформерні моделі, методи виявлення аномалій та інші підходи, описані в цьому підрозділі, формують комплексну методологію для розробки ефективних систем діагностики програмного забезпечення. Важливими аспектами цієї методології є також забезпечення інтерпретації результатів, оптимізація використання обчислювальних ресурсів та ефективна інтеграція з існуючими процесами розробки програмного забезпечення.

3.3 Створення помічника на базі ШІ за методом MLP Classifier

Дослідження засноване на власному досвіді роботи в провідній компанії, яка надає телекомунікаційні послуги зв’язку в Україні. Де мають місце нетипові запити клієнтів про проведення аналізу процесу ініціації вхідних та вихідних дзвінків. Сучасні віртуальні автоматичні телефоні станції побудовані на платформі Asterisk. При перегляді журналів логів платформи, завдання полягає у виявленні помилок в процесі реєстрації клієнта, перевірці коректності напису номеру телефону при ініціації вихідного дзвінка та підтвердження коректності використання телефонії клієнтом в особистому кабінеті користувача.

Для уникнення помилок в роботі, важливо встановити повну бібліотеку. При встановленні неповної бібліотеки модулів виникатиме помилка про те, що не вистачає певного модуля. Для коректної роботи написаної програми та уникнення блокування файлів формату «exe» Windows Defender, необхідно змінити налаштування в пункті захисту від вірусів та загроз. Після цього знаходимо файл

та прибраємо його з карантину, у вигляді виключення з правил. Після виконання налаштувань програма буде готова до тестування.

Для успішного написання програми діагностики помилок використано власний досвід, отриманий під час навчання в університеті та роботи в технічній підтримці провідної компанії телефонії в Україні. В процесі розробки використані форуми, журнали та канали про програмування. Головним завданням поставлена розробка програми, яка зможе читати log, аналізувати його та в реальному часі надавати відповідь на питання: «Що це за помилка? Скільки разів вона трапляється? Що означає визначена помилка?». Таким чином перед створенням самої програми було приділено час дослідженню log сховищ та зроблено висновок, що помилки у таких середовищах знайти буде складно, тому для проекту сконцентровано увагу на схемі з трьох класів помилок, а саме: WARN, ERROR, INFO. Застосовано метод машинного навчання моделі на базі MLP Classifier класифікації.

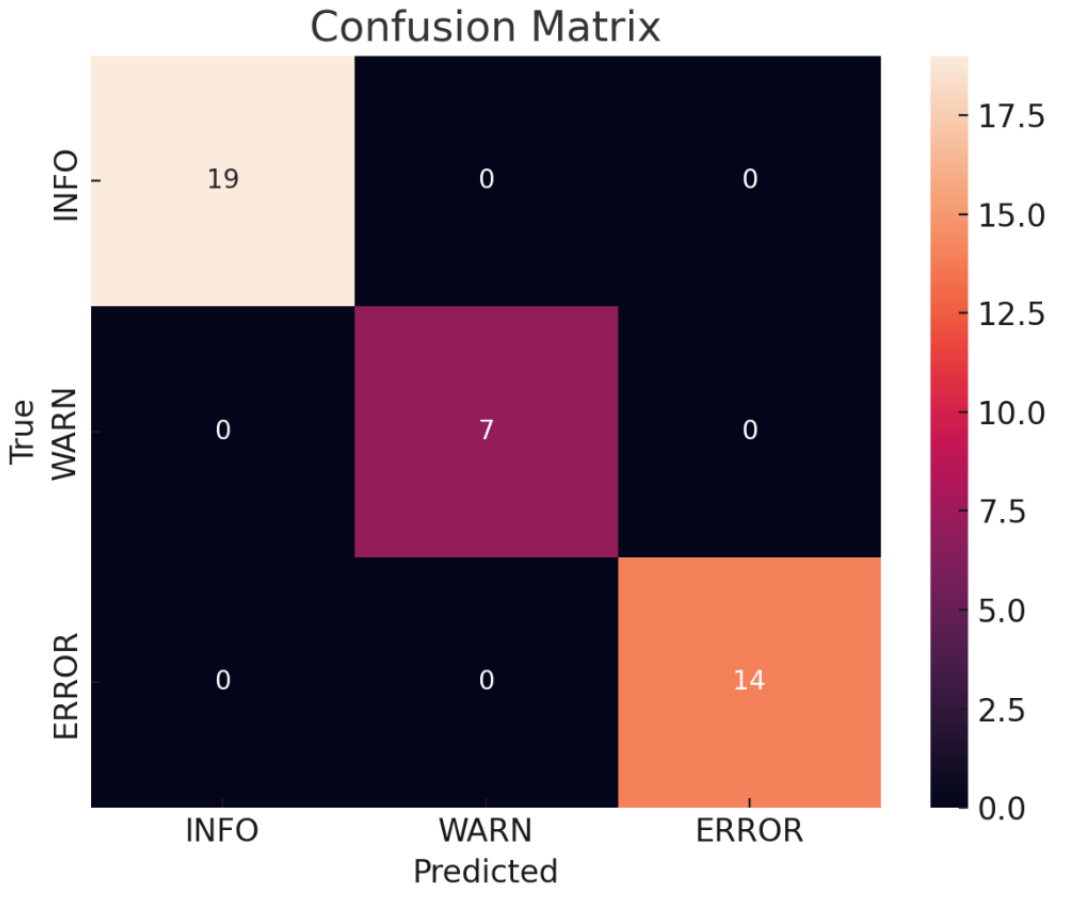

Рис. 3.2 Схема застосованих класів помилок.

В процесі навчання модель добре розрізняла три класи, тому з метою покращення об’єму, який може обробити штучний інтелект та якості створеної програми виконано розподіл у вигляді категорій. Категорії дали можливість додати більше помилок, попереджень та інформаційних повідомлень в текстовому файлі log. Створено повідомлення: Device already registered, Network request timeout, App not compatible with device, IP address changed to unknown location, Transaction error: insufficient funds, Invalid email format entered, Valid registration completed, Successful network request, Compatible device connected, Correct ID entered у Excel файл, що розміщений в одній папці з exe файлом. Це виконано для використання не справжнього масиву даних для машиного навчання, щоб уникнути використання приватних даних справжніх клієнтів.

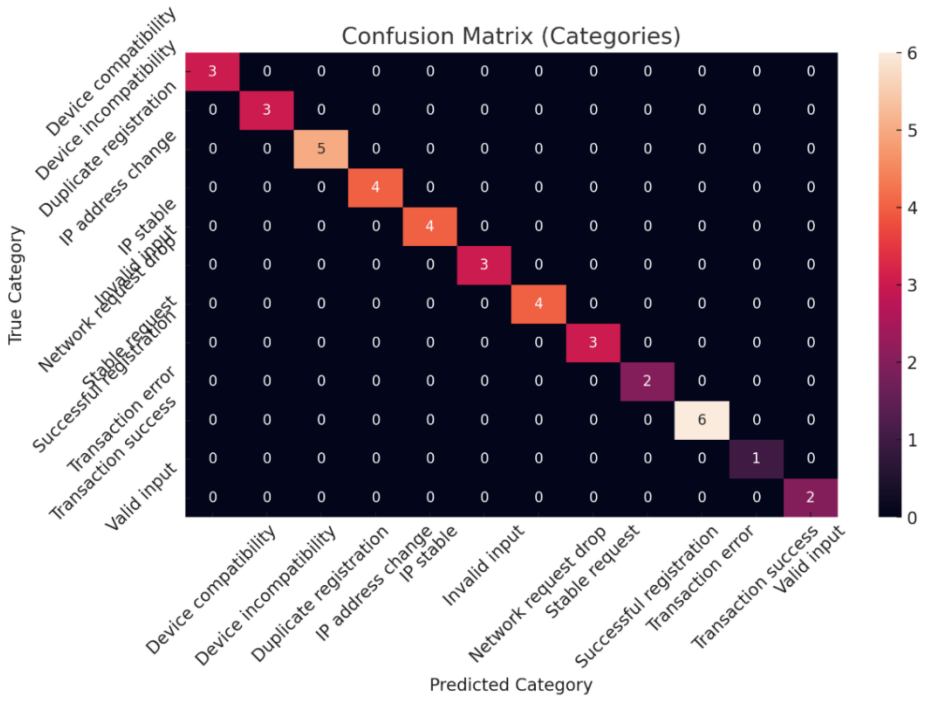

Рис. 3.3 Розширена схема застосованих класів помилок.

Після того як було створено на мові програмування Python вже сам exe файл, в даному випадку альфа версія інтерфейсу користувача, першим кроком проведено тестування системи і зчитування легких логів.

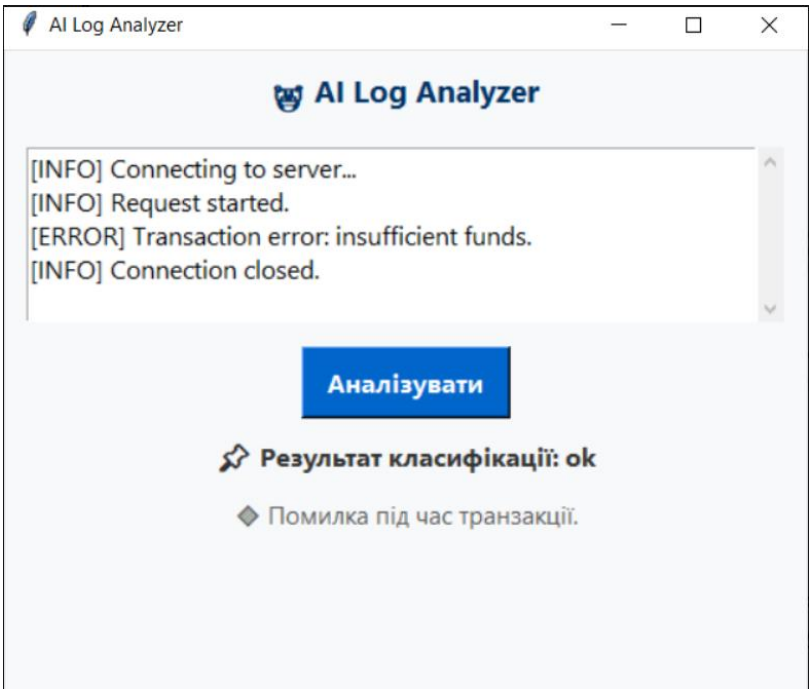

Рис. 3.4 Альфа версія застосунку Ai Log Analyzer.

Програма виконала свою задачу, побачила помилку під час транзакції, хоча і тестовий файл був невеликий, але такий тест дав зрозуміти, що сама програма має потенціал на більш складні запити та можливість постійно оновлювати як інтерфейс для користувача так і оновлювати сам масив даних який log машина зможе аналізувати. Технології мають схильність до розвитку, розвиток це нові можливості, можливості дають як і приємний досвід так і негативний, негативний досвід можна зменшити в свою чергу якщо бачити проблему та знати її вирішення, тому дана програма має потенціал допомагати як і звичайним користувачам сервісів так і технічним спеціалістам які займаються діагностикою складних систем. Бета версія програми вже мала більш зручний темний інтерфейс та кнопку очищення log файлу. Таким чином можна було опрацювати декілька запитів клієнта в одній розмові і надати одразу швидку відповідь чому не відбулася певна подія. Також у низу інтерфейсу більш відкрито стало видавати результат аналізу від мого штучного інтелекту. Та додав для зручності унизу зведення помилок для користувача програми також інформацію про те скільки разів повторилась певна помилка та у якому рядку чи рядках можна зустріти. В моєму випадку користувач по сценарію мав би бути співробітник технічної

підтримки певного сервісу. Не обов'язково телекомунікаційних послуг, мета зробити саме готовий фундамент для сервісів, які хотіли б виділити час на створення такого софт рішення лише додавши свої класи та категорії.

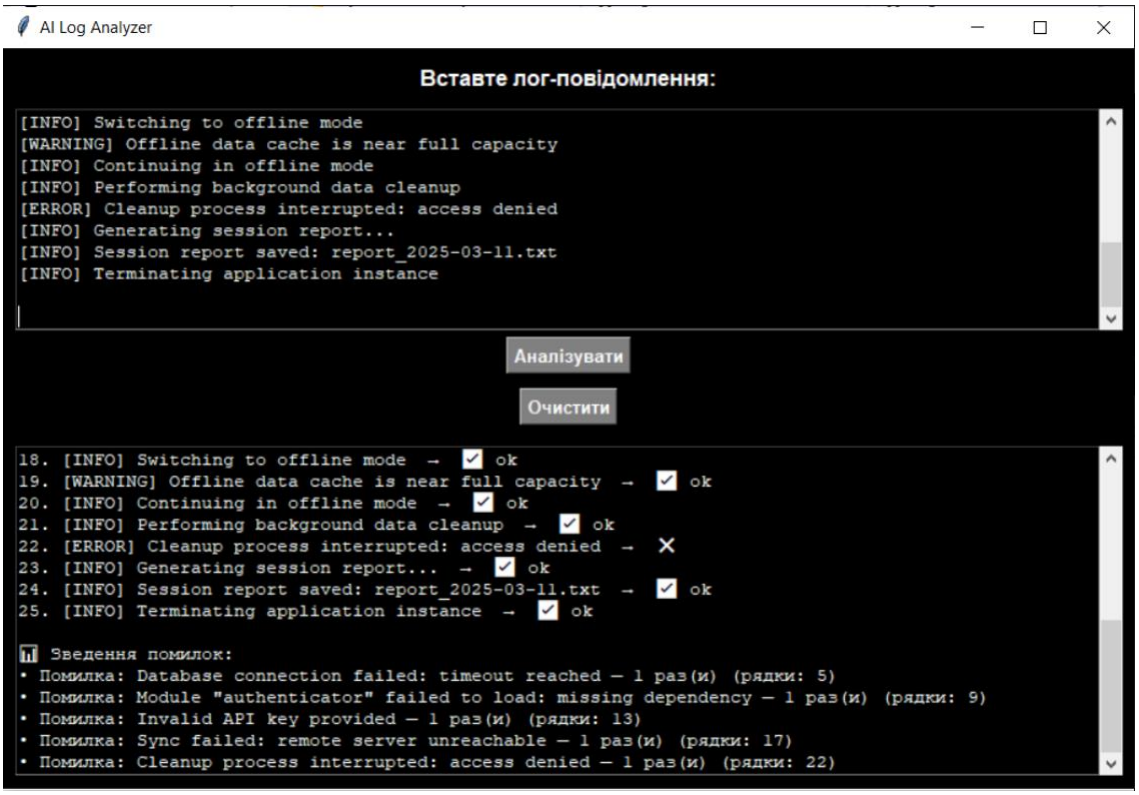

Рис. 3.5 Бета версія застосунку Ai Log Analyzer.

Якщо порівнювати альфа та бета версію програми то зміни стали більш помітними у позитивну сторону як для функціоналу програми, так і у плані зручності для користувача. Але ще зі сторони доопрацювань потрібно зробити можливість додавати логи в форматах csv або txt для можливості збереження файлу на пристрої та можливості також навпаки вивантажити log з локальної мережі або продовжити опрацювання вивантаженого раніше файлу завдяки кнопці, яка надасть можливість розмістити файл csv або txt у програму для аналізу. Також необхідно доопрацювати можливість бачити зведення помилок, у яких рядках і скільки разів повторюються помилки у вигляді спливаючого вікна після аналізу log файлу. Було розпочато розробку фінальної версії програми моделі на базі MLP Classifier. Подальші зміни викликали нові навантаження на програму і могли видавати певні помилки інтерфейсу або функціоналу, тому було вирішено зупинити дослідження на самому оптимальному варіанті інтерфейсу та

практичної частини власне програми Ai Log Analyzer. Фінальна версія представлена нижче:

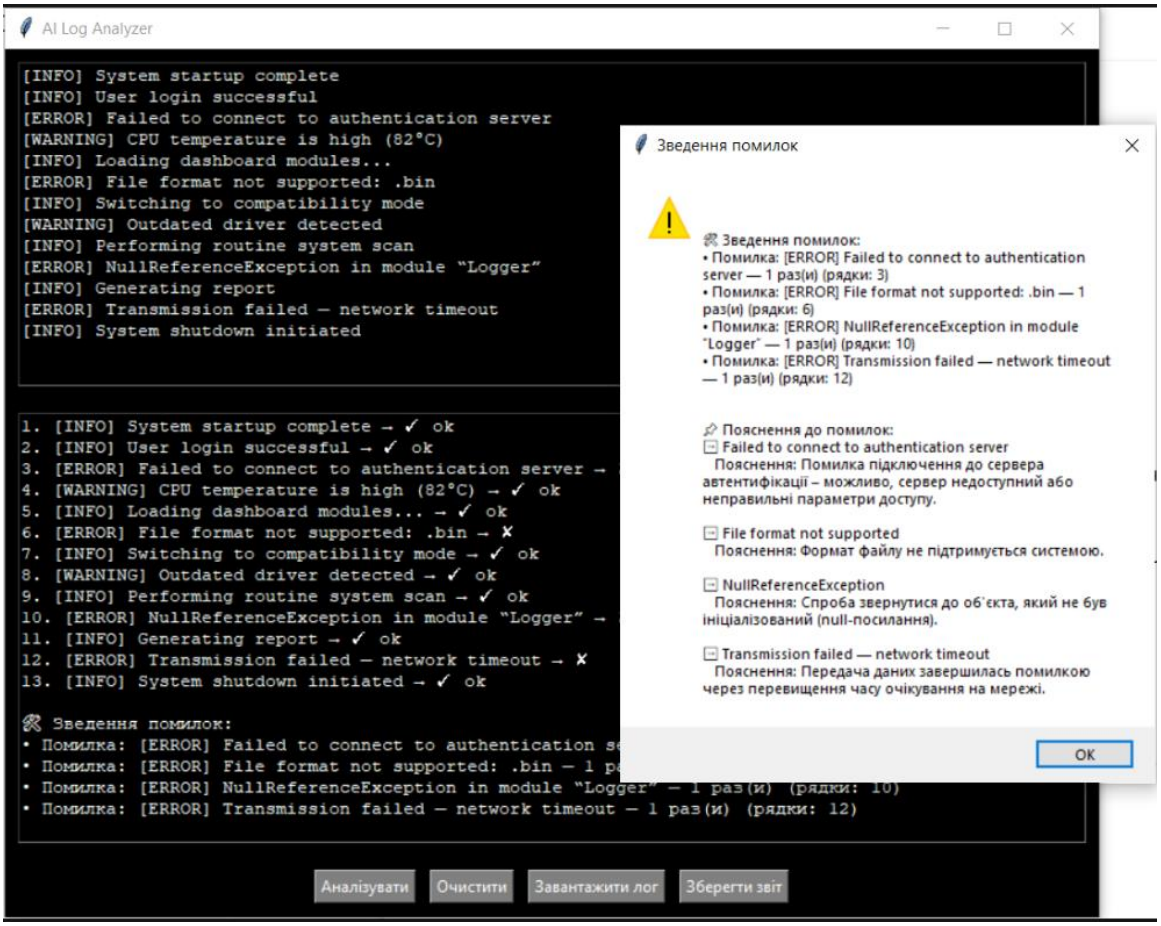

Рис. 3.6 Фінальна версія застосунку Ai Log Analyzer.

Тепер програма має можливість як і при альфа версії аналізувати log який ми вставили як текст, так і можливість працювати з завантаженим файлом у форматі csv або txt, які були отримані як і зовнішнім шляхом через глобальну або локальну мережу так і внутрішнім завдяки тому, що звіт міг бути збережений раніше для того щоб повернутися до питання клієнта в зручний для нього час. Для користувача тепер ще є нове вікно яке має завдання дати короткий звіт по файлу, тексту де саме трапилась помилка, скільки разів, що дана помилка означає. Такий інтерфейс покращить роботу сервісу технічної підтримки також тепер в вивченні помилок в log звітах тепер не має потреби. На практиці в деяких компаніях досі вручну переглядаються помилки у логах та надається відповідь клієнтам. Запити на практиці об’ємні, а концентрація та увага працівника можуть впливати на швидкість відповіді на питання та іноді негативно впливати на уявлення клієнта

про компанію, якщо не надати аргументовану відповідь клієнту на запит який виник під час користування сервісу. З такою програмою можна буде швидко аналізувати великі логи та одразу приступати до вирішення виявлених при діагностиці системи помилок, тим самим клієнт буде завжди впевнений у якості обслуговування сервісу технічної підтримки і мати позитивний досвід у вирішенні питань в форматі телефонного дзвінку на гарячу лінію. Розібравши позитивні сторони можна підвести підсумок та дати певну критику програмному забезпеченню з штучним інтелектом на базі MLP Classifier. Згідно опису роботи програми практичних випробувань було виявлено недоліки, наприклад сама по собі логіка фіксована, якщо виходити за рамки можливостей дозволеного програмою є можливість зустріти негативний досвід користування коли програма не зчитає певний новий для неї рядок проігнорувавши його існування або в цілому видати помилку при компіляції. Щоб вирішити це питання штучний інтелект треба буде вчити новому постійно і робити це вручну, що являє собою мінуси у вигляді неможливості реагувати на нові види складнощів у реальному часі та високим затратам часу і грошей на реалізацію оновлення програми. Тим самим можна зіткнутися з тим, що після оновлення сама програма може працювати гірше, ніж до оновлення.

Постійно потрібно слідкувати за оновленнями та регулювати складнощі переходу на нову версію, так як помилки можуть бути як прогнозованими так і не типовими.

3.4 Вдосконалення програми завдяки ключам API від OPEN AI

Для вдосконалення програми, реалізовано інтеграцію проекту з ключами API. Нажаль API від OPEN AI є платною послугою, що не стало на заваді продовженню розробки програми діагностики. Саму структуру програми залишено таку саму як і була до додавання API, вона лише трохи відрізнялась від уже існуючої саме у візуальній частині. Вже готову програму протестовано логами, з першою програми Ai Log Analyzer. Результат був дійсно схожим:

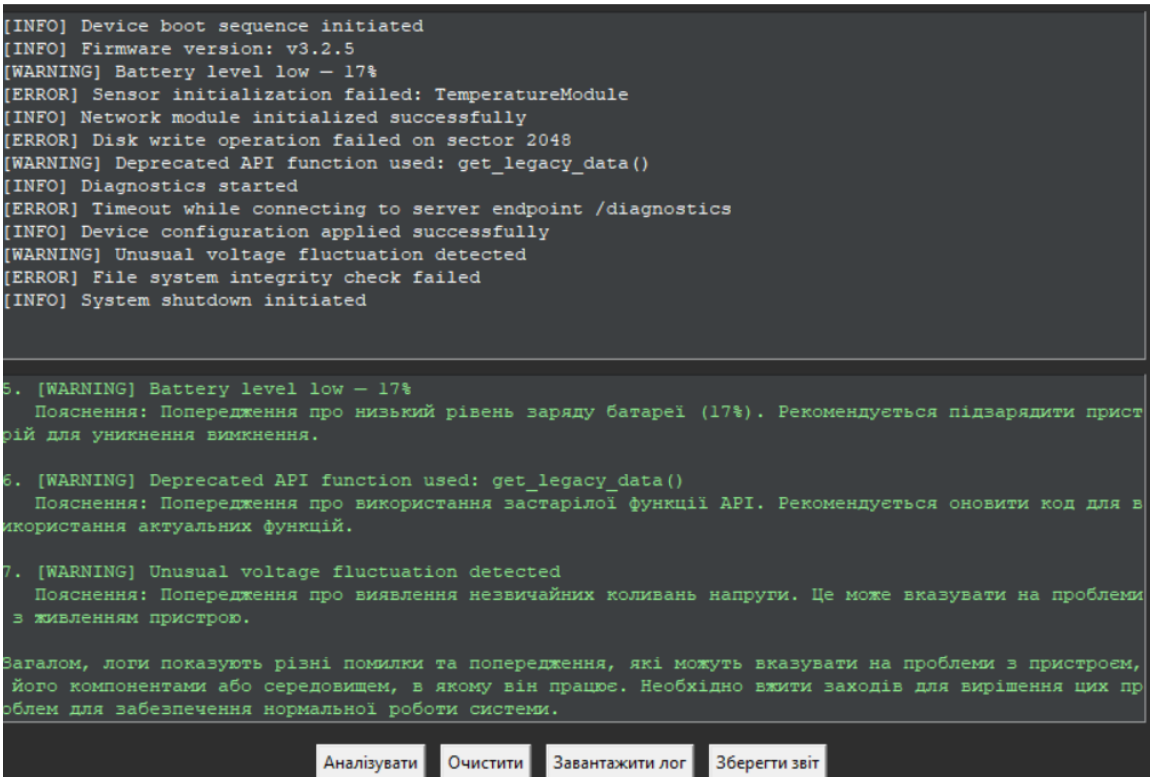

Рис. 3.7 Версія застосунку Ai Log Analyzer з API від OPEN AI.

Під час використання програми вже була помітна затримка в виконанні запиту, також тепер немає малого додаткового вікна зі сповіщенням тепер результат обробки запиту видавало аналогічно як у бета версії Ai Log Analyzer у низу під самим логом, але також додавався власний допис з більш легким поясненням від штучника з API. Вирішив ще протестувати можливості нового помічника, а саме вирішив зробити не коректні запити в Power Shell та відправити їх йому на аналіз помилок синтаксису. У Power Shell ввів команду Ping без уточнень, що саме треба буде перевіряти, який період часу, тому видало вікно з інформацією, що запит не було виконано через недостатньо описану команду, а потім ввів навмання набір символів на клавіатурі, щоб видало класичну помилку про некоректний синтаксис в такому вигляді скопіював отримане та відправив у пустий документ потім зберіг його у форматі txt, щоб відправити його нашому Ai Log Analyzer з API. Результат отримав почекавши секунди 4, він видав не лише описане вже у Power Shell, але й зміг доповнити опис для більш не ознайомлених людей з системою PS.

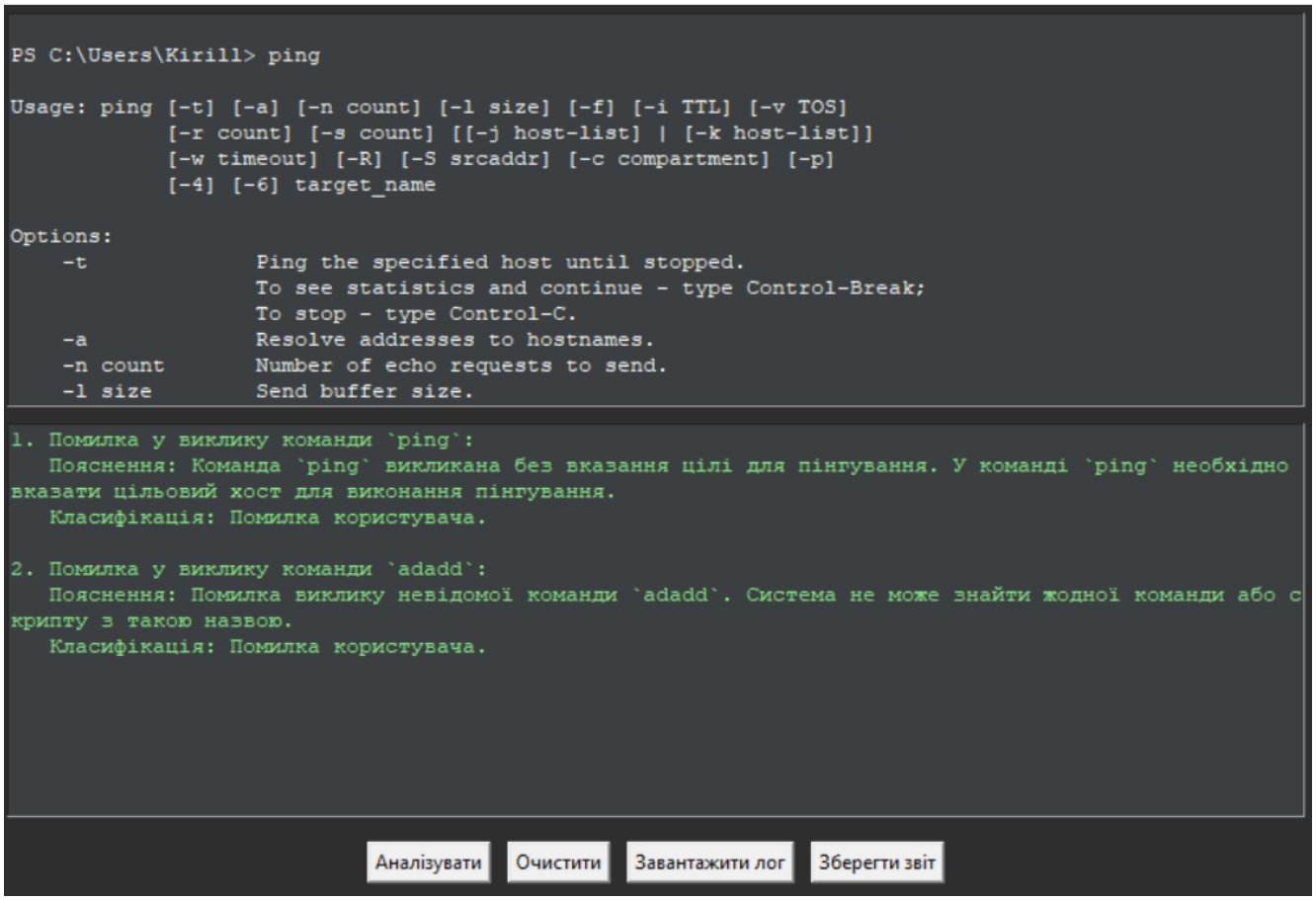

Рис. 3.8 Приклад тестового запиту.

Також опрацьовано запит не по темі запрограмованого коду, наприклад як приготувати млинці чи подібний запит, результат надав мені також через приблизно 3 або 4 секунди де штучний інтелект наголосив про те що, запит він виконати не може, тому що не бачить лог для опрацювання. Це також нам дає зрозуміти, що це просто умовний Gemini, GPT, Copilot і так далі, це дійсно штучний інтелект спрямований лише на читання логів. Дана програма є хорошим рішенням для компаній де треба інструмент для системних адміністраторів або наприклад для технічної підтримки розробників. Після тесту на нецільовий запит ще було цікаво як він відреагує на власний код, він описав деякі мої помилки допущені у коді які пізніше виправив, одразу після створення програма знадобилася та була корисною навіть для мене. Варто відзначити в цілому програма з API має потенціал не лише читати логи, вона могла робити б і більше якби це було б звісно описано в самому коді програми де знаходилось API.

3.5 Порівняння двох проектів аналізу логів між собою

В даному розділі маю мету приділити увагу дослідженню різниць обох лог-аналітичних систем створених з метою дослідження питання штучного інтелекту в діагностиці складних систем, обидві програми можуть внести користь певних компаніям які потребують постійного технічного діагностування зі сторони технічних інженерів. Обидві системи забезпечують можливість швидко зорієнтуватися як з нею взаємодіяти, тобто вони мають простий інтерфейс з темним оформленням для того щоб співробітник не мав певного навантаження на очі. Спільною рисою також є можливість як опрацювати власний файл, так і завантажити аналіз логу у власний клієнт. Також є можливість побачити результат обробки логу унизу інтерфейсу програми. Додатковою рисою хотілось би зазначити можливість очистити поле вводу та кнопку яка запускає процес аналізування. Після обговорення спільних рис маємо мету також дослідити і різницю між самими програмами між собою у вигляді таблиці:

Критерій оцінювання	**Ai Log Analyzer з API**	**Ai Log Analyzer**
Тип аналізу	Від готового штучного інтелекту	Вручну писана система яка розрізняє класи ERROR, WARN, INFO
Глибина аналізу	Визначає тип помилки, пояснює, класифікує	Знаходить помилку дає пояснення раніше описане в коді та рахує рядки та повторення помилок
Потреба в API	Дійсно залежний від API та ключа	Не потребує, готовий працювати без мережі
Швидкість обробки	Треба чекати певний час, зазвичай 3-4 секунди, очікування відповіді від API	Моментальне виконання запиту і висвітлення результату

Пояснення результату	Сам інтегрований штучник описує готове пояснення з більш легкою формою розуміння	Показує факт наявності помилки, а пояснення може і не бути якщо його не описати в коді
Зведена статистика	Немає, описує де помилка та її можливе рішення	Є, рахує скільки помилок, у яких рядках та класифікує по їх типам
Можливість розширення	API самооновлюється завдяки корпорації яка продає API доступ до ШІ	Обмежена треба вручну додавати дані у систему
Безпека в збереженні отриманих даних	Її немає, вони стають частиною системи та частиною програми навчання для ШІ яким ви користуєтесь з API	Є, програма незалежна від таких факторів як мережа і може існувати локально на одному пристрої
Оформлення(Дизайн)	Сірий, зелений, білий, немає додаткового вікна зі сповіщенням про помилки	Чорний та білий колір, має можливість надавати сповіщення в вигляді додаткового вікна
Повідомлення користувачу	Унизу в вигляді вже готового результату та власних думок від ШІ	Є як і унизу, так і у вигляді додаткового вікна зі зведеною статистикою
Завантаження в різні формати	CSV & TXT Сам лог	CSV & TXT Результат логу і сам лог
Відповідь на нецільовий запит	Вибачте, але у вашому повідомленні немає жодних логів для аналізу. Якщо у вас виникли проблеми з логами або ви	Проігнорує питання

	хочете їх проаналізувати, будь ласка, надайте більше інформації або скопіюйте логи сюди, щоб я міг допомогти вам з їх аналізом (скопіював)	

Таблиця 3.1 Порівняльний аналіз результатів тестування обох програм.

Розглянемо два принципово різних між собою підходи до реалізації лог аналітичних інструментів, які попри візуальну та функціональну схожість інтерфейсів для користувача, суттєво відрізняються між собою саме у внутрішній логіці обробки вхідних даних, гнучкості масштабування, а також у глибині інтерпретації результатів у відповідь на поставлену задачу. Другий в таблиці із зазначених в порівнянні підходів є умовно кажучи класичним. Його основи базуються на простих логічних правилах, орієнтованих на пошук специфічних ключових слів, рядків, класів наприклад як «ERROR». Реалізація такої системи може забезпечити базовий рівень діагностики шляхом виконання прямолінійного аналізу кожного рядка лог файлу з подальшою класифікацією запису як помилкового або й коректного. Хоча така модель дійсно відрізняється високою швидкістю обробки запиті, а також повною автономністю в плані не потреби бути поєднаною до будь-яких зовнішніх сервісів, вона є водночас досить жорстко детермінованою. Будь-яка зміна структури повідомлень або поява наприклад нових типів помилок вимагає від розробника ручного оновлення аналізу допустимих правил, що у свою чергу може знизити адаптивність програми до динамічних системних середовищ. Натомість другий підхід реалізований через інтеграцію з Open AI API є сучасною мовною моделлю на основі трансформованої архітектури, розробленої від великої корпорації. Така система не лише проводить базову ідентифікацію помилкових рядків, а й виконує контекстно-залежний систематичний аналіз, формуючи осмисленні пояснення виявлених під час аналізу помилок. Такий метод дає можливість отримувати не лише індикатор самої

помилки, а й отримувати глибше розуміння природи самої несправності, з урахуванням неявних закономірностей та міжрядкових зв’язків. Фактично друга програма виконує роль віртуального консультанта, який у режимі реального часу готовий та здатен генерувати технічні висновки, інтерпретуючи логи з рівнем складності, недосяжним для класичних логічних умов. Разом з тим, такий підхід супроводжується низкою певних критичних обмежень. Зокрема можна виділити, наявність залежності від зовнішнього API та його ключа, що створює також додаткові ризики, пов’язані з безпекою даних, стабільністю internet з’єднання, а також комерційною вартістю використання ресурсів API сервісів. Окрім того, тривалість обробки логів значною мірою все ж залежить від швидкодії вашої мережі та обмежень самої моделі. Не можна не згадати про ризики збереження конфіденційної інформації – передача логів на сторонній сервер завжди потребує додаткового контролю, що є критично важливим у випадках роботи з чутливими системами або наприклад користувацькими даними. Що маємо мету виділити, ключова різниця між обома розглянутими підходами полягає у їх глибині аналізу та інтелектуальній гнучкості. Вручну писана система, тобто Ai Log Analyzer є швидка та передбачена з трохи обмеженою аналітикою. Ai Log Analyzer з API гнучка, контекст адаптивна система, з певними елементами інтелектуального консультування, однак з підвищеними вимогами до інфраструктури та захисту даних. Вибір між цими обома рішеннями повинно здійснюватися з урахуванням специфікації предметної області дослідження, обсягу самих логів, дослідження рівня критичності системи, а також бюджету на обслуговування й оновлення програми. Таким чином порівняльний аналіз демонструє, що кожен із підходів не є універсальним рішенням, проте кожен може бути актуальних в межах потребуючих від них сил в певних обставинах. Тобто якщо ціль це швидкий та локальний перегляд логів на предмет виявлення помилок тут допоможе Ai Log Analyzer. Якщо ж мова буде йти вже про більш глибоку діагностику у складних, слабо формалізованих середовищах, чудовим рішенням було б Ai Log Analyzer з API.

ВИСНОВКИ

У кваліфікаційній роботі було проведено комплексне дослідження проблеми використання штучного інтелекту в процесах діагностики технічних систем. Отримані результати дозволяють зробити низку важливих теоретичних і практичних висновків, які мають значення для подальшого розвитку методів і технологій в цій галузі.

Проведений аналіз сучасного стану проблеми показав, що використання технологій штучного інтелекту в процесах діагностики програмного забезпечення є перспективним напрямком досліджень, який активно розвивається в останні роки. Зростання складності та масштабів сучасних програмних систем робить традиційні методи діагностики недостатньо ефективними, що зумовлює необхідність розробки нових, більш автоматизованих підходів на основі технологій штучного інтелекту. Вивчення наукових публікацій та практик впровадження у цій галузі дозволило виявити основні тенденції та напрямки досліджень, а також ідентифікувати існуючі обмеження та проблеми.

Результати теоретичного дослідження дозволили систематизувати методи штучного інтелекту, які можуть бути застосовані для автоматизації процесів діагностики програмного забезпечення. Було виявлено, що найбільш перспективними є методи на основі глибокого навчання, ансамблевих алгоритмів та генеративних моделей. Кожен з цих підходів має свої переваги та обмеження, які необхідно враховувати при розробці інтелектуальних систем діагностики.

На основі проведеного аналізу була розроблена концептуальна модель використання технологій штучного інтелекту в процесах діагностики програмного продукту. Запропонована модель включає чотири основні компоненти: модуль збору та обробки даних, модуль аналізу та класифікації помилок, модуль прогнозування потенційних проблем та модуль генерації рекомендацій щодо їх усунення. Важливою особливістю розробленої моделі є її адаптивність та можливість інтеграції з існуючими системами розробки та підтримки програмного забезпечення.

Для оцінки ефективності застосування технологій штучного інтелекту в процесах діагностики програмного забезпечення була розроблена спеціальна методика, яка враховує такі параметри як точність виявлення помилок, швидкість їх локалізації, повнота покриття коду аналізом, а також обсяг ресурсів процесу діагностики. Запропонована методика дозволяє об'єктивно оцінювати ефективність різних підходів та технологій в цій галузі.

На основі розробленої концептуальної моделі був створений прототип інтелектуальної системи діагностики комп'ютерних програм, який реалізує три основні функції: автоматичне виявлення помилок у вихідному коді, класифікацію виявлених помилок за типами та ступенем критичності, а також генерацію рекомендацій щодо їх усунення. Система використовує комбінацію методів статичного та динамічного аналізу коду, а також алгоритми машинного навчання для підвищення ефективності діагностики.

Експериментальне дослідження розробленої системи проводилося на реальних проектах програмного забезпечення різного масштабу та складності. Результати експерименту показали, що використання технологій штучного інтелекту дозволяє підвищити точність виявлення помилок в порівнянні з традиційними методами, зменшити час на їх локалізацію, а також забезпечити більш високий рівень автоматизації процесу діагностики. Особливо ефективним виявилося застосування розробленої системи для виявлення складних помилок, які важко ідентифікувати за допомогою стандартних підходів.

Важливим результатом дослідження є також визначення обмежень та проблем, пов'язаних з використанням технологій штучного інтелекту в процесах діагностики комп'ютерних програм. Було виявлено, що ефективність інтелектуальних систем діагностики значною мірою залежить від якості та обсягу навчальних даних, а також від специфіки конкретних проектів програмного забезпечення. Крім того, існують проблеми інтерпретації результатів, отриманих за допомогою методів штучного інтелекту, що може ускладнювати їх практичне використання розробниками програмного забезпечення.

На основі результатів дослідження були сформульовані практичні рекомендації щодо інтеграції технологій штучного інтелекту в існуючі процеси розробки та підтримки програмного забезпечення. Ці рекомендації включають вибір оптимальних методів та технологій залежно від специфіки проекту, організацію збору та підготовки даних для навчання інтелектуальних систем, а також підходи до оцінки ефективності їх застосування в реальних умовах.

Аналіз результатів дослідження дозволяє зробити висновок про підтвердження гіпотези щодо можливості підвищення ефективності процесів діагностики комп'ютерних програм за рахунок використання технологій штучного інтелекту. Розроблені в рамках дослідження методи, моделі та програмні рішення можуть бути використані для створення нового покоління інтелектуальних систем діагностики, які дозволять автоматизувати та оптимізувати процеси виявлення, локалізації та усунення помилок у програмному забезпеченні.

Водночас, результати дослідження виявили ряд перспективних напрямків для подальших досліджень у цій галузі. Зокрема, актуальними завданнями є розробка методів, що дозволяють ефективно працювати з обмеженими наборами навчальних даних, створення інтелектуальних систем, здатних адаптуватися до специфіки конкретних проектів програмного забезпечення, а також підвищення інтерпретації результатів, отриманих за допомогою методів штучного інтелекту.

Практична цінність результатів дослідження підтверджується їх успішним впровадженням у процеси розробки та підтримки програмного забезпечення в рамках пілотного проекту. Використання розробленої інтелектуальної системи діагностики дозволило підвищити якість програмного забезпечення, зменшити кількість помилок, які потрапляють до кінцевих користувачів, а також знизити витрати на тестування та підтримку.

Теоретична значущість отриманих результатів полягає у розвитку наукових основ використання технологій штучного інтелекту в процесах діагностики програмного забезпечення, систематизації та узагальненні існуючих підходів, а також у формулюванні нових теоретичних положень щодо оцінки ефективності застосування технологій штучного інтелекту в даній галузі.

Узагальнюючи результати дослідження, можна стверджувати, що використання технологій штучного інтелекту в процесах технічних систем є перспективним напрямком, який дозволяє суттєво підвищити ефективність виявлення, локалізації та усунення помилок в роботі системи. Розроблені в рамках кваліфікаційної роботи методи, моделі та програмні рішення створюють основу для подальшого розвитку цього напрямку та його практичного застосування в індустрії розробки програмного забезпечення.

Таким чином, мета дослідження була досягнута, а поставлені завдання успішно вирішені. Отримані результати мають як теоретичне, так і практичне значення для розвитку методів і технологій у галузі системного аналізу та штучного інтелекту. Подальші дослідження в цьому напрямку дозволять розробити більш ефективні та адаптивні інтелектуальні системи діагностики технічних систем, які зможуть стати невід'ємною частиною сучасних економічних процесів.

Важливим показником ефективності системи є також її здатність до навчання та адаптації. Експерименти показали, що з часом точність виявлення помилок та якість рекомендацій системи підвищуються завдяки механізму зворотного зв'язку, який дозволяє враховувати реакцію розробників на результати аналізу. Тобто такий підхід на основі навчання з підкріпленням дозволяє системі адаптуватися до специфіки конкретного проекту та покращувати свої результати з кожною інтеграцією аналізу.

Експерименти також показали, що використання розробленої системи особливо ефективне на ранніх етапах життєвого циклу програмного забезпечення, коли вартість виправлення помилок є найменшою. Адже виявлення та усунення помилок на етапі проектування та початкової розробки може бути в 10-100 разів дешевшим, ніж на етапі експлуатації. Розроблена система, завдяки своїй здатності виявляти потенційні проблеми на основі аналізу коду та ранніх ознак аномальної поведінки, дозволяє значно збільшити частку помилок, виявлених на ранніх етапах.

Результати експериментів також виявили певні обмеження розробленої системи, які представляють напрямки для подальшого вдосконалення. Зокрема, ефективність системи в деяких випадках обмежується якістю та обсягом доступних навчальних даних, особливо для нових типів проектів та технологій. Ця проблема може бути вирішена за допомогою методів трансферного навчання та активного навчання, які дозволяють ефективно використовувати обмежені навчальні дані.

Таким чином, практична реалізація та експериментальна оцінка розробленої інтелектуальної системи діагностики програмного забезпечення показали її високу ефективність у виявленні та класифікації помилок, а також у формуванні рекомендацій щодо їх усунення. Система демонструє значні переваги порівняно з традиційними методами аналізу коду, особливо при виявленні складних та нетривіальних помилок. Подальше вдосконалення системи пов'язане з розширенням бази навчальних даних, покращенням методів інтерпретації результатів та оптимізацією інтеграції з існуючими процесами розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «Концепція розвитку штучного інтелекту в Україні» № 1556-р від 2 грудня 2020 р. URL: https://www.kmu.gov.ua/npas/pro-shvalennya-koncepciyi-rozvitku-shtuchnogo-intelektu-v-ukrayini-s21220
2. Лубко Д. В., Шаров С. В. «Методи та системи штучного інтелекту: навчальний посібник» – Мелітополь: ФОП Однорог Т. В., 2019. – 264 с.
3. Androschuk G.O. Level of trust and attitude towards artificial intelligence: analysis of research results. Journal of the Kyiv University of Law. 2021. No. 3. P.195-201.
4. Звенігородський О.С., Зінченко О.В., Чичкарьов Є.А., Кисіль Т.М. Штучний інтелект. Вступний курс: Навчальний посібник. – К.: ДУТ, 2022. – 193 с.
5. Лащевська, Н. О. Швидке і ефективне поліпшення якості зображення за допомогою згорткової нейронної мережі / Лащевська Н. О., Недзельський О. Ю. // Радіотехнічні проблеми, сигнали, апарати та системи: матеріали X Міжнародної науково-технічної конференції [Київ, 09-11 листопада 2021 р.]. – Київ, 2021. С. 54–56.
6. Бостром, Н. «Суперінтелект: шляхи, небезпеки, стратегії.» Оксфорд: Oxford
7. Тегмарк, М. Життя 3.0: Бути людиною в епоху штучного інтелекту. Нью-Йорк: Alfred A. Knopf, 2017, 365 с.
8. Стрілінський А., Дейнеко А. О. «Сучасні технології проектного менеджменту для розроблення систем комп'ютерного зору // «Прикладні комп'ютерні науки»: збірник тез доп. науково-практ. конф.. 27 квітня 2023 року, Львів. – Львів: ПЗВО «ІТ СТЕП Університет», 2023. С. 118–119.
9. Клян А. «Правове регулювання штучного інтелекту в Україні та в світі» URL:https://golaw.ua/ua/insights/publication/pravove-regulyuvannya-shtuchnogo-intelektu-v-ukrayini-ta-sviti/
10. Буров М. «Хто несе відповідальність за помилки штучного інтелекту?» URL: http://uz.ligazakon.ua/ua/magazine_article/EA012676

11. Тиш Є.В., Литвиненко Я.В. «Надійність, контроль, діагностика та експлуатація ЕОМ»: Конспект лекцій. – Тернопіль, 2020. – 150 с.
12. В.В. БАТАРЕЄВ «Методи та системи штучного інтелекту»: Стаття. – Вісник ХНУ, 2021. – С. 17-21
13. Стаценко Д.В., Стаценко В.В., Злотенко Б.М., Демішонкова С.А. «Дослідження програм на основі штучного інтелекту в якості комп'ютерних засобів захисту інформації»: Стаття. – ТНУ імені В.І. Вернадського, 2023. – С. 244–249.
14. Витвицький Р., Якубовський В. «Використання штучного інтелекту та машинного навчання для автоматизації процесів тестування програмного забезпечення в Україні» »: Стаття. – Вісник ХНУ, 2024. – С. 21-27
15. Довженко Т. П., Бондарчук А. П. «Аналіз сучасного етапу розвитку штучного інтелекту в телекомунікаціях»: Стаття. – ДУІКТ, 2025. – С. 43–48.
16. Авраменко А.С., Авраменко В.С., Косенюк Г. В. «Тестування програмного забезпечення»: Навчальний посібник. – Черкаси: ЧНУ імені Богдана Хмельницького, 2017. – 284 с.
17. Синєкоп Ю. С., Продеус А. М., Швець Є. Я., Кісельов Є. М., Баран М. М. «Експертні системи в медицині»: Навчальний посібник. – Запоріжжя: Видавництво ЗДІА, 2014. – 332 с.
18. Месюра В.І., Яровий А.А., Арсенюк І. Р. «Експертні системи. Частина 1»: Навчальний посібник. – Вінниця: ВНТУ, 2006. – 114 с.
19. Козак О.Л. «Аналіз вимог до програмного забезпечення»: Опорний конспект лекцій. – Тернопіль: ТНЕУ, 2011. – 56 с.
20. Когут Ю.І. «Штучний інтелект і безпека»: Практичний посібник. – Київ: СІДКОН, 2024. – 294 с.