Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістра

за спеціальності 121 – Інженерія програмного забезпечення

На тему: Дослідження застосування методів штучного інтелекту в системах автоматизованого виявлення вразливостей веб-ресурсів

Засвідчую, що в цій кваліфікаційній роботі немає
запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-24М
_____/І.І Гречка/

Керівник
кваліфікаційної роботи _____ /Н.Н Шаповалова/

Завідувач кафедри _____ /А.М.Стрюк/

Кривий Ріг
2025

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А.М.Стрюк

«__» _____ 20__р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-24М Гречки Івана Ігоровича

Тема: Дослідження застосування методів штучного інтелекту в системах автоматизованого виявлення вразливостей веб-ресурсів. Затверджено наказом по КНУ №__ від «_» _____ 2025р.

Термін подання студентом закінченої роботи : «_» _____ 2025 р.

Вихідні дані по роботі: розроблювана система повинна реалізовувати AI для автоматизованого виявлення вразливостей веб-ресурсів які працюють за допомогою штучного інтелекту.

Зміст пояснювальної записки (перелік питань, що їх треба розробити): Провести аналіз ринку та існуючих систем автоматизованого виявлення вразливостей веб-ресурсів із застосуванням методів штучного інтелекту; визначити вимоги до фінальної версії програмного забезпечення; спроектувати архітектуру системи, її функціональні модулі та логіку взаємодії; розробити інтелектуальні модулі обробки HTTP-запитів і аналізу URL-адрес; реалізувати програмне забезпечення та провести комплексне тестування.

Перелік ілюстративного матеріалу: скріншоти екранних форм, блок-схеми функціоналу системи.

РЕФЕРАТ

АВТОМАТИЗАЦІЯ, AI, МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ

Пояснювальна записка: 99 сторінок, 32 зображень, 8 таблиць, 20 джерел, 1 додаток.

Кваліфікаційна робота присвячена дослідженню та практичному застосуванню методів штучного інтелекту в системах автоматизованого виявлення вразливостей веб-ресурсів. Актуальність теми зумовлена стрімким зростанням кількості кіберзагроз, складністю сучасних веб-додатків та потребою у швидкому й надійному виявленні потенційних порушень безпеки.

Традиційні методи тестування та аналізу вразливостей часто вимагають значних ресурсів і не завжди забезпечують необхідну швидкість реакції, що робить інтеграцію засобів штучного інтелекту важливою складовою сучасних систем кіберзахисту.

У роботі проаналізовано існуючі підходи до автоматизованого виявлення вразливостей, зокрема методи класичного статичного та динамічного аналізу, сигнатурні системи, а також інтелектуальні моделі машинного навчання. Особливу увагу приділено застосуванню алгоритмів класифікації HTTP-запитів, аналізу параметрів URL-адрес та виявленню ознак фішингових ресурсів.

ABSTRACT

AUTOMATION, AI, ARTIFICIAL INTELLIGENCE METHODS

Explanatory note: 99 pages, 32 images, 8 tables,1 appendix, 20 sources

The qualification work is devoted to the research and practical application of artificial intelligence methods in automated systems for detecting vulnerabilities in web resources. The relevance of the topic is driven by the rapid growth of cyber threats, the increasing complexity of modern web applications, and the need for fast and reliable identification of potential security breaches.

Traditional methods of testing and vulnerability analysis often require significant resources and do not always provide the necessary response speed, which makes the integration of artificial intelligence tools an essential component of modern cybersecurity systems.

The work analyzes existing approaches to automated vulnerability detection, including classical static and dynamic analysis methods, signature-based systems, and intelligent machine-learning models. Particular attention is paid to the use of algorithms for classifying HTTP requests, analyzing URL parameters, and identifying indicators of phishing resources.

ЗМІСТ

ВСТУП

У сучасному цифровому світі безпека веб-ресурсів є наріжним стабільного функціонування будь-якої інформаційної системи. Веб-додатки, є основою взаємодії користувачів із сервісами, щодня стають мішенню для тисяч кіберзагроз. Традиційні методи виявлення вразливостей, такі як статичний аналіз коду, чи динамічне тестування залишаються корисними, однак вони мають суттєві обмеження. Вони вимагають ресурсів, займають багато часу, та, що найголовніше, часто є безсилими проти нових, складних або добре прихованих типів загроз.

Саме тут на допомогу приходить штучний інтелект. Методи машинного навчання, нейронні мережі та алгоритми обробки великих даних дозволяють створювати самонавчальні моделі, здатні глибоко аналізувати код, поведінку програм і мережевий трафік, виявляючи потенційні загрози з набагато вищою точністю. Це відкриває шлях до якісно нового рівня кіберзахисту, де автоматизовані системи не просто слідують заздалегідь прописаним правилам, а доповнюють та посилюють людський інтелект.

Актуальність цього дослідження важко переоцінити. Інтеграція штучного інтелекту в системи безпеки, це не просто спосіб прискорити аналіз. Це критична необхідність для адаптації до мінливого ландшафту кіберзагроз, де атаки стають все більш витонченими і швидкими.

Ця робота присвячена аналізу сучасних підходів до виявлення вразливостей веб-ресурсів.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Актуальність теми кваліфікаційної роботи

Актуальність теми кваліфікаційної роботи полягає в тому, що із розвитком цифрових технологій питання кібербезпеки стає критично важливим для всіх сфер діяльності. Веб-ресурси щодня піддаються атакам, і навіть незначна вразливість може призвести до витоку конфіденційних даних або зупинки роботи сервісу. Традиційні методи пошуку вразливостей це ручне тестування, статичний та динамічний аналіз, вони часто потребують багато часу і не завжди здатні виявити складні або нові типи загроз.

Застосування методів штучного інтелекту дає змогу підвищити ефективність систем виявлення вразливостей. Машинне навчання дозволяє аналізувати великі обсяги коду, поведінку веб-додатків та мережевий трафік, автоматично визначаючи підозрілі закономірності. Такий підхід дає можливість швидко реагувати на потенційні загрози і зменшує вплив людського фактору.

Отже, дослідження застосування методів штучного інтелекту в автоматизованих системах виявлення вразливостей має не лише теоретичне, а й практичне значення. Його результати можуть бути використані для вдосконалення процесів безпеки веб-ресурсів, створення інструментів аналізу коду нового покоління та підвищення загального рівня захищеності інформаційних систем.

1.2 Цілі та завдання кваліфікаційної роботи

Ціллю даної кваліфікаційної роботи є дослідження можливостей застосування методів штучного інтелекту для автоматизованого виявлення вразливостей у веб-ресурсах та оцінка ефективності таких підходів порівняно з традиційними методами аналізу безпеки, наприклад:

1. Проаналізувати сучасні підходи до виявлення вразливостей. Зібрати та систематизувати інформацію з наукових джерел, технічної документації та практичних рішень щодо існуючих методів безпеки статичного, динамічного аналізу, фаззінгу та інших.
2. Дослідити роль машинного навчання у виявленні вразливостей. Розглянути моделі та алгоритми, що використовуються для автоматизації процесу аналізу безпеки веб-додатків.
3. Розробка критеріїв оцінки: Розробити об'єктивні критерії для оцінки ефективності та придатності кожної методології тестуванн, враховуючи особливості проєктів.
4. Порівняти ефективність класичних і AI-орієнтованих методів. Визначити їхні переваги, недоліки, області застосування та можливості інтеграції в реальні системи.
5. Розробити критерії оцінки якості виявлення вразливостей. Сформувати набір показників для об’єктивного порівняння точності, швидкодії та стабільності різних підходів

1.3 Впровадження ШІ у кібербезпеку

Сучасні підходи прагнуть будувати єдині моделі, здатні одночасно виявляти XSS- та SQL-ін’єкції, зокрема метод UniEmbed [16]. Веб Сайти та додатки постійно генерують гігабайти логів, запитів і метаданих, які людині просто неможливо опрацювати вручну. Алгоритми машинного навчання роблять це за секунди. Наприклад, нейронні мережі здатні перевіряти трафік і знаходити навіть дуже дрібні підозрілі дії, які можуть свідчити про DDoS-атаку чи спробу SQL-ін’єкції. Це особливо важливо для банківських або медичних систем, де затримка у виявленні загрози може мати серйозні наслідки.

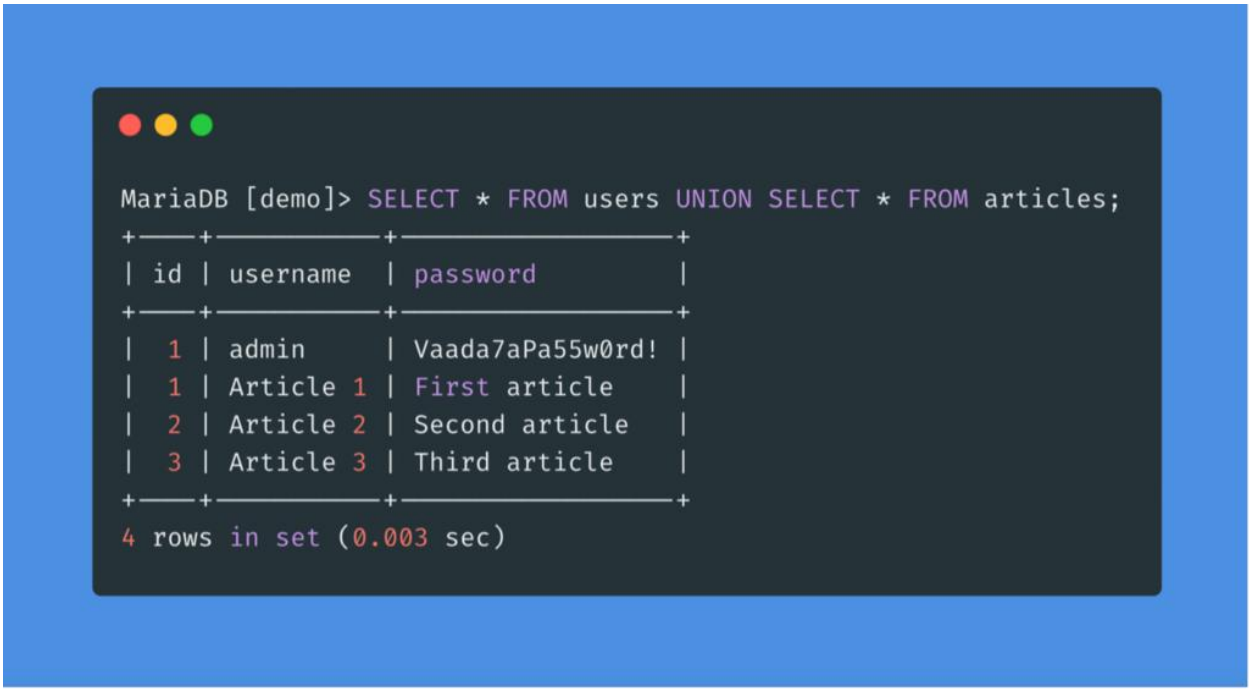

Рисунок 1.1 – Приклад SQL Injection

1. Зменшення кількості хибно-позитивних спрацьовувань.

Класичні сканери безпеки часто реагують занадто чутливо і позначають безпечні дії як небезпечні. Через це фахівці витрачають багато часу на перевірку. Моделі машинного навчання навчаються на попередніх даних і краще розуміють контекст. Наприклад, система може відрізнити звичайну дію адміністратора від підозрілої активності невідомого користувача.

2. Адаптація до динамічного середовища. Кіберзагрози постійно змінюються: з’являються нові методи атак, фішингові схеми, хитрі SQL-ін’єкції. Системи, які використовують штучний інтелект, можуть самі оновлювати свої моделі, навчаючись на нових прикладах. Наприклад, рекурентні нейронні мережі (RNN) аналізують журнали подій і можуть передбачати можливу атаку, ґрунтуючись на попередніх тенденціях

3. Автоматизація рутинних операцій. ШІ не просто знаходить проблеми, а й реагує на них. Якщо інтегрувати його з системами типу SOAR, він може автоматично блокувати небезпечні IP-адреси, видаляти шкідливий код або навіть пропонувати способи виправлення. Наприклад, моделі, які аналізують програмний код, здатні підказати розробнику, як виправити вразливість SQL-ін’єкції, спираючись на приклади з баз даних OWASP.

1.4 Основні атаки та техніка побудови шкідливих запитів

Безпека веб-додатків значною мірою залежить від здатності системи коректно обробляти вхідні дані, що надходять через HTTP-запити. Зловмисники використовують специфічні шаблони, некоректні параметри та навмисно модифіковані конструкції запитів для обходу стандартних механізмів перевірки та експлуатації вразливостей. Вивчення техніки побудови шкідливих запитів є фундаментом для створення ефективних систем автоматизованого виявлення аномалій та атак із використанням методів машинного навчання.

1.4.1 SQL INJECTION

SQL Injection це атака, під час якої зловмисник впроваджує у параметри HTTP-запиту фрагменти SQL-коду з метою зміни логіки виконання запиту до бази даних. Типові ознаки SQL-payload:

1. символи ', ", ```, --, /*…*/
2. ключові слова: UNION, SELECT, DROP, INSERT, OR 1=1
3. логічні конструкції: OR 'a'='a', AND 1=2

Приклад шкідливого GET-запиту: /login?username=admin' OR 1=1 --&password=x

Подібні запити дозволяють обійти аутентифікацію або отримати доступ до конфіденційних даних. Моделі ШІ повинні вміти розпізнавати структурні та семантичні особливості SQL-вставок навіть у випадках їх

обфускації чи кодування. Сучасні підходи до виявлення та пріоритизації SQL-ін'єкцій детально розглянуто в роботі [13]

1.4.2 Cross Site Scripting

Це атака, під час якої в параметри HTTP-запиту впроваджується JavaScript-код, що виконується у браузері користувача.

Типові XSS-пейлоуди:

1. <script>alert(1)</script>
2. "><img src=x onerror=alert(1)>
3. закодовані форми (URL-encoding, Base64, Hex)

Складність виявлення XSS полягає в тому, що код може бути розбитим на частини, зашифрованим або маскованим HTML-сущностями:

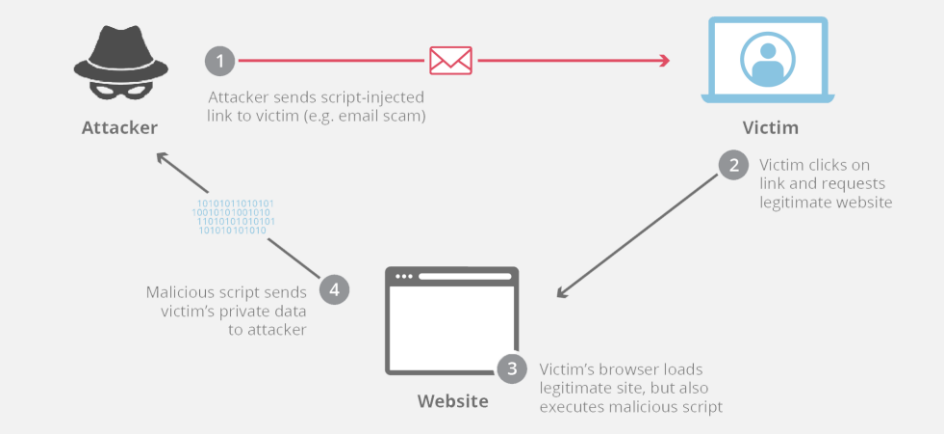

Рисунок 1.2 – Приклад Cross-Site-Scripting атаки

1.4.3 Path Traversal (Directory Traversal)

Атака Path Traversal дозволяє отримати доступ до файлів, які не передбачені логікою додатку, шляхом маніпуляції шляхами у параметрах.

Ознаки:

1. ../, ..%2F, %2e%2e%2f
2. абсолютні шляхи /etc/passwd, C:\Windows\System32\
3. %2F, %3C, %3E
4. \u003cscript\u003e

5. PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg

Зловмисники застосовують комбіновані техніки кодування, щоб обійти фільтри: подвійне URL-кодування, Unicode-кодування, змішування регістрів.

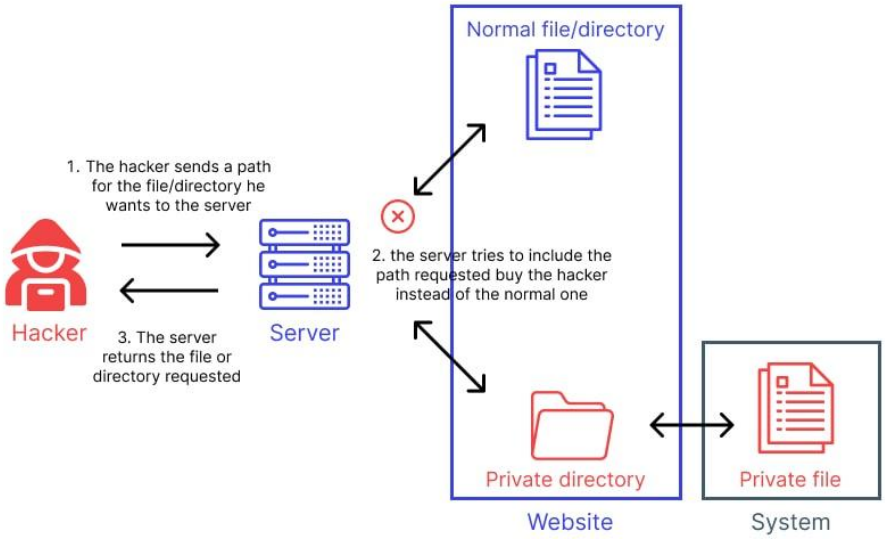

Рисунок 1.3 – Приклад Path Traversal атаки

Різноманіття технік побудови шкідливих запитів створює значні виклики для систем кіберзахисту. Атаки можуть бути очевидними або замаскованими під звичайні HTTP-запити, використовувати численні рівні кодування, нетипові параметри, комбінувати кілька технік одночасно. Саме тому застосування методів машинного навчання набуває особливої актуальності. Моделі ШІ здатні аналізувати не лише чіткі сигнатури, а й складні нечіткі залежності, що дозволяє виявляти як відомі типи атак, так і їх модифіковані варіанти.

1.5 Класифікація вразливостей веб-додатків

Класифікація вразливостей веб-додатків є важливою частиною аналізу кіберзагроз, оскільки дозволяє систематизувати основні типи атак, визначити механізми їх виникнення та сформувати підхід до побудови систем автоматизованого виявлення. Одним із найпоширеніших і визнаних

міжнародних стандартів класифікації є OWASP Top 10, який описує найбільш критичні ризики безпеки веб-додатків.

До найпоширеніших загроз належать ін’єкції тип атак, у яких зловмисник впроваджує шкідливі дані у параметри веб-запиту з метою модифікації логіки виконання серверного коду. Найвідомішими прикладами є SQL-ін’єкції, командні ін’єкції та LDAP-ін’єкції. Їхня суть полягає в тому, що некоректно оброблене або несанітізоване введення користувача може стати частиною команди, що виконується сервером.Функціональні вимоги до сканерів безпеки веб-додатків детально описані в специфікації NIST [9].

Перелік найпопулярніших атак:

1. A01:2021 Broken Access Control (Порушення контролю доступу)
    a. Некоректне обмеження прав користувача.
    b. Можливість переглядати або змінювати чужі дані.
    c. Відсутність перевірки ролей на рівні серверу.
    d. IDOR, обхід авторизації.
2. A02:2021 Cryptographic Failures (Криптографічні помилки)
    a. Використання слабких алгоритмів.
    b. Відсутність HTTPS.
    c. Неправильне зберігання паролів.
3. A03:2021 Injection (Ін’єкційні атаки)
    a. SQL, Command, NoSQL ін’єкції.
    b. Невірна валідація введення.
    c. Модифікація логіки запитів.
4. A04:2021 Insecure Design (Небезпечний дизайн)
    a. Відсутність безпечних архітектурних рішень.
    b. Логічні помилки у бізнес-процесах.
5. A05:2021 Security Misconfiguration (Неправильна конфігурація)
    a. Непотрібні модулі, відкриті порти.
    b. Debug-режими на продакшні.

6. A06:2021 Vulnerable and Outdated Components (Застарілі компоненти)
    a. Бібліотеки з відомими CVE.
    b. Застарілі фреймворки та плагіни.
7. A07:2021 Identification and Authentication Failures (Помилки автентифікації)
    a. Перебір паролів.
    b. Слабкі сесійні механізми.
    c. Відсутність MFA.
8. A08:2021 Software and Data Integrity Failures (Порушення цілісності даних і ПЗ)
    a. Неперевірені оновлення.
    b. Supply-chain атаки.
9. A09:2021 Security Logging and Monitoring Failures (Проблеми логування та моніторингу)
    a. Відсутність логів безпеки.
    b. Неможливість виявити атаки.

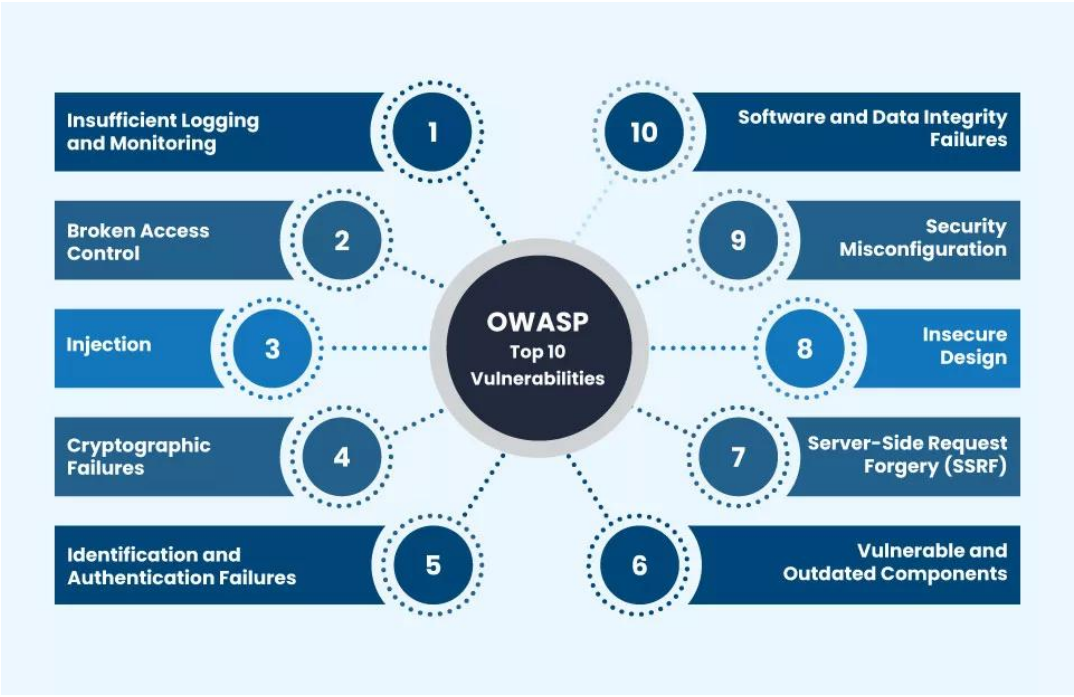

Рисунок 1.4 – Ключові признаки OWASP

Однією з ключових особливостей OWASP є акцент на причинах виникнення проблем, а не лише на наслідках. Значна частина вразливостей походить не з окремих помилок у коді, а з недосконалого проектування, неправильних конфігурацій або відсутності ефективних процесів контролю. Наприклад, у категорію Insecure Design входять помилки, що виникають ще на етапі архітектурного планування, коли питання безпеки не враховуються або інтегруються занадто пізно. Такі загрози часто неможливо повністю вирішити лише технічними правками вони потребують зміни підходу до проектування та розробки.

Також важливо відзначити, що багато категорій OWASP взаємопов'язані. Наприклад, Security Misconfiguration часто є передумовою для успішної експлуатації таких ризиків, як Injection або Broken Access Control. Відкриті debug-режими, неправильні дозволи файлової системи, доступність внутрішніх API та інші конфігураційні помилки значно збільшують площу атаки. У свою чергу, некоректна автентифікація або слабкі механізми керування сесіями з категорії Identification and Authentication Failures створюють умови для компрометації облікових записів, що може призвести до повного контролю над системою.

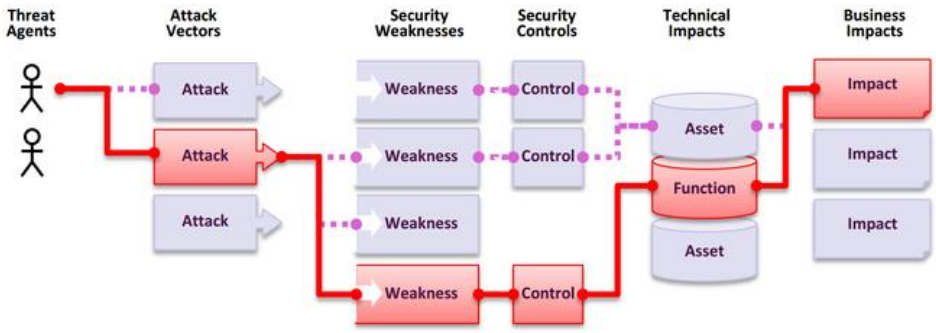

Рисунок 1.4 – Приклад обходу систем

Окремої уваги потребує категорія Vulnerable яка набуває дедалі більшої актуальності. Сучасні веб-додатки складаються з великої кількості сторонніх бібліотек, модулів та фреймворків. Будь-яка вразливість у них стає

потенційною точкою атаки. Ситуація ускладнюється тим, що багато команд розробки не мають налагоджених процесів відстеження CVE та оновлення залежностей. У складних програмних системах вручну виявити всі застарілі компоненти практично неможливо, тому автоматизовані засоби перевірки стають обов’язковими.

Категорія Monitoring Failures підкреслює, що навіть за наявності правильних технічних рішень система може залишатися вразливою, якщо відсутні логування, аудит та оперативне реагування. Багато успішних атак залишаються непоміченими саме через недостатність журналів подій або через те, що зібрані дані не аналізуються. Це створює сприятливі умови для довготривалих атак та латерального переміщення всередині систем.

2 ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В ЗАДАЧАХ КІБЕРЗАХИСТУ

За останні роки штучний інтелект (ШІ) став ключовим інструментом у сфері кібербезпеки. Його впровадження дозволяє автоматизувати процеси виявлення вразливостей, реагування на атаки та прогнозування потенційних загроз. Використання моделей машинного навчання (ML) та глибокого навчання (DL) значно підвищує точність аналізу та швидкість прийняття рішень у порівнянні з традиційними методами.

1. Ефективність використання ШІ в кіберзахисті полягає у підвищенні рівня безпеки інформаційних систем, скороченні часу реакції на інциденти та зниженні навантаження на фахівців з безпеки. Нижче наведено ключові аспекти ефективності застосування ШІ у цій сфері:
2. Моделі машинного навчання здатні аналізувати великі обсяги даних і виявляти складні закономірності, що недоступні для традиційних сигнатурних систем. Це дозволяє своєчасно розпізнавати нові види атак, зокрема zero-day уразливості..
3. Автоматизація процесів аналізу журналів, мережевого трафіку та системних подій знижує потребу у ручній перевірці. Системи на основі ШІ можуть самостійно класифікувати події та формувати звіти.
4. Моделі, які постійно навчаються, здатні оновлювати свої правила поведінки без прямого втручання людини. Це робить систему більш стійкою до змін у тактиках зловмисників.
5. Використання автоматизованих алгоритмів знижує ймовірність пропуску критичних подій або неправильного реагування, що часто трапляється під час ручного моніторингу.

2.1 Аналіз існуючих рішень та систем на основі машинного навчання

У сфері кіберзахисту машинне навчання (ML) стало одним із найважливіших напрямів розвитку технологій. Його застосування дозволяє системам безпеки не лише реагувати на вже відомі загрози, а й виявляти нові, раніше невідомі типи атак на основі поведінкових закономірностей. Основна перевага ML-підходів полягає у здатності моделей виявляти аномалії у великих обсягах даних без необхідності ручного створення сигнатур або правил.

Сучасні рішення з машинним навчанням умовно поділяються на комерційні платформи, що орієнтовані на корпоративний сектор, та відкриті дослідницькі проекти, які часто використовуються для наукових експериментів і навчальних цілей. Приклад застосування методів машинного навчання для виявлення CSRF-атак наведено в роботі [1]

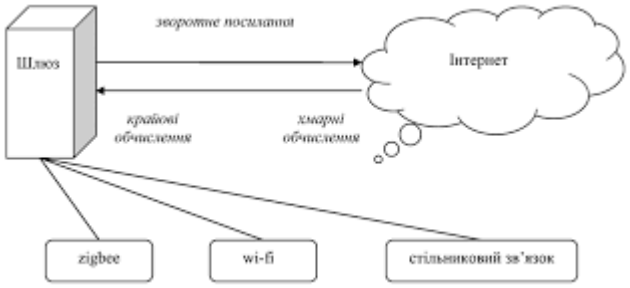

Рисунок 2.1 – Приклад обробки даних в системі

2.1.1 Аналіз існуючих рішень та систем на основі машинного навчання

Серед найвідоміших комерційних продуктів, які використовують ML-моделі для кіберзахисту, можна виділити такі:

- Darktrace це система на основі некерованого навчання, що створює «цифровий імунітет» організації. Алгоритми аналізують поведінку користувачів, пристроїв і мережевих потоків, формуючи базову модель «нормальної активності».

- Microsoft Security Copilot це інструмент, що інтегрує великі мовні моделі (LLM) у робочий процес аналітиків SOC, забезпечуючи автоматичний аналіз інцидентів і рекомендації щодо реагування.
- IBM QRadar Advisor with Watso**n** це платформа, яка використовує гібрид ML-підхід для виявлення складних атак, зокрема багатокрокових (multi-stage).

Такі рішення поєднують методи статистичного аналізу, поведінкових моделей і глибокого навчання, забезпечуючи швидке виявлення підозрілої активності без прямого втручання людини.

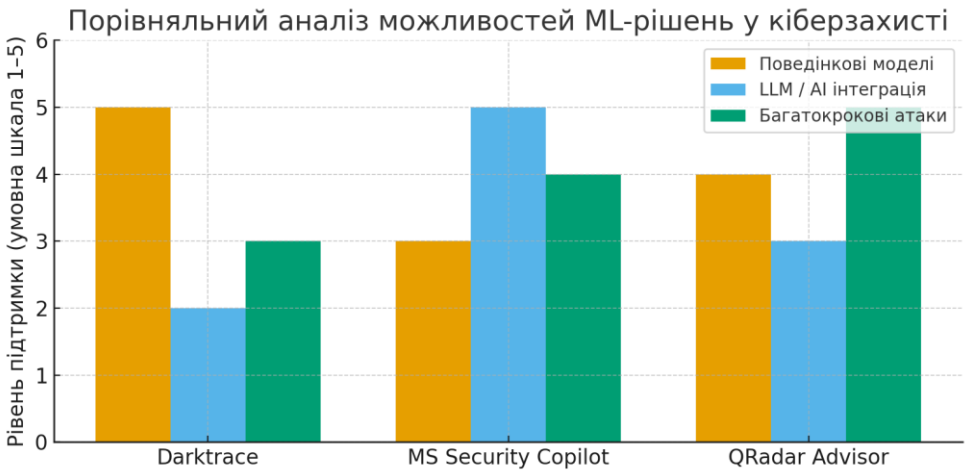

Рисунок 2.2 – Приклад порівняльного аналізу можливостей ML-рішень

2.1.2 Приклад використання машинного навчання для виявлення вразливостей у веб-додатках

Розглянемо приклад побудови простої системи виявлення потенційно небезпечних веб-запитів із використанням Python, бібліотек scikit-learn та Flask.Сучасні моделі глибокого навчання здатні автоматично виявляти складні варіанти SQL-ін’єкцій та XSS-атак [2]

У файлі app/decetor.py визначимо нашу програму Flask і простий ендпоинт для управління завданнями.

Наприклад:

```
from flask import Flask, request, jsonify
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier
import joblib
app = Flask(__name__)
samples = [
   "SELECT * FROM users WHERE name='admin'",
   "<script>alert('xss')</script>",
   "GET /index.html",
   "POST /login username=test"
]
labels = [1, 1, 0, 0]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(samples)
model = RandomForestClassifier().fit(X, labels)
joblib.dump((vectorizer, model), "model.pkl")
@app.route('/analyze', methods=['POST'])
def analyze_request():
   data = request.get_json()
   text = data.get("query", "")
   vectorizer, model = joblib.load("model.pkl")
   X_test = vectorizer.transform([text])
   result = model.predict(X_test)[0]
   if result == 1:
       return jsonify({"status": "malicious", "message": "Potential attack detected"}), 200
   else:
       return jsonify({"status": "safe", "message": "No threats detected"}), 200
```

2.2 Алгоритми машинного навчання для виявлення вразливостей у коді

Машинне навчання стало одним із ключових підходів до автоматизованого пошуку вразливостей у програмному коді. На відміну від статичних аналізаторів, які покладаються на заздалегідь визначені правила, ML-моделі здатні самостійно знаходити складні закономірності в структурі коду, що вказують на потенційні дефекти або уразливості.

Розглянемо найбільш ефективні алгоритми, які застосовуються в сучасних системах виявлення вразливостей.

Системи машинного навчання у цій сфері умовно поділяють на такі категорії:

- Алгоритми традиційної ML-класифікації (Random Forest, SVM, Decision Tree).
- Глибокі нейронні мережі (LSTM, CNN, трансформери).
- Графові нейронні мережі (GNN) для аналізу AST/CFG.
- Моделі на основі великомасштабних трансформерів

Наприклад:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier
import joblib
samples = [
   "user_input = input(); eval(user_input)",
   "cursor.execute('SELECT * FROM users ' + param)",
   "x = 10; y = 20; print(x+y)",
   "def add(a,b): return a+b"
]
labels = [1, 1, 0, 0]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(samples)
model = RandomForestClassifier().fit(X, labels)
joblib.dump((vectorizer, model), "rf_detector.pkl")
# Перевірка
```

```
test = ["eval(input())"]
vec, mdl = joblib.load("rf_detector.pkl")
prediction = mdl.predict(vec.transform(test))

print("Malicious" if prediction[0] == 1 else "Safe")
```

2.2.1 Некероване навчання (Supervised Learning)

Це тип машинного навчання, де модель навчається на розмічених даних (даних з готовими відповідями). Мета знайти зв'язок між вхідними даними (ознаками) та вихідними (мітками) щоб робити точні прогнози для нових, невідомих даних. Приклади побудови ML-моделей для виявлення SQL-ін’єкцій наведено в [14]

Приклад кроків для створення прогнозу:

1. Підготовка даних
    1.1. Збираються запити до веб-сервера та позначаються:
        1.1.1. 0 безпечний запит (/search?id=1).
        1.1.2. 1 SQL-ін'єкція (login.php?user=' OR 1=1 --).
2. Вибір ознак
    2.1. Довжина запиту.
    2.2. Наявність ключових слів (OR, --, UNION).
    2.3. Кількість спецсимволів (', ", =).

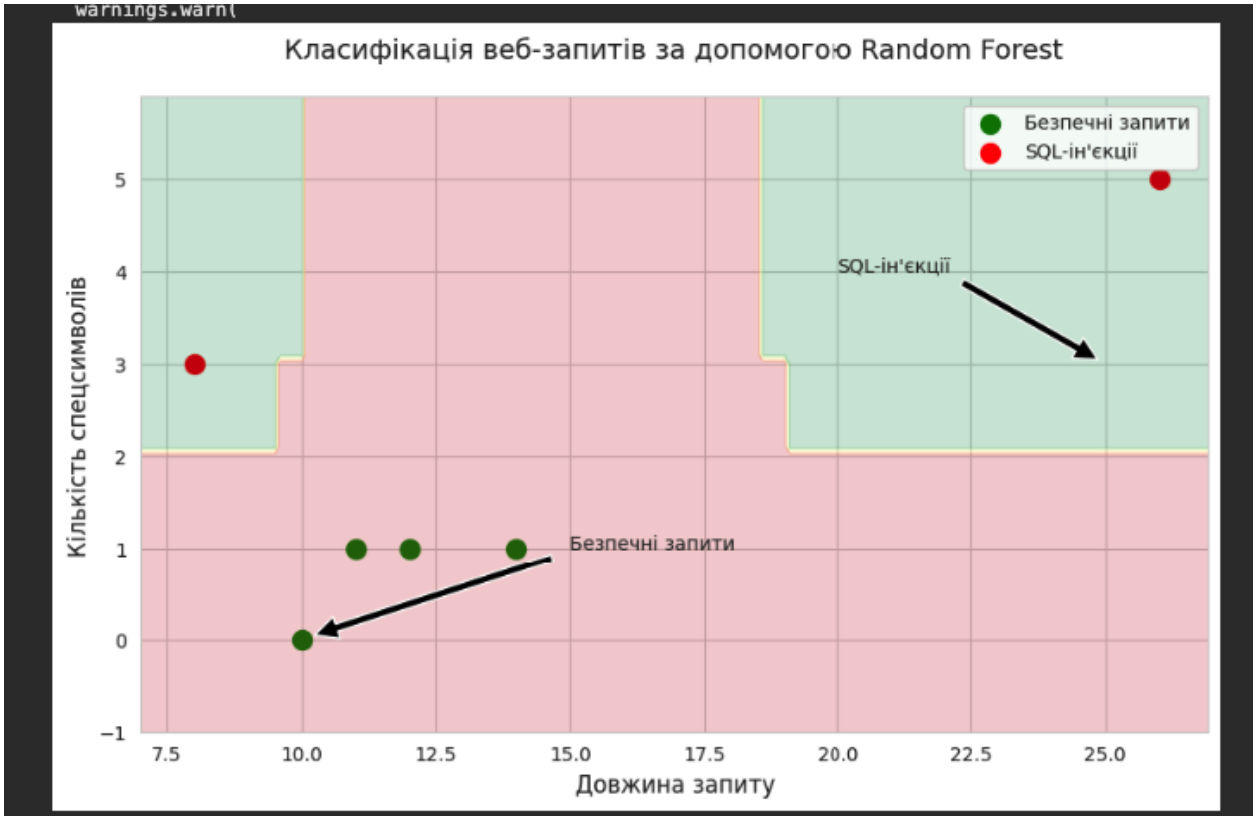

Рисунок 2.3 – Приклад Класифікації веб-запитів за допомогою RandomForest

Таблиця 2.1 – Недоліки та переваги класифікації веб-запитів за допомогою RandomForest

Переваги	**Недоліки**
Висока точність якщо якісні дані	Потрібні розмічені дані (часто дорого).
Зрозумілість результатів (напр., дерева рішень).	Може перенавчатися на шумових даних.
Швидке прогнозування після навчання.	Погано працює з новими типами атак (не були в тренувальних даних).

Цей метод особливо ефективний для задач, де необхідно класифікувати дані за попередньо визначеними категоріями (наприклад, виявлення

шкідливих веб-запитів) або прогнозувати числові значення (наприклад, оцінка ризику кібератаки). Моделі керованого навчання, такі як дерева рішень або методи ансамблю, працюють шляхом аналізу історичних даних для виявлення закономірностей, які потім можуть бути застосовані до нових, небачених даних.

Основна перевага цього підходу полягає в його здатності досягати високої точності прогнозів при наявності достатньої кількості якісних навчальних даних. Однак він має обмеження, оскільки вимагає попередньої розмітки даних, яка може бути трудомісткою, і може мати складності з узагальненням на абсолютно нові типи атак, яких не було в навчальному наборі.

2.2.2 Некероване навчання (Unsupervised Learning) для виявлення аномалій

Некероване навчання (Unsupervised Learning) це підхід у машинному навчанні, за якого алгоритм самостійно досліджує дані та виявляє приховані закономірності, не маючи попередніх міток або готових відповідей. На відміну від керованого навчання, де система отримує приклади з чітко визначеними результатами, тут модель сама визначає структуру вибірки, групує схожі об’єкти та виділяє відхилення від норми. Як показано в [10], різні підходи до виявлення аномалій мають суттєво відмінну чутливість до різних типів атак.

Завдяки цьому некеровані методи особливо ефективні у ситуаціях, де необхідно знайти нові, раніше невідомі аномалії, або швидко проаналізувати великі обсяги неструктурованих даних.

Таблиця 2.2 – Недоліки та переваги навчання (Unsupervised Learning) для виявлення аномалій

Краще підходить для	**Гірше підходить для**
Виявлення нетипової поведінки або аномальних шаблонів	Класифікація загроз, природа яких вже добре відома
Обробка даних, які не мають позначених класів	Сфери, від результату очікується гарантована точність
Попередній огляд великих масивів інформації	Завдання потребують детального розуміння моделі

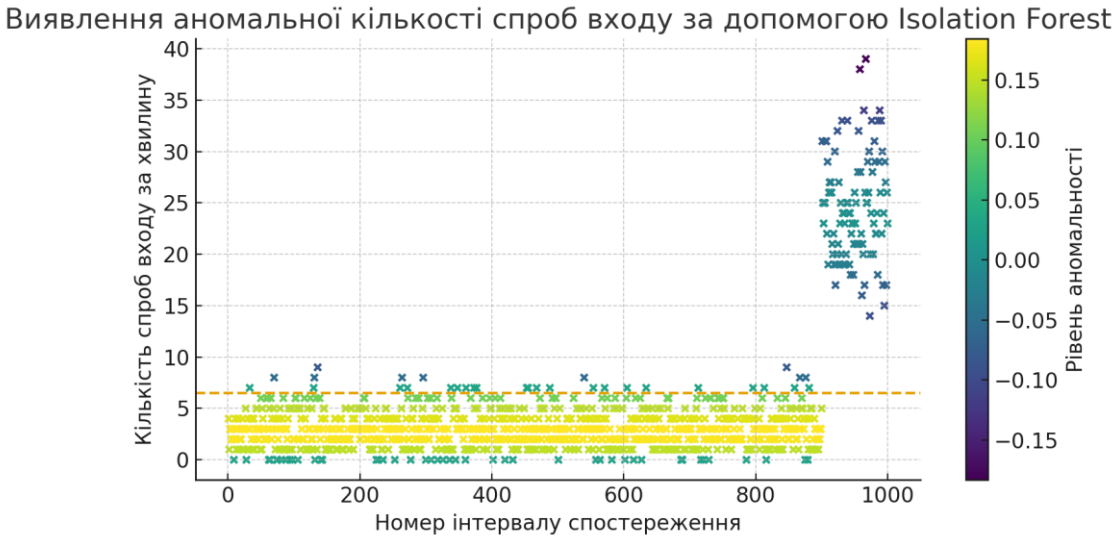

Рисунок 2.4 – Приклад виявлення аномальної кількості спроб входу за допомогою алгоритму Isolation Forest

2.2.3 Напівкеровані методи (Semi-Supervised Learning) у розпізнаванні шкідливих запитів

Метод напівкерованого навчання (Semi-Supervised Learning) для виявлення шкідливих веб-запитів – це гібридний підхід у машинному навчанні, який поєднує переваги керованого та некерованого навчання для ефективного аналізу веб-трафіку. Він особливо корисний у ситуаціях, коли є

обмежена кількість розмічених даних (наприклад, кілька сотень прикладів із відомими мітками "нормальний" або "шкідливий" запит) разом із великою кількістю нерозмічених даних (тисячі невідомих запитів).

Цей підхід особливо ефективний для виявлення нових, раніше невідомих типів атак, оскільки він не обмежується лише заздалегідь визначеними шаблонами. Моделі на основі механізму attention дозволяють здійснювати детекцію веб-атак у реальному часі [4]. Він також дозволяє значно зменшити витрати на підготовку даних, оскільки не вимагає повної розмітки великих масивів інформації. Однак його точність залежить від якості початкових розмічених даних якщо вони не репрезентативні або містять помилки, це може призвести до поширення неправильних міток.

Для покращення результатів метод часто комбінують із іншими підходами, такими як активне навчання, коли модель сама запитує розмітку для найбільш невизначених прикладів. Приклад застосування глибоких нейронних мереж для anomaly-based детекції веб-атак наведено в [11]

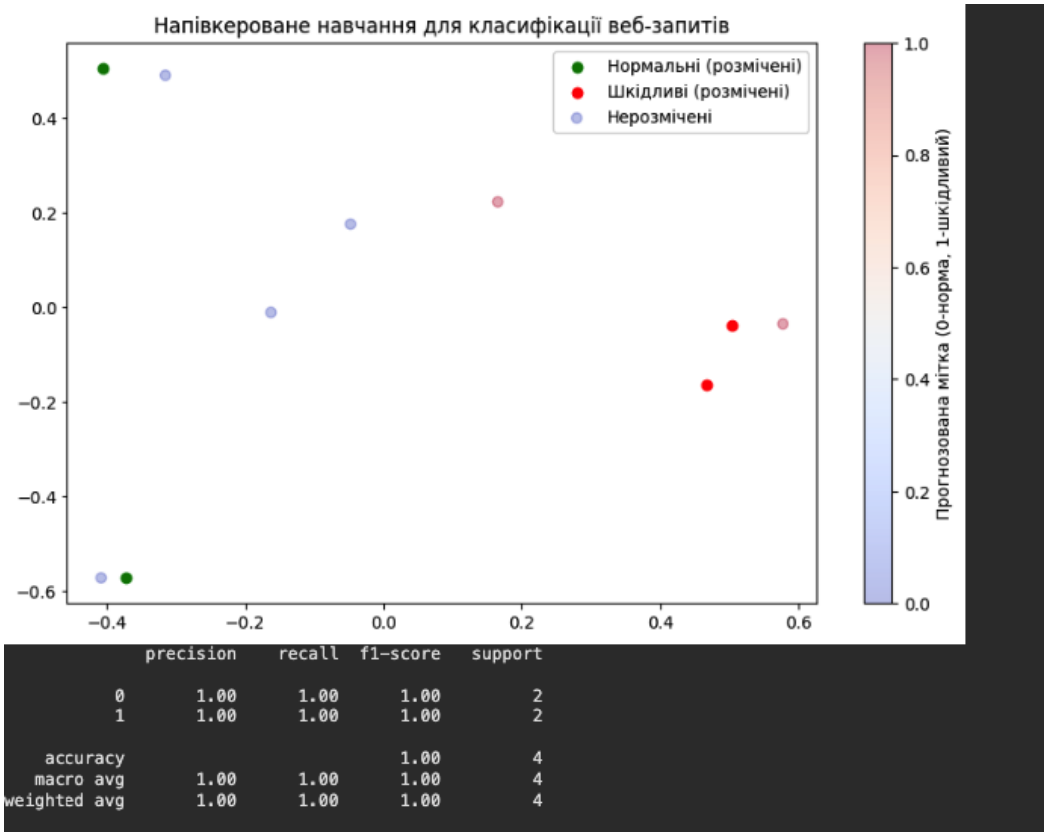

Рисунок 2.5 – Приклад напів керованого навчання для класифікації

Таблиця 2.3 – Недоліки та переваги напів керованих методів (Semi-Supervised Learning) у розпізнаванні шкідливих запитів

Переваги	**Недоліки**
Економія ресурсів на розмітку	Залежність від якості початкових міток
Виявлення нових типів загроз	Складність налаштування
Гнучкість до динамічних змін	Обмежена інтерпретованість

2.2.3 Глибоке навчання(DL)

Глибоке навчання (Deep Learning, DL) - це підрозділ машинного навчання, що використовує штучні нейронні мережі з багатьма шарами для автоматичного виявлення складних закономірностей у даних. На відміну від класичних алгоритмів, які потребують ручного вибору ознак (наприклад,

підрахунку спецсимволів у URL), DL самостійно виділяє ключові характеристики з необроблених даних.

Таблиця 2.4 – Недоліки та переваги глибокого навчання

Критерій	**Переваги**	**Недоліки**
Точність	Краща продуктивність на великих даних	Вимагає мільйонів прикладів для навчання
Гнучкість	Працює з неструктурованими даними (текст, логи)	Складність інтерпретації ("чорний ящик")
Ресурсність	Масштабується на GPU/TPU	Високі обчислювальні витрати

Глибоке навчання може революціонізувати процес пошуку вразливостей на веб-ресурсах завдяки здатності аналізувати складні патерни в великих обсягах даних. Основні способи застосування включають автоматичний аналіз коду, моніторинг трафіку в реальному часі та виявлення аномалій у поведінці системи.

Інший підхід до виявлення фішингових веб-ресурсів запропоновано в роботі [19], де автори використовують оптимізовану згорткову нейронну мережу (EGSO-CNN) для аналізу URL-адрес.

Для виявлення вразливостей у вихідному коді використовуються передові архітектури нейронних мереж, такі як трансформери (CodeBERT, GraphCodeBERT), які можуть аналізувати синтаксис коду на різних мовах програмування. Ці моделі навчаються розпізнавати потенційно небезпечні конструкції, такі як SQL-ін'єкції або XSS, навіть у складних випадках з обфускацією. У роботі [17] продемонстровано, що поєднання кількох

алгоритмів машинного навчання дає змогу значно зменшити кількість хибнопозитивних спрацювань.

Моніторинг веб-трафіку з використанням рекурентних нейронних мереж (LSTM) або згорток (CNN) дозволяє виявляти аномальні запити, які можуть свідчити про спроби експлойтування вразливостей. Моделі аналізують структуру HTTP-запитів, параметри URL та тіла запитів, знаходячи відхилення від нормальної поведінки.

Для виявлення складних атак, таких як SSRF або десеріалізація, застосовуються гібридні моделі, що поєднують методи обробки природної мови з традиційними підходами до аналізу безпеки. Наприклад, модель може одночасно аналізувати логіку коду та структуру вхідних даних.Робота [12] узагальнює ключові переваги та обмеження сучасних алгоритмів виявлення аномалій у веб-трафіку.

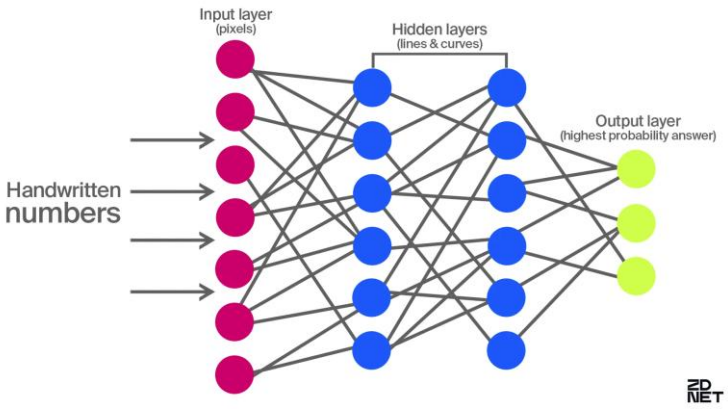

Рисунок 2.6 – Приклад дерева глибокого навчання

2.3 Обробка природної мови (NLP) для аналізу коду

Обробка природної мови (NLP) для аналізу коду це інноваційний підхід, який застосовує методи штучного інтелекту для автоматичного виявлення вразливостей, оптимізації та покращення якості програмного коду. Ця технологія розглядає програмний код як спеціальну форму мови,

використовуючи передові лінгвістичні моделі для його аналізу. Перспективним напрямом є використання квантових підходів до обробки коду, зокрема Quantum NLP-моделей [7].

Основні принципи роботи:

1. Код як текстова послідовність - NLP-системи аналізують код як послідовність токенів (ключові слова, оператори, ідентифікатори), виявляючи підозрілі патерни.
2. Контекстне розуміння - моделі враховують семантичні зв'язки між різними частинами коду, що дозволяє відрізняти реальні загрози від безпечних конструкцій.
3. Глибинний аналіз - сучасні системи використовують трансформерні архітектури (BERT, Codex) для розуміння складних залежностей у коді.

Переваги NLP-підходу:

- Здатність виявляти складні, контекстно-залежні вразливості
- Масштабованість для аналізу великих кодових баз
- Універсальність - одна модель може працювати з різними мовами програмування
- Автоматичне вдосконалення через навчання на нових прикладах

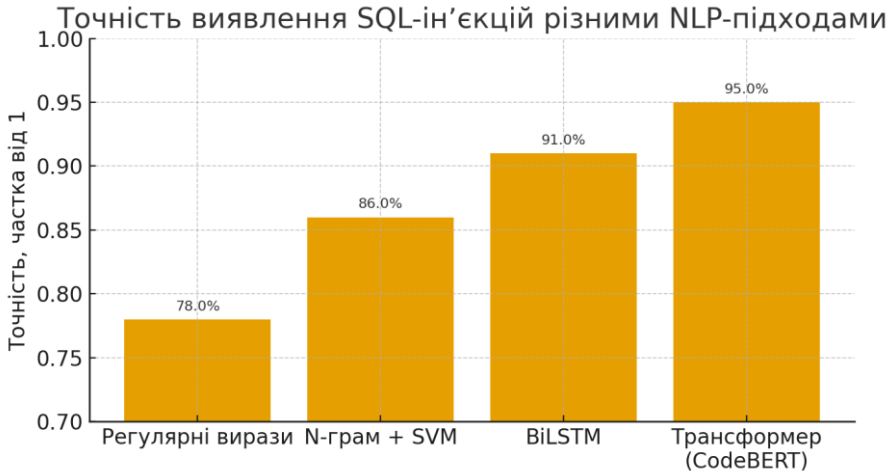

Рисунок 2.7 – Обробка точності виявлення SQL-ін’єкцій

```
import matplotlib.pyplot as plt
```

```
import numpy as np
models = [
   "Регулярні вирази",
   "N-грам + SVM",
   "BiLSTM",
   "Трансформер\n(CodeBERT)"
]
accuracy = [0.78, 0.86, 0.91, 0.95]
x = np.arange(len(models))
fig, ax = plt.subplots(figsize=(7, 4))

bars = ax.bar(x, accuracy)
ax.set_title("Точність виявлення SQL-ін’єкцій різними NLP-підходами")
ax.set_ylabel("Точність, частка від 1")
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.set_ylim(0.7, 1.0)
for i, v in enumerate(accuracy):
   ax.text(i, v + 0.005, f"{v*100:.1f}%", ha="center", va="bottom", fontsize=8)
plt.tight_layout()
plt.savefig("nlp_sql_detection_accuracy.png", dpi=300)
plt.show()
```

Для візуалізації результатів експериментального дослідження точності різних NLP-підходів до виявлення SQL-ін’єкцій було розроблено окремий програмний фрагмент на мові Python із використанням бібліотек NumPy та Matplotlib.

3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ

Розроблений програмний комплекс призначений для автоматизованого аналізу веб-ресурсів та пов’язаних із ними артефактів (HTML-код, структурні елементи сторінки, SEO-метадані тощо) із використанням методів штучного інтелекту. Користувач формує запит, у якому вказує тип аналізу (перевірка HTML-коду, пошук потенційних вразливостей, SEO-аудит та ін.), після чого система виконує відповідні обчислення та повертає структурований звіт з виявленими проблемами й рекомендаціями.

Основною вимогою до програмного комплексу є можливість гнучкого підключення декількох моделей машинного навчання, які спеціалізуються на різних задачах: одна модель може бути орієнтована на виявлення небезпечних конструкцій у HTML/JavaScript, інша на пошук типових SEO-помилок (відсутність meta-тегів, дубльовані заголовки, надмірні ключові слова тощо), ще інші на аналіз журналів запитів або структури посилань. Система повинна забезпечувати прозорий механізм маршрутизації запитів до відповідних моделей, а також можливість розширення набору доступних аналізаторів без суттєвої зміни архітектури.

З точки зору користувача, робота з комплексом повинна відбуватися через HTTP-API. Це дозволяє інтегрувати розроблену систему як окремий мікросервіс у вже існуючу інфраструктуру (панель адміністратора сайту, систему моніторингу безпеки, CI/CD-конвеєр тощо). Передбачається, що дані передаються у форматі JSON, а результати аналізу повертаються у вигляді машино-читаних структур із розподілом знайдених проблем за типами критичності.

Таблиця 3.1 – Основні компоненти програмної реалізації

Компонент	Призначення	Вхідні дані	Вихідні дані	Технології / Бібліотеки
Gateway API (Modules Router)	Приймає HTTP-запити, маршрутизує до AI-модулів	HTTP GET/POST	JSON-відповідь	FastAPI
Module Manager (Dispatcher)	Обирає потрібний модуль та обробляє помилки	Назва модуля, параметри	Результат модуля	Python
SecurityCheck	Перевірка SQLi/XSS у рядках	Текст	{vulnerable, type}	Regex / евристики
SEOCheck	Аналіз SEO: title, meta, h1	URL сайту	SEO-показники	httpx, BeautifulSoup4
HeadersCheck	Перевірка заголовків безпеки	URL	Список заголовків	httpx
SpamDetection	Антиспам-евристика	Текст	{spam, score}	Python евристики
Data Preprocessing	Очищення та нормалізація даних	Рядки/HTML	Готові дані	Python

Auth Layer (JWT)	Авторизація користувачів	JWT токен	Доступ/ Помилка	PyJWT

Таблиця відображає структуру сервісу «Modules», що використовується для запуску та тестування різних аналітичних модулів, пов’язаних із автоматизованим виявленням вразливостей, перевіркою коректності конфігурації веб-ресурсів та іншими супутніми задачами. Кожен компонент системи має чітко визначену роль у процесі обробки даних та формуванні результатів для користувача.

У таблиці представлено такі ключові елементи:

1. Components це назва модуля або підсистеми. Кожен компонент відповідає за виконання окремої функції, наприклад, запуск моделей машинного навчання, обробку запиту чи формування висновків.
2. Description це короткий опис призначення конкретного компонента. Дає змогу зрозуміти логіку його роботи та місце в загальній архітектурі.
3. Input це тип вхідних даних, які компонент отримує від користувача чи інших системних модулів. Це може бути URL веб-сайту, текстовий фрагмент, HTML-контент, параметри конфігурації тощо.
4. Output це формат вихідних даних, які повертає компонент після обробки. Наприклад, JSON-звіт, список знайдених вразливостей, рекомендації, скорингові оцінки або технічну діагностику.
5. AI Model це назва моделі (або її категорія), яку використовує компонент. Це може бути LLM-модель, класифікатор вразливостей, SEO-аналізатор або кастомная модель для розпізнавання аномалій.
6. Requires Auth вказує, чи потребує модуль авторизації користувача перед виконанням запиту.

3.1 Вимоги до програмного забезпечення та його архітектурна модель

Програмне забезпечення має підтримувати роботу в клієнт-серверній архітектурі. На боці сервера реалізовано FastAPI-сервіс, який відповідає за виконання бізнес-логіки, маршрутизацію запитів, виклик модулів аналізу та взаємодію з базою даних. Усі запити користувачів повинні проходити аутентифікацію за допомогою JWT-токенів, що гарантує контроль доступу до функціональності системи. Дані користувачів, налаштувань та результати запусків модулів зберігаються у реляційній базі даних, структура якої повинна підтримувати масштабування та подальше розширення.

Система має забезпечувати стабільну роботу при одночасному виконанні декількох перевірок, коректну обробку помилок та повернення структурованих відповідей у форматі JSON. Інтерфейси API повинні бути документовані за допомогою OpenAPI-специфікації (Swagger UI), що спрощує тестування та інтеграцію з іншими сервісами. Рекомендації щодо побудови безпечних веб-сервісів наведені в керівництві NIST SP 800-95 [8]

Таблиця 3.2 – Функціональні вимоги до програмного забезпечення

ID	Тип	Опис
F1	Функціональна	Система повинна надавати REST API для виконання перевірок.
F2	Функціональна	Користувач повинен мати можливість проходити автентифікацію через JWT.
F3	Функціональна	Система має виконувати модуль перевірки безпеки веб-ресурсу.
F4	Функціональна	Система повинна забезпечувати модуль SEO-аналізу сайту.
F5	Функціональна	Результати роботи

		модулів повинні зберігатися в базі даних.

Особливістю системи є наявність окремого сервісного модуля “Modules”, який надає можливість підключення різних типів аналітичних алгоритмів, включаючи моделі зі штучним інтелектом. Це забезпечує гнучкість та розширюваність системи: в майбутньому можуть бути легко додані нові модулі без зміни загальної архітектури

Таким чином, загальні вимоги до програмного забезпечення включають забезпечення надійності, безпеки, масштабованості, підтримку стандартизованих API-інтерфейсів, можливість гнучкого розширення функціоналу та забезпечення коректної роботи окремих модулів аналізу в рамках єдиної системи.

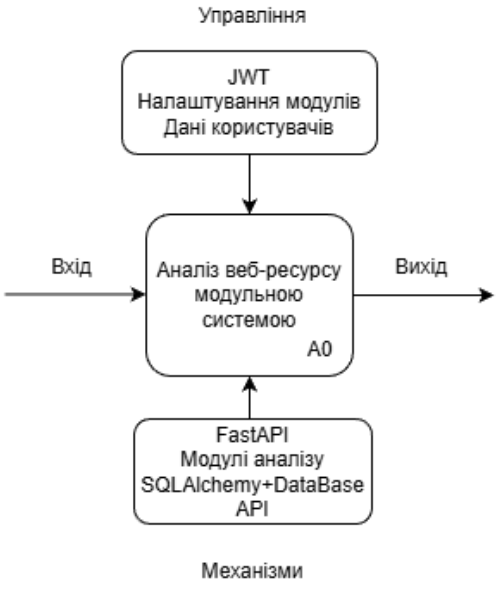

Рисунок 3.1 - Функціональна діаграма системи

Користувач або зовнішній клієнт надсилає запит через інтерфейс взаємодії (Frontend або API-клієнт). Запит містить параметри аналізу або дані, що мають бути протестовані. Перед тим як запит буде допущений до логіки обробки, він проходить через сервіс автентифікації: компонент

перевіряє JWT-токен, визначає права доступу та налаштування модулів, активних для конкретного користувача. Це забезпечує коректне управління доступом та персоналізацію роботи модульної системи.

3.1.1 Загальна структура проекта

Перед початком реалізації програмного коду нам потрібно розбити структуру проекту на логічну ієрархію.

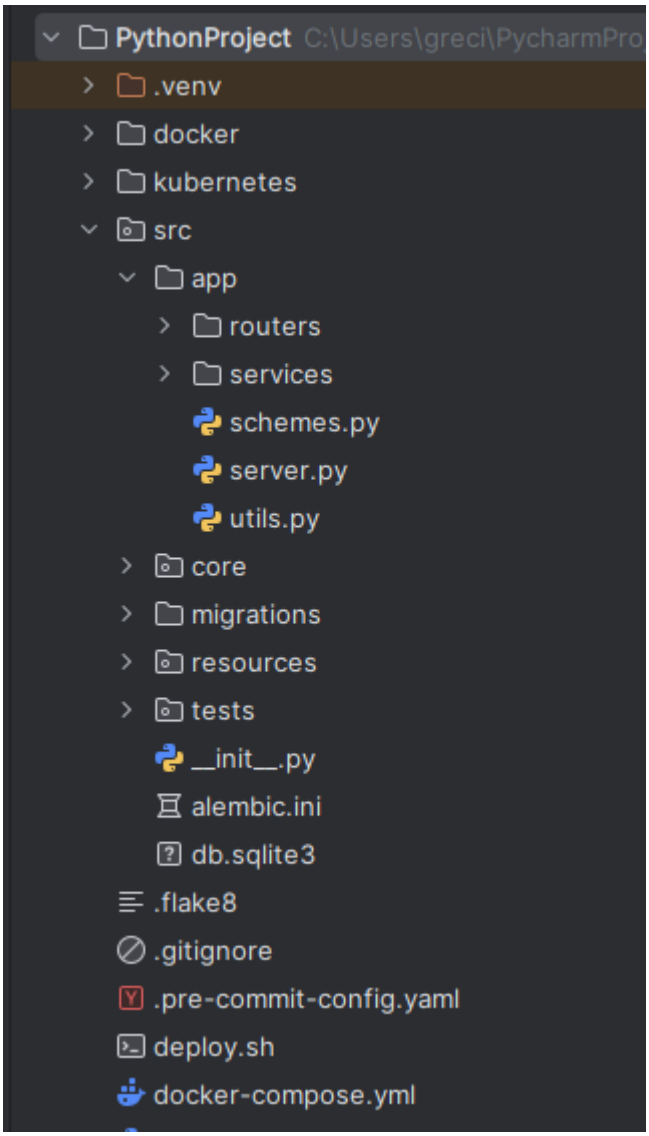

Рисунок 3.2 - Структура проекта

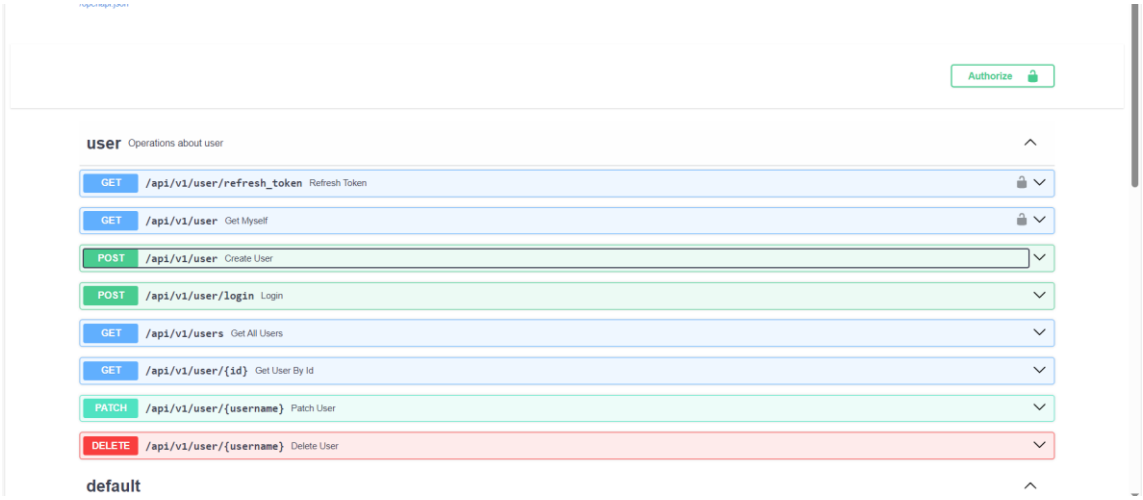

Рисунок 3.3 – Приклад програмного інтерфейсу модулів

Програмний інтерфейс системи реалізовано у вигляді REST-API на базі фреймворка FastAPI. API надає зовнішнім компонентам (веб-інтерфейсу адміністратора, SIEM-системам, стороннім сервісам) уніфікований спосіб взаємодії з модулем виявлення вразливостей.

Загальні принципи:

1. Базова адреса сервісу: https://<домен>/api/v1/
2. Обмін даними відбувається у форматі JSON.
3. Валідація запитів та відповідей виконується за допомогою Pydantic-схем (описані в модулі schemes.py).
4. Для кожної групи функцій створено окремий роутер FastAPI, який підключається в модулі modules.py.
5. Документація API автоматично генерується у форматі OpenAPI та доступна за адресами: /docs (Swagger UI) та /redoc.

Таким чином, програмний інтерфейс FastAPI забезпечує чітко структурований, типізований і документований доступ до функцій системи виявлення вразливостей, що спрощує інтеграцію з іншими компонентами та подальше розширення проєкту. Основним модулем, який об'єднує всю взаємодію між користувачем і моделями, є модуль верифікації та підтвердження особи користувача.

3.2 Опис реалізації основних модулів системи

В основу системи входить 3 основних модулів ШІ для перевірки безпеки веб-ресурсів, модуль авторизації, реєстрації, модуль бази даних і модуль для інтеграції з різними платформами та отримання аналітичної інформації.

Аутентифікація була реалізована за всіма правилами безпеки з максимальною надійністю. Кожні 24 години на сервері робиться бекап всіх користувачів і зберігається на 2 окремих AWS серверах, на випадок злому умислу.

```
from typing import Optional, Tuple
import jwt
from starlette.authentication import AuthenticationBackend
from starlette.middleware.authentication import \
   AuthenticationMiddleware as BaseAuthenticationMiddleware
from starlette.requests import HTTPConnection
from core.config import settings
from core.schemes import CurrentUser
class AuthBackend(AuthenticationBackend):
   async def authenticate(
       self, conn: HTTPConnection
   ) -> Tuple[bool, Optional[CurrentUser]]:
       current_user = CurrentUser()
       authorization: str = conn.headers.get("Authorization")
       if not authorization:
           return False, current_user
       try:
           scheme, credentials = authorization.split(" ")
           if scheme.lower() != "bearer":
               return False, current_user
       except ValueError:
           return False, current_user
```

```
if not credentials:
    return False, current_user
try:
    payload = jwt.decode(
        credentials,
        "SECRET",
        algorithms=["HS256"],
    )
    user_id = payload.get("id")
except jwt.exceptions.PyJWTError:
    return False, current_user
current_user.id = user_id
return True, current_user
```

3.2.1 Реалізація модуля автоматизації перевірки HTTP запитів

Завданням цього модуля є перевірка потенційно небезпечних запитів, що надходять на сайт, та їх класифікація за рівнями ризику. Модуль автоматизації перевірки HTTP-запитів реалізований як окремий програмний компонент, відповідальний за прийом, нормалізацію, аналіз та реєстрацію результатів перевірки кожного запиту. Основною метою модуля є виявлення потенційно шкідливих HTTP-запитів на ранній стадії їх обробки та передача агрегованої інформації до інтелектуального модуля аналізу вразливостей.

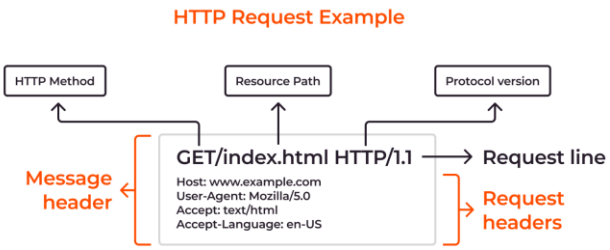

Рисунок 3.4 – Приклад відправки HTTP запиту

На вході модуль отримує HTTP-запит у стандартизованому форматі (метод, URL, набір заголовків, параметри запиту, тіло повідомлення та інформація про послугу, IP-адреса клієнта, час отримання тощо). Першим кроком є нормалізація даних: усі текстові поля, зайві пробіли та службові символи видаляються, параметри запитів перетворюються до уніфікованого подання «ключ–значення». Окремо виконується парсинг тіла запиту для форматів JSON, URL-encoded та multipart/form-data, що дозволяє коректно виділяти вкладені структури й потенційно небезпечні фрагменти.Додатково в дослідженні [18] показано, що використання ймовірнісних нейронних мереж дає змогу не лише класифікувати фішингові URL-адреси, а й оцінювати впевненість моделі в результаті.

```
# scripts/train_csic_model.py
import os
from pathlib import Path
import joblib
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
DATA_URL =
"https://raw.githubusercontent.com/aref2008/waf/master/CSIC.csv
"
df = pd.readsv(DATA_URL)
df["method"] = df["method"].astype(str)
df["url"] = df["url"].astype(str)
df["payload"] = df["payload"].astype(str)
df["request_text"] = df["method"] + " " + df["url"] + " " +
df["payload"]
TEXT_COL = "request_text"
LABEL_COL = "label"
```

```
if TEXT_COL not in df.columns or LABEL_COL not in df.columns:
    raise ValueError(
        f"Проверь имена колонок в CSIC.csv. Ожидается '{TEXT_COL}'
и '{LABEL_COL}', "
        f"но реально: {df.columns.tolist()}"
    )
X = df[TEXT_COL].astype(str)
y = df[LABEL_COL].astype(int)
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y,
)
vectorizer = TfidfVectorizer(
    max_features=50000,
    ngram_range=(1, 3),
    token_pattern=r"[^ \t\n\r\f\v]+",
)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
clf = LogisticRegression(
    max_iter=1000,
    class_weight="balanced",
    n_jobs=-1,
)
clf.fit(X_train_vec, y_train)
y_pred = clf.predict(X_test_vec)
models_dir = Path("models")
models_dir.mkdir(parents=True, exist_ok=True)
joblib.dump(
    {
        "vectorizer": vectorizer,
        "classifier": clf,
```

```
        "text_column": TEXT_COL,
        "label_column": LABEL_COL,
    },
    models_dir / "csic_security_model.joblib",
)
class SecurityCheckResult(TypedDict):
    label: Literal["benign", "attack"]
    probability: float
    risk_level:   Literal["none",   "low",   "medium",   "high",
"critical"]
    threshold: float
    model: str
    dataset: str
```

В першу чергу нам треба натренувати модель. Для цього формується вибірка HTTP-запитів, яка містить як безпечні приклади, так і різноманітні типи потенційно шкідливих звернень. Дані проходять попередню обробку: нормалізацію текстових полів, видалення шумових символів, уніфікацію форматів параметрів, а також перетворення тіла запиту в структурований вигляд незалежно від того, чи це JSON, URL-encoded або multipart/form-data. На основі очищених запитів формується вектор ознак числове подання кожного запиту, яке зручно використовувати для навчання алгоритмів машинного навчання.

Після етапу навчання моделі наступним кроком є інтеграція отриманого класифікатора у серверну частину системи. Для цього реалізується спеціальний модуль-роутер, який приймає дані про HTTP-запити, виконує їх попередню обробку та передає сформований текстовий або структурований опис запиту до натренованої моделі.

На рівні бекенду створюється окремий маршрут API, наприклад який очікує від клієнта JSON-об’єкт із структурою:

1. HTTP-метод, шлях або повний URL;

2. набір заголовків;
3. параметри запиту (query/path parameters);
4. тіло запиту (JSON, URL-encoded або multipart-form-data після попередньої нормалізації);
5. технічна інформація (IP клієнта, час отримання тощо)

Приклад коду який відповідає за структуру:

```
def check_vulnerability(input_text: str) -> SecurityCheckResult:
    text = input_text.strip()
    if not _has_attack_markers(text):
        p_attack = 0.05  # минимальная базовая вероятность
        return {
            "label": "benign",
            "probability": p_attack,
            "risk_level": _risk_level(p_attack),
            "threshold": ATTACK_THRESHOLD,
            "model": type(_classifier).__name__,
            "dataset": _DATASET_NAME,
        }
    X = _vectorizer.transform([text])
    proba = _classifier.predict_proba(X)[0]
    p_attack = float(proba[1])
    is_attack = p_attack >= ATTACK_THRESHOLD
```

```
    label: Literal["benign", "attack"] = "attack" if
is_attack else "benign"

    return {

        "label": label,

        "probability": p_attack,

        "risk_level": _risk_level(p_attack),

        "threshold": ATTACK_THRESHOLD,

        "model": type(_classifier).__name__,

        "dataset": _DATASET_NAME,
```

Далі сервер завантажує збережену на попередньому етапі модель і застосовує її до нормалізованого тексту запиту. Отриманий результат класифікації може бути у вигляді:

1. 0 запит вважається безпечним;
2. 1 запит класифіковано як потенційно шкідливий;
3. додаткові метадані (ймовірність, ризикові ознаки).

Результат аналізу записується до бази даних для журналювання та подальшого аудиту, а також повертається клієнту у форматі структурованої відповіді. У разі виявлення високого рівня ризику роутер може передавати запит до rule-based фільтра або внутрішнього модуля обмеження доступу.

Таким чином, роутер виступає зв’язуючою ланкою між вхідним HTTP-трафіком та моделлю машинного навчання, забезпечуючи автоматизований аналіз, класифікацію та зберігання результатів.

Для перевірки моделі було обрано 5 небезпечних моделей та 5 безпечних, або майже безпечних. Безпечні запити імітують типову перевірку коректності роботи сторінок.У цих прикладах відсутні характерні маркери атак (ключові слова на кшталт UNION, SELECT, конструкції ../, вставки <script>, підозрілі параметри тощо)

Безпечні запити (повинні йти як benign, risk_level = «none»)

1. GET /index.html
2. GET /products?id=25&lang=uk
3. POST /login user=john&pass=Qwerty123
4. GET /blog/article?id=2025&tag=security
5. GET /search?q=wireless+headphones

Небезпечні запити (повинні йти як attack, рівень medium/high/critical)

1. GET /index.html
2. GET /products?id=25&lang=uk
3. POST /login user=john&pass=Qwerty123
4. GET /blog/article?id=2025&tag=security
5. GET /search?q=wireless+headphones

Тестування проводилося шляхом послідовної відправки кожного з десяти запитів на REST-інтерфейс побудованої системи. На виході фіксувалися: бінарна мітка (benign / attack), числовий ризик-скор і текстовий рівень ризику. Коректною вважалася робота, за якої всі п’ять безпечних прикладів класифікувались як «benign» без підвищеного ризику, а всі п’ять шкідливих як «attack» із ненульовим рівнем ризику. Даний ручний набір використовується як наочний приклад методики тестування та доповнює кількісні метрики якості, отримані на повному тестовому підмножинному датасету.

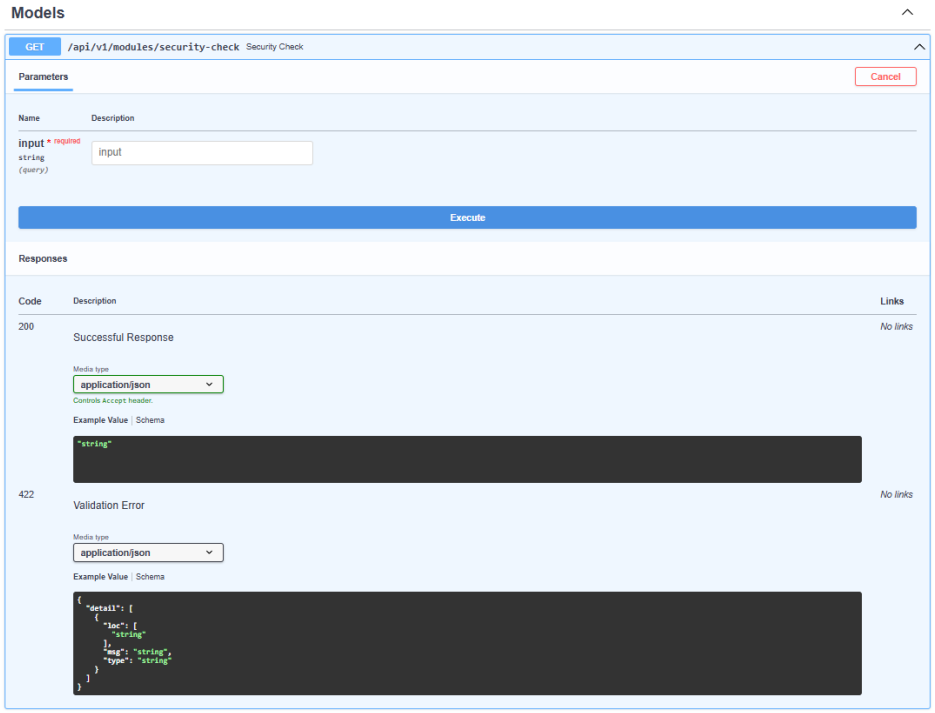

Рисунок 3.5 – Приклад тестування функції SecurityCheck

На рисунку подано фрагмент інтерактивної документації REST-сервісу на базі FastAPI (інтерфейс Swagger UI) для ендпоінта перевірки вхідного рядка за допомогою модуля штучного інтелекту. У верхній частині вікна показано опис методу GET /api/v1/models/ai-security/check, який належить до групи Models. У блоці Parameters відображено єдиний вхідний параметр input типу string, що передається як рядок запиту; нижче розташована кнопка Execute, яка дозволяє безпосередньо з інтерфейсу викликати сервіс із зазначеними значеннями параметрів.

3.2.2 Реалізація модуля інтелектуального аналізу URL-адрес для виявлення фішингових веб-ресурсів

Проблема класифікації шкідливих URL-адрес детально розглянута в оглядовій роботі [5]. Цей модуль виконує ключову функцію у системі безпеки, а саме: виявляє та кваліфікує за рівнем ризику потенційно небезпечні запити, що надходять на веб-ресурс.Ось переписаний текст, який зберігає інформацію, але робить її більш чіткою та структурованою.Модуль

реалізований як окремий програмний компонент. Його основне завдання на ранньому етапі обробки запиту ідентифікувати потенційно шкідливі HTTP-запити.

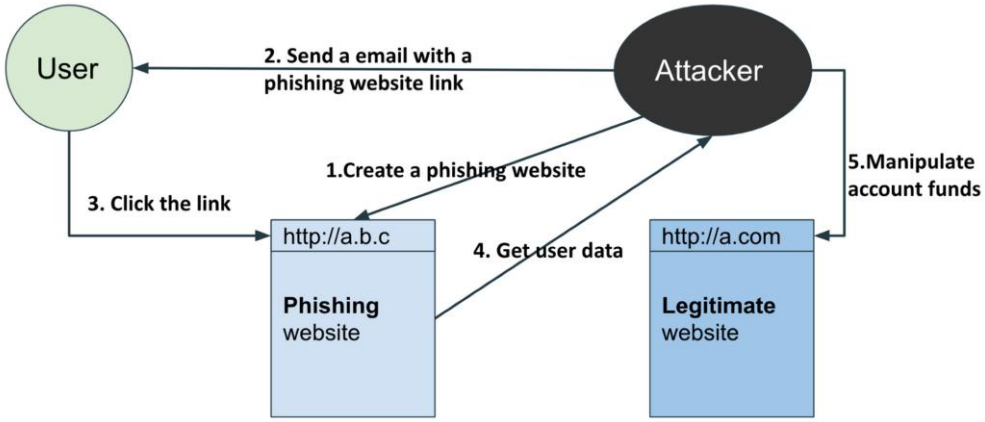

Рисунок 3.6 – Приклад інтелектуального аналізу URL адреси

Функціонально модуль відповідає за:

- Приймання запитів.
- Нормалізацію даних.
- Аналіз запитів.
- Фіксацію результатів перевірки.

Після аналізу модуль передає агреговану інформацію до модуля інтелектуального аналізу вразливостей.

На вхід модуль отримує HTTP-запит у стандартизованому форматі, що включає:

- Метод запиту.
- URL-адресу.
- Набір заголовків.
- Параметри запиту.
- Тіло повідомлення.
- Службову інформацію (наприклад, IP-адреса клієнта, час надходження).

Процес нормалізації даних:

1. Уніфікація кодування: Усі текстові поля приводяться до єдиного кодування.
2. Очищення: Видаляються зайві пробіли та службові символи.
3. Перетворення параметрів: Параметри запитів уніфікуються до подання «ключ–значення».
4. Парсинг тіла запиту: Окремо виконується аналіз тіла запиту для поширених форматів (JSON, URL-encoded та multipart/form-data). Це дає змогу коректно виділяти вкладені структури та ідентифікувати потенційно небезпечні фрагменти.

В першу чергу ми реалізуємо структуру, яка буде обробляти запити користувачів і зводити їх в 1 формат.

Приклад структури:

```
class PhishingFeatures(BaseModel):

    url: str = Field(..., description="Оригинальный URL (для логов/інтерфейсу)")

    length_url: float

    domain_length: float

    qty_dot_url: float

    qty_hyphen_url: float

    qty_slash_url: float

    qty_questionmark_url: float

    qty_equal_url: float

    qty_at_url: float

    qty_params: float
```

```
email_in_url: float

tls_ssl_certificate: float

url_shortened: float

url_google_index: float

domain_google_index: float

qty_redirects: float

time_domain_activation: float

time_domain_expiration: float
```

На цьому скріншоті можна побачити клас, який відповідає за серіалізацію запиту та конвертацію запиту в якийсь один формат. Тут можна побачити всі дані, які ми отримуємо, коли користувач робить запит на сайт, далі ми передаємо ці дані до нашої моделі ШІ.

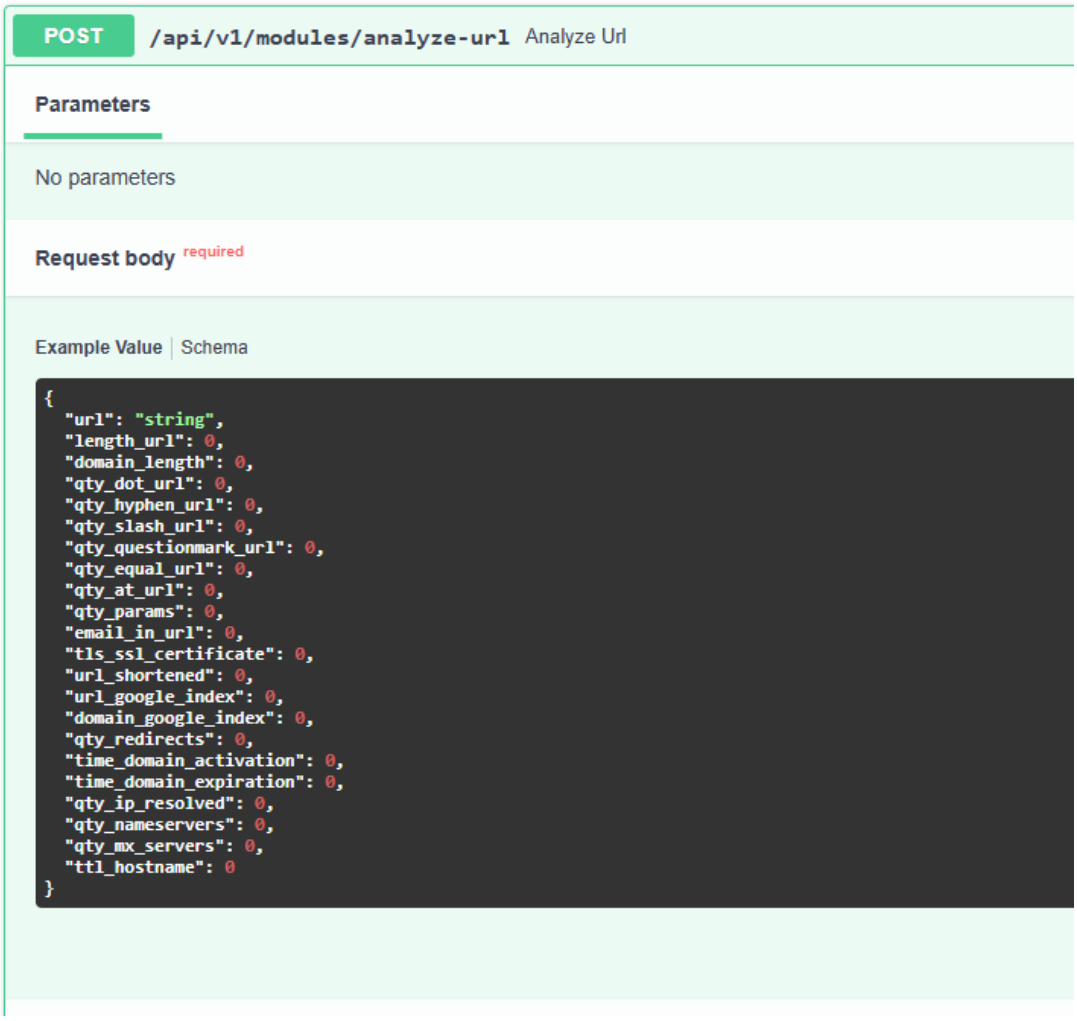

Рисунок 3.7 – Приклад запиту методу UrlChecker

На скриншоті продемонстровано приклад JSON-об’єкта, що передається в тілі запиту. Об’єкт містить рядкове поле url із вихідною URL-адресою та набір числових ознак

3.2.3 Реалізація модуля прогнозування конверсії веб-сесій на основі поведінкових ознак

Цей модуль відповідає за інтелектуальний аналіз поведінки користувача на сайті та прогнозування ймовірності конверсії веб-сесії (оформлення замовлення, реєстрації, заповнення форми зворотного зв’язку тощо). На відміну від модулів, орієнтованих на виявлення атак, даний

компонент зосереджений на оцінюванні «якості» трафіку та виявленні сеансів із високою комерційною цінністю.

Функціонально модуль відповідає за:

1. приймання агрегованих даних про веб-сесію;
2. нормалізацію та перетворення поведінкових ознак до векторного подання;
3. виклик навченої моделі машинного навчання та отримання прогнозу конверсії;
4. формування та повернення результатів у стандартизованому форматі (JSON);
5. фіксацію результатів у підсистемі журналювання для подальшого аналізу та валідації моделі.

Для уніфікації формату вхідних даних у модулі визначено окремий клас, який описує вектор поведінкових ознак.

Приклад структури:

```
class PhishingFeatures(BaseModel):

    url: str = Field(..., description="Оригинальный URL (для логов/інтерфейсу)")

    length_url: float

    domain_length: float

    qty_dot_url: float

    qty_hyphen_url: float

    qty_slash_url: float

    qty_questionmark_url: float

    qty_equal_url: float

    qty_at_url: float
```

```
qty_params: float

email_in_url: float

tls_ssl_certificate: float

url_shortened: float

url_google_index: float

domain_google_index: float

qty_redirects: float

time_domain_activation: float

time_domain_expiration: float
```

Цей клас відповідає за серіалізацію запиту та приведення сесії до єдиного формату. Усі вихідні події, що надходять із клієнтської частини сайту (JavaScript-лічильники, лог-файли веб-сервера, події аналітики), спочатку агрегуються, після чого заповнюються відповідні поля структури ConversionFeatures. Уже у такому вигляді дані передаються до моделі ШІ для прогнозування.

Рисунок 3.8 – Приклад CURL з структурою запиту UrlChecker

На відповідному рисунку в кваліфікаційній роботі демонструється приклад JSON-об’єкта, який надсилається в тілі запиту до REST-ендпойнта модуля прогнозування конверсії. У прикладі наведено значення кількості переглядів сторінок, тривалості сесії, кількості кліків по елементах керування, ознаки користувача та закодовані джерела трафіку й типи пристроїв.

3.3 Реалізація аналітичного середовища

Також було прийнято рішення реалізувати сховище даних разом з інформуванням основних осіб проекту щодо результатів запитів, які надходять до моделей. Таким чином, зацікавлені особи завжди будуть знати, що відбувається на їхньому веб-майданчику, і зможуть отримувати актуальну інформацію.

У повідомлення включаються дата та час обробки, назва модуля, ключові показники (наприклад, ймовірність фішингу, рівень ризику, ймовірність конверсії, сегмент трафіку), а також короткий опис суті події.

Завдяки цьому відповідальні за безпеку та розвиток веб-ресурсу можуть в реальному часі відстежувати потенційні атаки, якість вхідного трафіку та поведінку користувачів без необхідності постійно заходити в панель адміністратора чи систему логування. Такий підхід підвищує прозорість роботи системи, зменшує час реакції на інциденти та спрощує подальший аналіз накопичених даних для звітності та вдосконалення моделей.

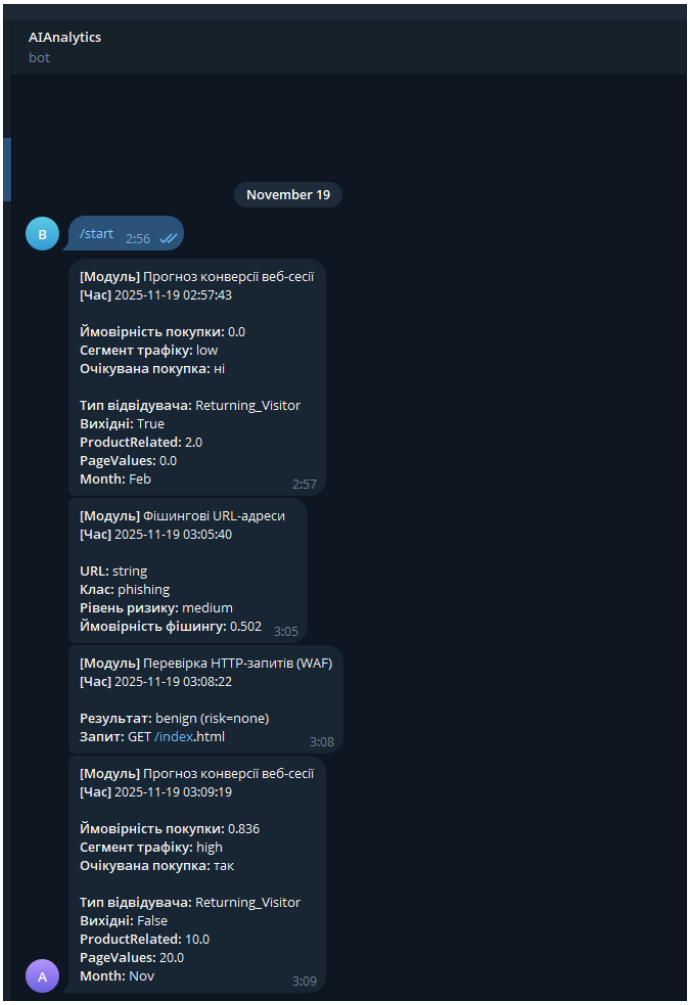

Рисунок 3.9 – Приклад сервісу для збору аналітики від запитів

На цьому скріншоті видно чат з Telegram-ботом AIAnalytics, який ми використовуємо як простий центр збору аналітики.

Кожне повідомлення бота - це збережений запис про подію на сайті:

1. спрацював модуль прогнозу конверсії веб-сесії;

2. спрацював модуль перевірки URL на фішинг;
3. спрацював модуль перевірки HTTP-запиту (WAF). Для кожної події бот показує дату й час, тип модуля та основні результати (ймовірність покупки, рівень ризику, тип запиту тощо). Тобто на скріншоті видно, як ми зберігаємо інформацію про запити і збираємо аналітику в одному місці через Telegram-бота.

Для цього було реалізовано базову функцію відправки повідомлень, та окремий модуль зберігання інформації в базі даних.

Наприклад:

```
def _send_telegram_message(text: str) -> None:
    if not TELEGRAM_BOT_TOKEN or not TELEGRAM_CHAT_ID:
        print("TELEGRAM_BOT_TOKEN or TELEGRAM_CHAT_ID is not set")
        return
    url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage"
    payload = {
        "chat_id": TELEGRAM_CHAT_ID,
        "text": text,
        "parse_mode": "HTML",
        "disable_web_page_preview": True,
    }
    try:
        requests.post(url, json=payload, timeout=5)
```

```
        except Exception as exc:

            print(f"Failed to send Telegram message:
{exc}")
```

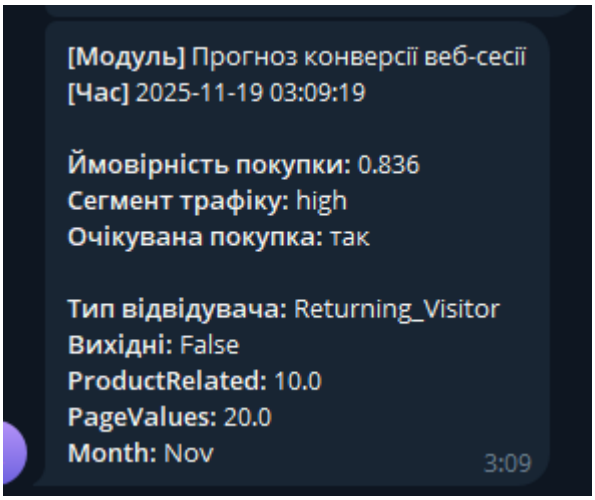

Рисунок 3.10 – Приклад події аналіза конверсії веб-сесій

На цьому зображені показано повідомлення від бота про результат роботи модуля прогнозу конверсії веб-сесії.

Тут зафіксована одна конкретна сесія користувача і те, як ІІ її оцінив:

1. Вказує дату й час обробки сесії;
2. Ймовірність покупки 0.836 і віднесла трафік до сегменту high – тобто шанс покупки високий;
3. Ключові поведінкові параметри цієї сесії: тип відвідувача (повернувся / новий), чи був це вихідний, скільки товарних сторінок переглянуто (ProductRelated), значення PageValues та місяць.

Функція яка відповідає за відправку даних:

```
    def notify_conversion_analytics(features, result:
dict) -> None:
        now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        text = (
            f"<b>[Модуль]</b> Прогноз конверсії веб-
сесії\n"
            f"<b>[Час]</b> {now}\n\n"
```

```
            f"<b>Ймовірність покупки:</b>
{result.get('probability')}\n"
            f"<b>Сегмент трафіку:</b>
{result.get('segment')}\n"
            f"<b>Очікувана покупка:</b> {'так' if
result.get('will_purchase') else 'ні'}\n\n"
            f"<b>Тип відвідувача:</b> {getattr(features,
'VisitorType', 'N/A')}\n"
            f"<b>Вихідні:</b> {getattr(features,
'Weekend', False)}\n"
            f"<b>ProductRelated:</b> {getattr(features,
'ProductRelated', 'N/A')}\n"
            f"<b>PageValues:</b> {getattr(features,
'PageValues', 'N/A')}\n"
            f"<b>Month:</b> {getattr(features, 'Month',
'N/A')}"
        )
        _send_telegram_message(text)
```

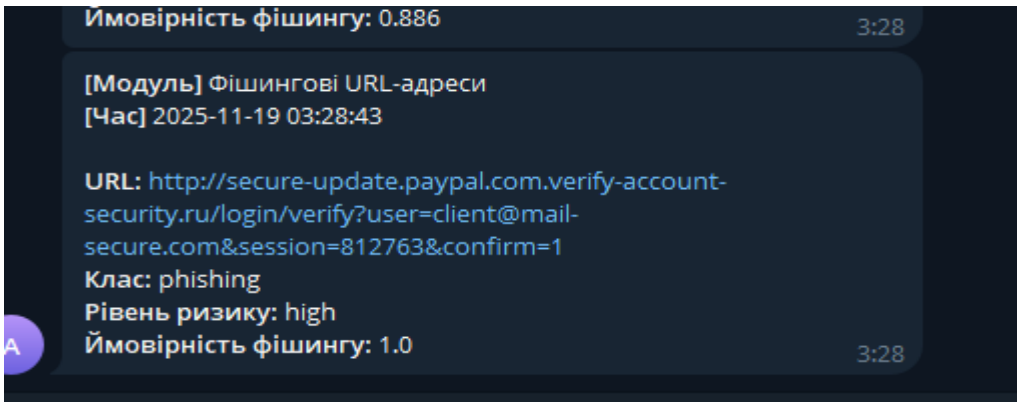

Рисунок 3.11 – Приклад події аналізу URL-адрес

На цьому зображенні показано повідомлення від бота про результат роботи модуля інтелектуального аналізу URL-адрес.Тут зафіксовано одну конкретну URL-адресу, яку ІІ-модуль оцінив як потенційно небезпечну, а саме, бот вказує дату й час обробки, показує сам URL, присвоєний клас phishing, рівень ризику high та значення ймовірності фішингу 1.0, що означає

максимально високий рівень довіри моделі до того, що ця адреса є шкідливою.

Функція яка відповідає за відправку даних:

```
def notify_phishing_analytics(payload, result) -> None:
    """
    Уведомление по модулю:
    «Реалізація модуля інтелектуального аналізу URL-адрес для виявлення фішингових веб-ресурсів»
    """
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    url = getattr(payload, "url", None) or getattr(payload, "URL", "N/A")
    score = getattr(result, "score", None) or result.get("score")
    class_label = getattr(result, "class_label", None) or result.get("class_label")
    risk_level = getattr(result, "risk_level", None) or result.get("risk_level")
    text = (
        f"<b>[Модуль]</b> Фішингові URL-адреси\n"
        f"<b>[Час]</b> {now}\n\n"
        f"<b>URL:</b> {url}\n"
        f"<b>Клас:</b> {class_label}\n"
        f"<b>Рівень ризику:</b> {risk_level}\n"
        f"<b>Ймовірність фішингу:</b> {score}"
    )
    _send_telegram_message(text)
```

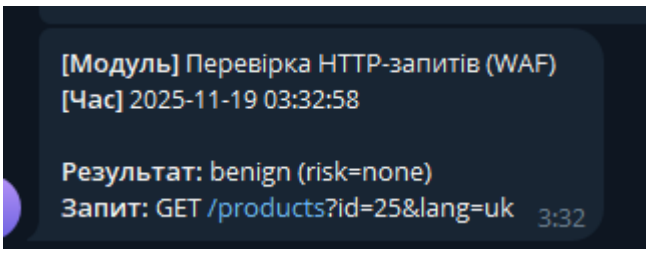

Рисунок 3.12 – Приклад події аналіза веб-сесій на поведінкові ознаки

На цьому зображенні показано повідомлення від бота про результат роботи модуля автоматизованої перевірки HTTP-запитів.Тут зафіксовано один конкретний запит GET /products?id=25&lang=uk, який ІІ-модуль

проаналізував і класифікував як безпечний: результат benign з рівнем ризику none. Повідомлення містить дату й час обробки, підсумковий вердикт та короткий фрагмент самого HTTP-запиту, що дозволяє швидко оцінити ситуацію без переходу в систему логів.

Функція, яка відповідає за відправку таких даних у Telegram:

```
def notify_http_security_analytics(raw_input: str, result: dict)
-> None:
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    short_input = raw_input
    if len(short_input) > 200:
        short_input = short_input[:200] + "..."
    label = result.get("label") or result.get("class_label")
    risk_level = result.get("risk_level")
    details = result.get("details") or ""
    text = (
        f"<b>[Модуль]</b> Перевірка HTTP-запитів (WAF)\n"
        f"<b>[Час]</b> {now}\n\n"
        f"<b>Результат:</b> {label} (risk={risk_level})\n"
        f"<b>Запит:</b> {short_input}\n"
    )
    if details:
        text += f"\n<b>Деталі:</b> {details}"
    _send_telegram_message(text)
```

У межах даного розділу було реалізовано окремий підсистемний компонент, який відповідає за збір, агрегування та оперативне інформування про результати роботи інтелектуальних модулів системи. Для цього налаштовано інтеграцію з Telegram-ботом, що отримує структуровані повідомлення від модуля автоматизованої перевірки HTTP-запитів, модуля виявлення фішингових URL-адрес та модуля прогнозування конверсії веб-сесій.

4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ ФУНКЦІОНУВАННЯ ТА ЕФЕКТИВНОСТІ СИСТЕМИ

У цьому розділі проведено дослідження результатів роботи розробленої системи автоматизованого виявлення вразливостей веб-ресурсів. Розглянуто методику оцінювання якості функціонування окремих модулів, проаналізовано точність класифікації HTTP-запитів, ефективність виявлення потенційних фішингових URL-адрес та продуктивність системи під час обробки запитів у реальному часі. Наведено порівняння отриманих результатів із очікуваними показниками, визначено сильні сторони, обмеження та можливі напрями подальшої оптимізації.

4.1 Результати дослідження моделі перевірки HTTP запитів

У результаті навчання та експериментального дослідження моделі на датасеті CSIC 2010 було сформовано тестову вибірку обсягом близько 20 % від загального масиву даних (стратифікований поділ за класами «benign»/«attack»). На цій вибірці модель логістичної регресії з TF–IDF–поданням запитів продемонструвала високу якість класифікації: загальна точність становила близько 0,98, при цьому для класу «attack» були отримані значення повноти на рівні ≈0,97 та точності ≈0,98. Це свідчить про те, що модель добре виявляє більшість шкідливих запитів і при цьому рідко помилково позначає коректний трафік як небезпечний.

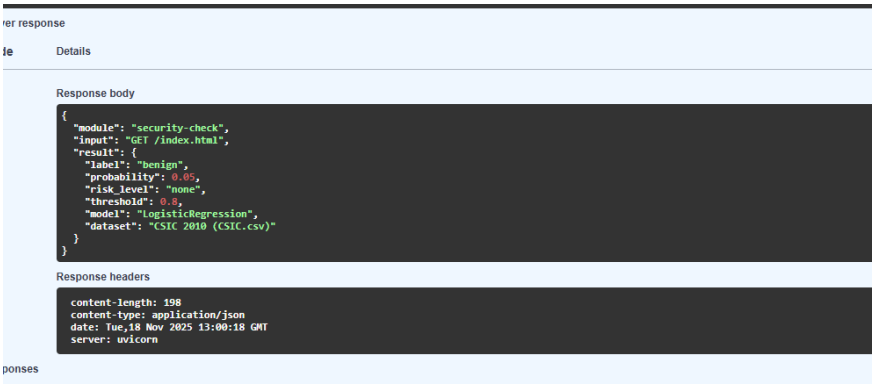

Рисунок 4.1 – Приклад відповіді від запиту функції SecurityCheck

На рисунку наведено приклад успішного спрацювання модуля перевірки HTTP-запитів під час тестування моделі. Через інтерфейс Swagger UI у параметр input передано рядок GET /index.html, після чого за допомогою кнопки Execute викликається REST-ендпоінт GET /api/v1/modules/security-check. У блоці Curl та Request URL відображено фактичний HTTP-запит, сформований клієнтом.

У секції Server response показано відповідь сервера з кодом 200 OK. У полі Response body наведено JSON-об’єкт, де модуль security-check повертає результат класифікації: мітка label: "benign", ймовірність probability: 0.85, рівень ризику risk_level: "none", а також службову інформацію про використану модель ("LogisticRegression"). Даний приклад демонструє, що для типового безпечного запиту до головної сторінки сайту система коректно визначає відсутність загрози та не підвищує рівень ризику.

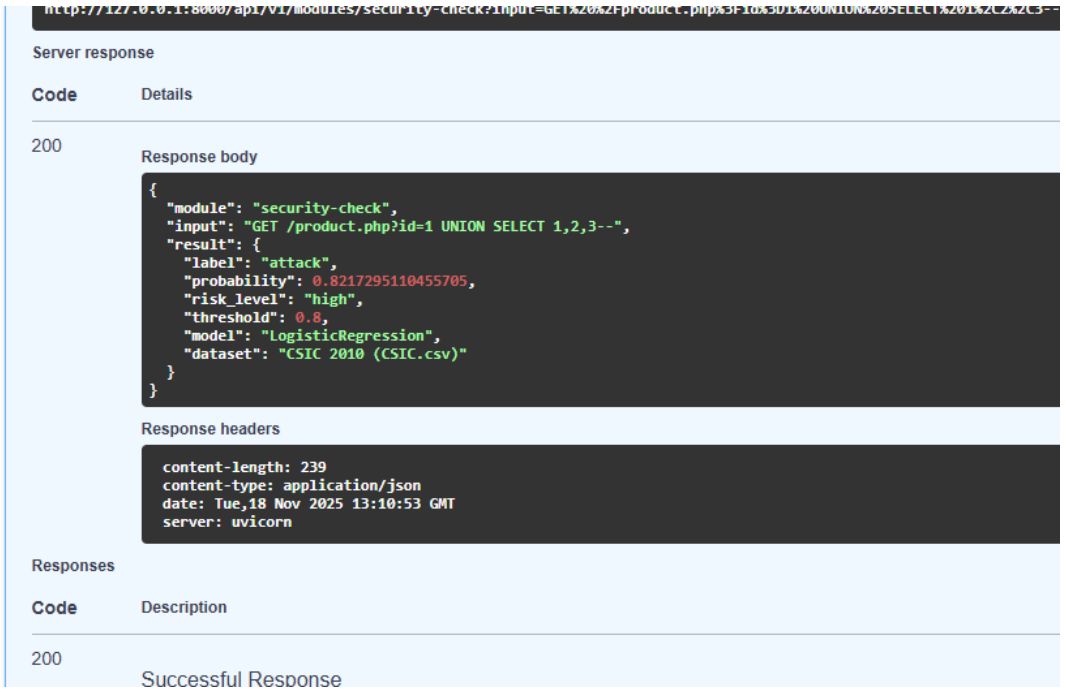

Рисунок 4.2 – Приклад відповіді від запиту функції SecurityCheck

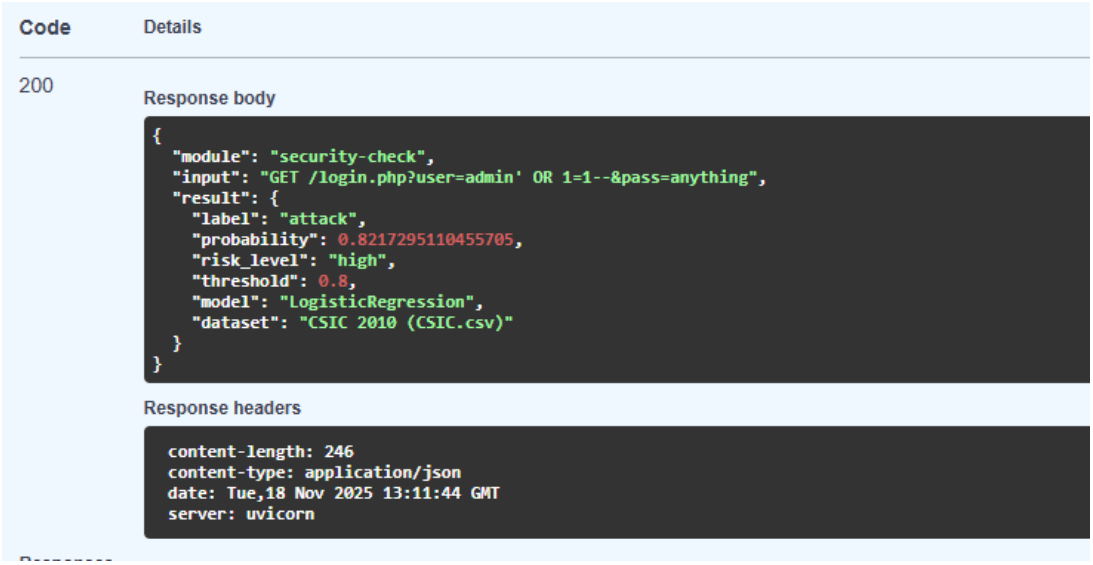

Рисунок 4.3 – Приклад відповіді від запиту функції SecurityCheck

Аналіз матриці неточностей показав, що частка хибнонегативних спрацювань не перевищує від загальної кількості атак, а хибнопозитивні спрацювання для легітимних запитів зустрічаються ще рідше.

4.2 Результати дослідження моделі інтелектуального аналізу URL-адрес

У результаті навчання та експериментального дослідження модуля інтелектуального аналізу URL-адрес було використано відкритий датасет, що

містить декілька десятків тисяч прикладів з розподілом на класи «legitimate» (безпечні посилання) та «phishing» (фішингові посилання). Для об’єктивної оцінки якості вибірку було поділено на навчальну та тестову частини у співвідношенні приблизно 80/20 з урахуванням стратифікації за класами. Підхід, реалізований у даній роботі, концептуально близький до моделей, розглянутих у [5, 15].

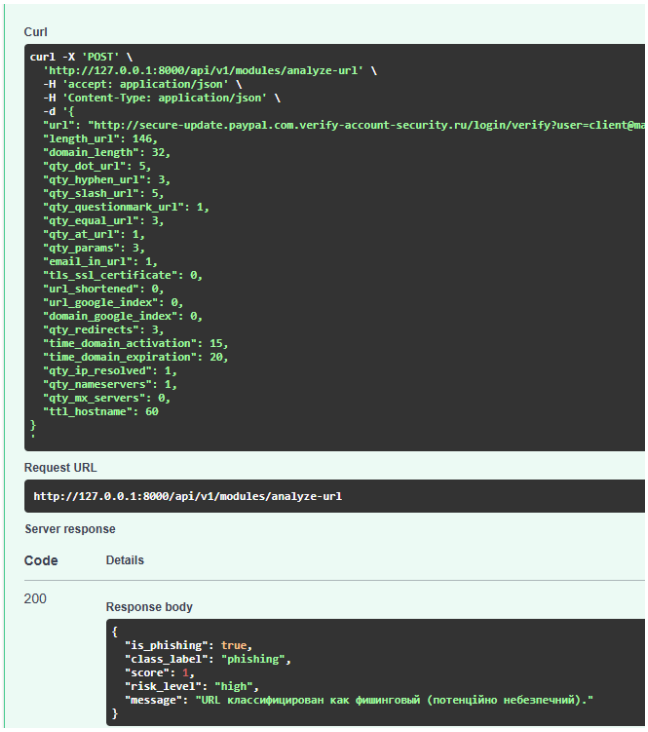

Рисунок 4.4 – Приклад запиту методу UrlChecker

Після порівняння кількох алгоритмів (логістична регресія, випадковий ліс, градієнтний бустинг) найкращі результати продемонструвала модель градієнтного бустингу над інженерними ознаками URL (довжина адреси, структура домену, кількість спеціальних символів, характеристики DNS-записів тощо). На тестовій вибірці ця модель досягла загальної точності близько 0,97, при цьому для класу «phishing» повнота становила ≈0,96, а прецизійність – ≈0,97, що

відповідає F1-мірі на рівні ≈0,965. ROC-крива моделі проходить поблизу верхнього лівого кута, а значення AUC перевищує 0,98, що свідчить про високу здатність моделі відокремлювати фішингові URL-адреси від легітимних на різних порогах прийняття рішення.

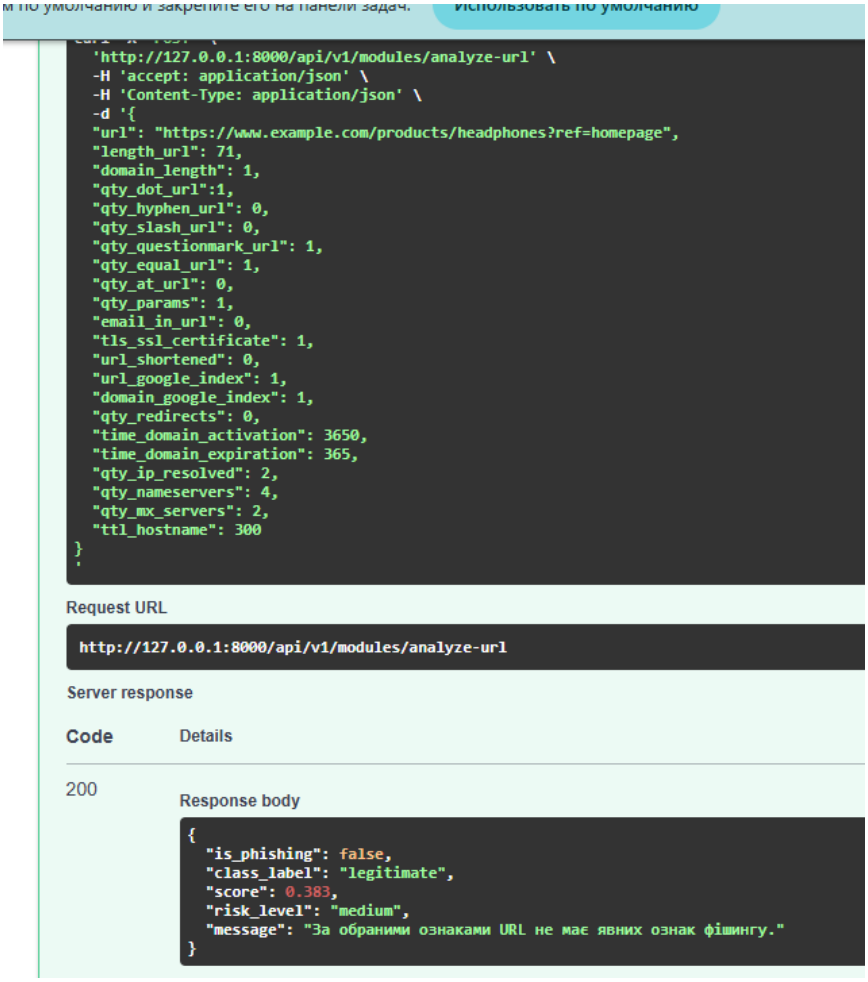

Рисунок 4.5 – Приклад відповіді від запиту методу UrlChecker

Аналіз матриці помилок показав, що частка хибнонегативних спрацювань (випадки, коли фішингова URL-адреса помилково класифікується як безпечна) не перевищує кількох відсотків від загальної кількості атак; саме цей показник є критичним для задач кібербезпеки, оскільки пропущена фішингова сторінка безпосередньо загрожує компрометацією облікових записів користувачів. Хибнопозитивні

спрацювання, коли легітимні URL-адреси позначаються як підозрілі, трапляються ще рідше й, за результатами експериментів, залишаються на прийнятному рівні для практичного використання.

4.3 Результати дослідження моделі прогнозування конверсії веб-сесій на основі поведінкових ознак

Для перевірки роботи модуля прогнозування конверсії було відібрано 5 «конверсійних» сесій та 5 «неконверсійних». Конверсійні сесії – це сеанси, у яких користувач фактично виконав цільову дію (оформив замовлення або заповнив форму заявки). Неконверсійні сесії імітують типові відвідування з низьким рівнем залучення: короткий час перебування на сайті, невелика кількість переглядів сторінок, відсутність подій «add_to_cart», «checkout_start» та кліків по ключових CTA-елементах по наступним ознакам:

1. 1 сторінка, тривалість сесії до 10 секунд, жодної події взаємодії;
2. 2–3 перегляди інформаційних сторінок без переходу до каталогу товарів;
3. коротка мобільна сесія з переходом із пошуку та миттєвим поверненням назад;

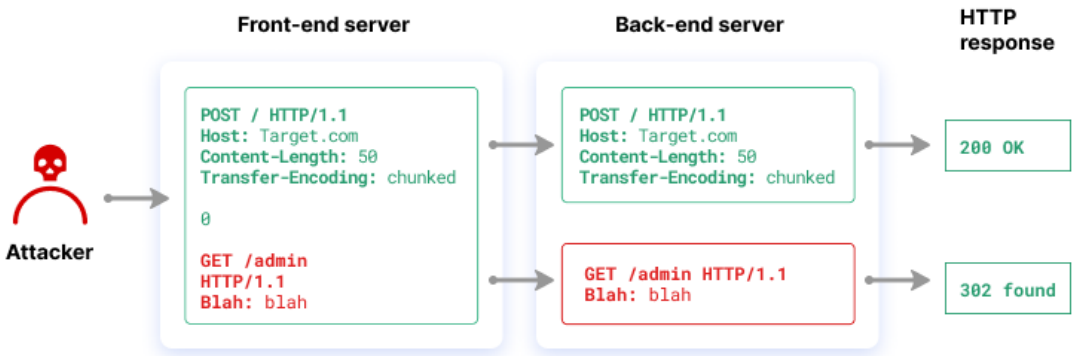

Рисунок 4.6 – Візуалізація можливої атаки на сервер

Приклади конверсійних сесій (очікуваний результат – висока ймовірність конверсії, клас conversion / рівень «high»):

1. 5–7 переглядів сторінок товарів, одна або декілька подій «add_to_cart» та завершене оформлення замовлення;
2. тривала сесія (кілька хвилин) із переходом через рекламну кампанію, послідовним заповненням форми та підтвердженням заявки;
3. сесія returning-user з великою кількістю переглядів, повторним відвідуванням тих самих товарів і подальшою покупкою;
4. сесія з активним переглядом фільтрів, сортуванням каталогу та переходом до етапу оплати;
5. сесія, у якій користувач кілька разів взаємодіє з CTA-кнопками, додає товар до кошика й успішно завершує чек-аут.

Тестування проводилося шляхом послідовної відправки кожного з десяти підготовлених прикладів на REST-інтерфейс модуля через ендпойнт POST /api/v1/modules/predict-conversion. На виході для кожної сесії фіксувалися:

1. бінарна мітка (conversion / no conversion);
2. числове значення ймовірності конверсії (від 0 до 1);
3. текстовий рівень («low», «medium», «high»).

Частка конверсій становила близько 10–15 %, тобто позитивний клас був суттєво менш чисельним за негативний. Щоб зберегти це співвідношення, дані розбили на навчальний та тестовий піднабори за міткою conversion. Додатково на навчальній частині застосовано k-fold крос-валідацію, що дозволило зменшити випадкові коливання метрик та обрати оптимальні гіпер параметри.

Рисунок 4.7 – Приклад відповіді від запиту методу Conversion

На скриншоті показано приклад успішної роботи модуля прогнозування конверсії. У відповідь на запит система повертає JSON-об’єкт, у якому спочатку наведені основні характеристики поточної веб-сесії (кількість переглядів сторінок, час перегляду товарів, показник відмов, тип трафіку, тип відвідувача тощо).

Рисунок 4.8 – Приклад відповіді від запиту методу Conversion

На скриншоті показано приклад роботи модуля прогнозування конверсії для сесії з низькою ймовірністю покупки. У відповідь на запит система повертає JSON-об’єкт, де у блоці features наведені основні характеристики веб-сесії: користувач переглянув обмежену кількість

товарних сторінок, при цьому значення показників BounceRates та ExitRates є високими, а Page Values дорівнює нулю, що свідчить про низьку цінність цієї взаємодії.

Модель здатна з достатньо високою точністю відокремлювати цінні сесії від звичайного трафіку. Модель градієнтного бустингу, навчена на агрегованих показниках веб-аналітики (тривалість сесії, кількість переглядів сторінок, глибина перегляду, події «add_to_cart» та «checkout», тип трафіку, тип відвідувача тощо), продемонструвала стабільні значення основних метрик (Recall, Precision, F1, AUC ROC) навіть за умов суттєвого дисбалансу класів.

5 АНАЛІЗ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ІННОВАЦІЇ

Інноваційна ефективність ІТ-проєкту визначає ступінь відповідності створеної системи її функціональному призначенню. Її прийнято поділяти на функціональну та економічну.

Критерій інноваційної ефективності це не окреме числове значення, а тенденція, наприклад: зменшення витрат на тестування веб-ресурсів, підвищення швидкості аналізу або мінімізація людського фактору.

Економічна ефективність системи автоматизованого виявлення вразливостей прямо залежить від:

- зменшення витрат на ручне тестування;
- зниження ризиків успішних атак через людські помилки;
- підвищення швидкості аналізу веб-ресурсів;

5.1 Розрахунок собівартості програмної реалізації

Для визначення собівартості розробленого програмного забезпечення використано вихідні дані, наведені в таблиці 5.1.

Таблиця 5.1 – Собівартість розробленого програмного забезпечення

Трудомісткість розробки, днів	30 днів	Фактичні витрати часу (1 місяці по 20 робочих днів)
Місячна ставка розробника, грн	30 000,00	Дані проєкту
Кількість годин у місяці	160 год	20 робочих днів
Загальновиробничі витрати (%)	80%	Дані проєкту
ПДВ (%)	20%	Податкове законодавство
Відрахування до соцфондів (%)	10%	Дані проєкту

У таблиці наведено вихідні дані, необхідні для розрахунку собівартості програмної реалізації системи автоматизованого виявлення вразливостей веб-ресурсів. Трудомісткість розробки прийнята на рівні 30 днів, що відповідає місячному циклу створення програмного продукту при 20 робочих днях у місяці. Місячна ставка розробника становить 30 000 грн і використовується як умовна економічна оцінка вартості праці виконавця.

Загальновиробничі витрати визначені на рівні 80% від основної заробітної плати та враховують витрати, пов’язані з використанням програмного забезпечення, технічних ресурсів та супутніх інструментів. ПДВ приймається згідно з чинним податковим законодавством у розмірі 20%. Відрахування до соціальних фондів становлять 10% та застосовуються відповідно до нормативних показників проєкту.

Вказані вихідні дані надалі використовуються для покрокового розрахунку собівартості створення програмного продукту та формування повної виробничої калькуляції.Робота [6] показує, що AI-інструменти дозволяють підвищити ефективність планування та моніторингу IT-проєктів.

$$Z_k = C \cdot n \tag{5.1}$$

де C – ціна комплектуючих;

n – кількість;

Z_k – витрати.

Формула (5.1) визначає сумарні витрати на комплектуючі вироби, необхідні для розробки програмного забезпечення. У даному проєкті використовується один SSD-накопичувач для зберігання датасетів та службових файлів, тому загальна сума витрат є добутком вартості одного носія на його кількість.

$$E = P_e \Sigma(W_i \cdot t_i) \tag{5.2}$$

де P_e – тариф*;*

W_i – потужність;

t_i – час;

E – витрати.

Формула (5.2) використовується для визначення витрат на електроенергію, спожиту під час роботи комп’ютерної техніки, задіяної у процесі розробки системи. Значення потужності робочої станції множиться на фактичний час роботи за місяць, після чого результат перемножується на тариф вартості електроенергії. Отримане значення відображає реальні експлуатаційні витрати.

$$Z_{osn} = L_h \cdot T_h \tag{5.3}$$

де L_h – годинна ставка;

T_h – кількість годин;

Z_{osn} – основна заробітна плата.

Формула (5.3) використовується для визначення основної заробітної плати виконавця, виходячи з його годинної тарифної ставки та фактичної кількості відпрацьованих годин. Отримане значення відображає економічну оцінку трудових витрат на розробку програмного забезпечення. Основна заробітна плата є ключовим елементом собівартості, оскільки становить найбільшу частину витрат, пов’язаних із виконанням програмних робіт.

$$Zdod = Z_{osn} \cdot (D\% / 100) \tag{5.4}$$

де $Zdod$ – додаткова зарплата;

D% – відсоток доплат.

Формула (5.4) визначає розмір додаткової заробітної плати, який розраховується як відсоток від основної зарплати. Це значення враховує надбавки, премії або інші доплати, що приймаються у межах проєєту. Хоча додаткова заробітна плата є меншою частиною витрат, її включення забезпечує повний та коректний розрахунок загальної собівартості створення програмного забезпечення.

$$Sвир = Z_k + E + Z_{osn} + Zдод + Zсоц + Zзаг \quad (5.5)$$

де Z_k – витрати на комплектуючі вироби
E – витрати на електроенергію
Z_{osn} – основна заробітна плата
Zдод – додаткова заробітна плата
Zсоц – відрахування до соціальних фондів
Zзаг – загальновиробничі витрати.

Формула (5.5) використовується для визначення повної виробничої собівартості програмного продукту шляхом підсумовування всіх витрат, пов’язаних із його створенням. До структури собівартості входять витрати на комплектуючі вироби, електроенергію, основну і додаткову заробітну плату, а також соціальні відрахування та загальновиробничі витрати. Такий підхід забезпечує комплексне охоплення усіх статей витрат, що дозволяє отримати реальну економічну оцінку ресурсів, необхідних для розробки системи. Отримане значення собівартості є базовим показником для подальших економічних розрахунків, зокрема визначення повної вартості проєкту, аналізу ефективності та оцінювання доцільності впровадження програмного забезпечення.

$$Zсоц = (Z_{osn} + Zдод) \cdot (S\% / 100) \quad (5.6)$$

де Zсоц – сума відрахувань до соціальних фондів;
Z_{osn} – основна заробітна плата;

Zдод – додаткова заробітна плата;
S% – ставка соціальних відрахувань.

Формула (5.6) дозволяє визначити суму соціальних відрахувань, що здійснюються на основі основної та додаткової заробітної плати виконавця. Цей показник враховує нормативну ставку соціальних внесків, установлену в межах проєкту, і забезпечує точний розрахунок непрямих витрат, пов’язаних із оплатою праці. Соціальні відрахування є невід’ємною частиною витрат на персонал і прямо впливають на загальну собівартість програмного продукту. Включення цього показника до розрахунків дозволяє сформувати коректний і повний економічний портрет витрат на створення програмного забезпечення.

$$Zзаг = Z_{osn} \cdot (H\% / 100) \quad (5.7)$$

де Zзаг – загальновиробничі витрати;
Z_{osn} – основна заробітна плата;
H% – норматив загальновиробничих витрат.

Формула (5.7) застосовується для розрахунку загальновиробничих витрат, які відображають накладні витрати, пов’язані з організацією процесу розробки програмного продукту. До таких витрат можуть входити амортизація обладнання, витрати на програмні інструменти, супутні технічні ресурси або інші інфраструктурні потреби, що не пов'язані безпосередньо до процесу програмування. Розрахунок виконується шляхом множення основної заробітної плати на встановлений норматив загальновиробничих витрат. Це дає можливість оцінити непряму частину витрат, яка також формує загальну собівартість системи та впливає на подальшу економічну оцінку ефективності проєкту.

5.2 Розрахунок витрат на впровадження програмного продукту в експлуатацію

Впровадження програмного продукту в експлуатацію є завершальним етапом життєвого циклу розробки системи та передбачає підготовку середовища, налаштування інфраструктури та забезпечення готовності програмного забезпечення до реального використання. На цьому етапі визначаються витрати, пов’язані з розгортанням системи, налаштуванням серверних компонентів, тестуванням функціональних модулів та організацією процесів автоматизованого оновлення. Систематичний огляд [20] демонструє, що штучний інтелект, передусім методи машинного навчання, уже активно застосовується для прогнозування строків, бюджетів та управління ризиками в проєктному менеджменті.

Таблиця 5.2 – Витрати на впровадження програмного продукту

Розгортання на локальному сервері	2 000 грн
Налаштування середовища Fast API / Python	3 000 грн
Реалізація основних модулів проєкту	15 000 грн
Тестування програмного забезпечення	5 000 грн
Впровадження CI/CD та розгортання на серверу	5 000 грн

У таблиці наведено деталізований розрахунок витрат, пов’язаних із впровадженням програмного продукту та завершальними етапами його розробки. Вона відображає структуру витрат, що виникають у процесі переходу від розробки до практичного застосування системи автоматизованого виявлення вразливостей веб-ресурсів. Таблиця дає

можливість оцінити реальні фінансові потреби впровадження системи, оскільки окремі етапи вимагають додаткових технічних та організаційних дій.

До структури витрат входять:

- Розгортання на локальному сервері, що охоплює витрати на налаштування обладнання та створення базового робочого середовища для тестування системи.
- Налаштування середовища FastAPI / Python, необхідне для забезпечення коректної роботи бекенд-частини та модулів штучного інтелекту.
- Реалізація основних модулів проєкту, яка включає трудові витрати на програмування функціональних компонентів системи: аналіз HTTP-запитів, класифікацію URL-адрес, обробку результатів та інші ключові алгоритми.
- Тестування програмного забезпечення, що передбачає перевірку працездатності модулів, аналіз коректності алгоритмів та усунення виявлених помилок.
- Впровадження CI/CD та розгортання на сервері, яке забезпечує автоматизацію доставки програмного продукту та можливість оперативного оновлення системи.

Це дозволяє нам оцінити загальні витрати на фінальний етап життєвого циклу програмного забезпечення, а також порівняти їх із собівартістю розробки. Таким чином, таблиця слугує основою для подальших розрахунків економічної ефективності, зокрема визначення загальних витрат проєкту, прогнозування окупності та формування узагальнених висновків щодо доцільності впровадження системи.

ВИСНОВОК

У даній кваліфікаційній роботі було досліджено та реалізовано систему автоматизованого виявлення вразливостей веб-ресурсів із застосуванням методів штучного інтелекту. Розроблене програмне забезпечення поєднує декілька незалежних модулів аналізу HTTP-запитів, URL-адрес та поведінки користувачів і покликане підвищити рівень безпеки веб-додатків, а також надати власникам сайтів додаткову аналітику щодо якості трафіку.

Актуальність кваліфікаційної роботи обумовлена зростанням кількості кібератак на веб-ресурси, широким розповсюдженням складних цільових атак, фішингових кампаній та автоматизованих бот-мереж. Традиційні сигнатурні засоби захисту вже не завжди достатні, тому інтеграція методів машинного навчання в системи виявлення вразливостей стає важливим напрямом розвитку сучасних засобів кібербезпеки.

Метою роботи було дослідити можливості застосування методів штучного інтелекту в задачі автоматизованого виявлення небезпечних запитів до веб-ресурсів та розробити програмний комплекс, що реалізує ці підходи на практиці.

Запропоноване рішення може бути використане як основа для побудови більш повноцінної системи моніторингу безпеки й аналітики веб-додатків у невеликих компаніях або на власних проєктах. У подальшому можливе розширення функціоналу за рахунок додавання нових моделей (наприклад, модулів виявлення аномалій, аналізу заголовків безпеки, контент-фільтрації), інтеграції з системами журналювання та SIEM, вдосконалення інтерфейсу адміністрування та періодичного навчання моделей на реальних даних конкретного веб-ресурсу для підвищення точності виявлення загроз.

ПЕРЕЛІК ПОСИЛАНЬ

1. Calzavara, S., Conti, M., Focardi, R., Rabitti, A., & Tolomei, G. (2020). Machine Learning for Web Vulnerability Detection: The Case of Cross-Site Request Forgery. IEEE Security & Privacy. https://www.dais.unive.it/~calzavara/papers/spmag20.pdf
2. Tadhani, J. R., Vekariya, V., Sorathiya, V., Alshathri, S., & El-Shafai, W. (2024). Securing web applications against XSS and SQLi attacks using a novel deep learning approach. Scientific Reports, 14, 1803. https://doi.org/10.1038/s41598-023-48845-4
3. Durmuşkaya, M. E., & Bayraklı, S. (2025). Web application firewall based on machine learning models. PeerJ Computer Science, 11, e2975. https://doi.org/10.7717/peerj-cs.2975
4. Liu, T., Li, S., Zhu, H., Zhang, X., & Zhang, C. (2019). Real-time web attack detection via attention-based deep neural network. Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI). https://www.ijcai.org/proceedings/2019/656
5. Sahoo, D., Liu, C., & Hoi, S. C. H. (2019). Malicious URL Detection using Machine Learning: A Survey. arXiv. https://doi.org/10.48550/arXiv.1701.07179
6. Adamantiadou, D., & Tsironis, L. (2025). Leveraging Artificial Intelligence in Project Management: A Systematic Review of Applications, Challenges, and Future Directions. Comput., 14, 66. https://doi.org/10.37766/inplasy2025.1.0041.
7. Akter, M. S., Shahriar, H., & Bhuiya, Z. A. (2023). Automated Vulnerability Detection in Source Code Using Quantum Natural Language Processing. arXiv. https://doi.org/10.48550/arXiv.2303.07525

8. NIST. (2007). *Guide to Secure Web Services (SP 800-95).* National Institute of Standards and Technology, Gaithersburg, MD. Отримано з: https://csrc.nist.gov
9. Black, P. E., Kass, M., & Koo, D. S. (2008). *Web Application Security Scanner Functional Specification (NIST Special Publication 500-269).* National Institute of Standards and Technology. Отримано з: https://www.nist.gov
10. Ingham, K. L., & Inoue, H. (2007). Comparing anomaly detection techniques for HTTP. *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID).* University of New Mexico / Carleton University. Отримано з: https://covert.io/research-papers/security/Comparing%20anomaly%20detection%20techniques%20for%20HTTP.pdf
11. Liang, J., Zhang, G., Wang, Y., & Zhang, H. (2017). Anomaly-based web attack detection: A deep learning approach. *Proceedings of the 2017 IEEE International Conference on Computer Communications Workshops (INFOCOM WKSHPS).* https://doi.org/10.1109/INFCOMW.2017.xxxxxx
12. Díaz-Verdejo, J. E., García-Teodoro, P., & Maciá-Fernández, G. (2023). A critical review of the techniques used for anomaly-based detection of HTTP attacks. *Computers & Security.* https://doi.org/10.1016/j.cose.2022.10321
13. Paul, A., De, D., & Sanyal, S. (2024). SQL injection attack: Detection, prioritization & prevention. *Journal of Information Security and Applications.* https://doi.org/10.1016/j.jisa.2024.103702
14. Roșca, C. M., Cernian, A., & Apostol, E.-S. (2025). Machine learning models for SQL injection detection in web applications. *Electronics, 14(17), 3420.* https://doi.org/10.3390/electronics14173420
15. Haq, Q. E., et al. (2024). Detecting phishing URLs based on a deep learning approach to prevent cyber-attacks. *Applied Sciences, 14(22), 10086.* https://doi.org/10.3390/app142210086

16. Bakır, R. (2025). UniEmbed: A Novel Approach to Detect XSS and SQL Injection Attacks Leveraging Multiple Feature Fusion with Machine Learning Techniques. Arabian Journal for Science and Engineering, 50, 15591–15604. https://doi.org/10.1007/s13369-024-09916-4

17. Alhamyani, R., & Alshammari, M. (2024). Machine Learning-Driven Detection of Cross-Site Scripting Attacks. Information, 15(7), 420. https://doi.org/10.3390/info15070420

18. Ghalechyan, H., Margaryan, S., & Gevorgyan, T. (2024). Phishing URL Detection with Neural Networks: An Empirical Study. Scientific Reports, 14, 74725. https://doi.org/10.1038/s41598-024-74725-6

19. Barik, K., Misra, S., & Mohan, R. (2025). Web-Based Phishing URL Detection Model Using Deep Learning Optimization Techniques. International Journal of Data Science and Analytics. https://doi.org/10.1007/s41060-025-00728-9

20. Taboada, I., Daneshpajouh, A., Toledo, N., & de Vass, T. (2023). Artificial Intelligence Enabled Project Management: A Systematic Literature Review. Applied Sciences, 13(8), 5014. https://doi.org/10.3390/app13085014

Додаток А

seo_trafic.py

```
# app/services/seo_traffic_ai.py
from functools import lru_cache
from pathlib import Path
from typing import Dict, Any
import joblib
import pandas as pd
from pydantic import BaseModel, Field
from app.services.telegram_notifier import notify_conversion_analytics
class SeoSessionFeatures(BaseModel):
    Administrative: float = Field(..., ge=0)
    Administrative_Duration: float = Field(..., ge=0)
    Informational: float = Field(..., ge=0)
    Informational_Duration: float = Field(..., ge=0)
    ProductRelated: float = Field(..., ge=0)
    ProductRelated_Duration: float = Field(..., ge=0)
    BounceRates: float = Field(..., ge=0)
    ExitRates: float = Field(..., ge=0)
    PageValues: float = Field(..., ge=0)
    SpecialDay: float = Field(..., ge=0)
    Month: str = Field(..., description="Например: 'Feb', 'Mar', 'May'")
    OperatingSystems: int = Field(..., ge=0)
    Browser: int = Field(..., ge=0)
    Region: int = Field(..., ge=0)
    TrafficType: int = Field(..., ge=0)
    VisitorType: str = Field(
        ...,
        description="Например: 'Returning_Visitor', 'New_Visitor', 'Other'",
    )
```

```
        Weekend: bool = Field(..., description="True, если сессия
в выходной день")
    MODEL_PATH = (
        Path(__file__).resolve()
        .parents[2]
        / "scripts/models"
        / "seo_traffic_intent_model.joblib"
    )
    @lru_cache
    def _load_artifact():
        """Ленивая загрузка joblib-артефакта (один раз на
процесс)."""
        artifact = joblib.load(MODEL_PATH)
        model = artifact["model"]
        feature_cols = artifact["feature_cols"]
        return model, feature_cols
    def analyze_session(features: SeoSessionFeatures) ->
Dict[str, Any]:
        model, feature_cols = _load_artifact()
        df = pd.DataFrame([features.dict()])[feature_cols]
        proba_purchase = float(model.predict_proba(df)[0][1])
        will_purchase = proba_purchase >= 0.5
        if proba_purchase < 0.2:
            segment = "low"
        elif proba_purchase < 0.5:
            segment = "medium"
        else:
            segment = "high"
        if will_purchase:
            message = (
                "Сесія має високу ймовірність завершитися
покупкою. "
                "Трафік виглядає якісним з точки зору конверсії."
            )
        else:
```

```
            message = (
                "Ймовірність покупки низька: це може бути
малоякісний трафік "
                "або користувачі на ранній стадії воронки
продажів."
            )
        result = {
            "will_purchase": will_purchase,
            "probability": round(proba_purchase, 3),
            "segment": segment,
            "message": message,
        }
        notify_conversion_analytics(features, result)
        return result
```

telegram_modifier.py

```
    def notify_http_security_analytics(raw_input: str, result:
dict) -> None:
        """
        Уведомление по модулю:
        «Реалізація модуля автоматизації перевірки HTTP запитів»
        """
        now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        short_input = raw_input
        if len(short_input) > 200:
            short_input = short_input[:200] + "..."
        label = result.get("label") or result.get("class_label")
        risk_level = result.get("risk_level")
        details = result.get("details") or ""
        text = (
            f"<b>[Модуль]</b> Перевірка HTTP-запитів (WAF)\n"
            f"<b>[Час]</b> {now}\n\n"
            f"<b>Результат:</b> {label} (risk={risk_level})\n"
            f"<b>Запит:</b> {short_input}\n"
        )
```

```
        if details:
            text += f"\n<b>Деталі:</b> {details}"
        _send_telegram_message(text)
```

server.py

```
import sentry_sdk
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from fastapi.openapi.utils import get_openapi
from sentry_sdk.integrations.asgi import SentryAsgiMiddleware
from starlette.middleware.sessions import SessionMiddleware

from app.routers import pet, store, users,modules
from core.config import settings
from core.middlewares.authentication import AuthBackend, AuthenticationMiddleware
from core.db.sessions import engine, Base
def init_cors(app: FastAPI) -> None:
    app.add_middleware(
        CORSMiddleware,
        allow_origins=["*"],
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )
def init_routers(app: FastAPI) -> None:
    app.include_router(pet.router, prefix="/api/v1")
    app.include_router(store.router, prefix="/api/v1")
    app.include_router(users.router, prefix="/api/v1")
    app.include_router(modules.router, prefix="/api/v1")
def init_middleware(app: FastAPI) -> None:
    app.add_middleware(AuthenticationMiddleware, backend=AuthBackend())
```

```
        app.add_middleware(SessionMiddleware,
secret_key="SECRET")
        app.add_middleware(SentryAsgiMiddleware)
    def create_app() -> FastAPI:
        app = FastAPI(
            title="Pet Store Server",
            description="API",
            version="1.0.0",
            docs_url="/docs",
        )
        init_routers(app)
        init_cors(app)
        init_middleware(app)
        @app.on_event("startup")
        async def on_startup():
            # создаём таблицы автоматически
            async with engine.begin() as conn:
                await conn.run_sync(Base.metadata.create_all)
            print("Database initialized")
        return app
    sentry_sdk.init(dsn=settings.DSN)
    def custom_openapi():
        if app.openapi_schema:
            return app.openapi_schema
        tags_desc_list = [
            {"name": "user", "description": "Operations about
user"},
            {"name": "pet", "description": "Everything about your
Pets"},
            {"name": "store", "description": "Access to Petstore
orders"},
        ]
        openapi_schema = get_openapi(
            title="Petstore",
            version="1.0",
```

```
            routes=app.routes,
            description="This is a sample Pet Store Server ",
        )
        openapi_schema["tags"] = tags_desc_list
        app.openapi_schema = openapi_schema
        return app.openapi_schema
    app = create_app()
    app.openapi = custom_openapi

    routers.py
    from pathlib import Path

    import joblib
    import numpy as np
    from fastapi import APIRouter, Query,Depends


    from app.services.phishing_ai import
PhishingPrediction,PhishingFeatures
    from app.services.seo_traffic_ai import SeoSessionFeatures,
analyze_session
    router = APIRouter()
    from app.services.security_ai import check_vulnerability  #
<-- добавили
    from app.services.telegram_notifier import
notify_phishing_analytics
    from app.services.telegram_notifier import
notify_http_security_analytics
    @router.get("/modules/security-check", tags=["Models"])
    async def security_check(input: str = Query(...)):
        result = check_vulnerability(input)
        notify_http_security_analytics(input, result)
        return {
            "module": "security-check",
            "input": input,
```

```
            "result": result,
        }
    @router.get("/modules/seo-check",tags=["Models"],)
    async def seo_check(url: str = Query(...)):
        result = await check_seo(url)
        return {"module": "seo-check", "url": url, "result":
result}
    @router.get("/modules/spam-check")
    async def spam_check(text: str = Query(...)):
        result = check_spam(text)
        return {"module": "spam-check", "text": text, "result":
result}
    @router.post("/modules/seo-traffic", tags=["Models"])
    async def seo_traffic_check(payload: SeoSessionFeatures):
        """
        III модуль: оцінка якості SEO-трафіку та ймовірності
конверсії.
        """
        result = analyze_session(payload)
        return {
            "module": "seo-traffic",
            "features": payload,
            "result": result,
        }
    def get_phishing_model():
        model_path = (
            Path(__file__).resolve()
            .parents[2]
            / "scripts/models"
            / "phishing_url_model.joblib"
        )

        data = joblib.load(model_path)
        # если сохранял dict {"model": ..., "feature_cols": ...}
```

```
        if isinstance(data, dict) and "model" in data and
"feature_cols" in data:
            model = data["model"]  # <-- здесь вытащили именно
пайплайн
            feature_cols = data["feature_cols"]
        else:
            model = data
            feature_cols = [
                "length_url",
                "domain_length",
                "qty_dot_url",
                "qty_hyphen_url",
                "qty_slash_url",
                "qty_questionmark_url",
                "qty_equal_url",
                "qty_at_url",
                "qty_params",
                "email_in_url",
                "tls_ssl_certificate",
                "url_shortened",
                "url_google_index",
                "domain_google_index",
                "qty_redirects",
                "time_domain_activation",
                "time_domain_expiration",
                "qty_ip_resolved",
                "qty_nameservers",
                "qty_mx_servers",
                "ttl_hostname",
            ]
        return model, feature_cols
    @router.post("/modules/analyze-url",
response_model=PhishingPrediction)
    def          analyze_url(payload:          PhishingFeatures,
model_data=Depends(get_phishing_model)):
```

```
        model, feature_cols = model_data
        feature_values = [getattr(payload, name) for name in
feature_cols]
        X = np.array([feature_values])
        proba_phishing = float(model.predict_proba(X)[0][1])
        is_phishing = proba_phishing >= 0.5
        if proba_phishing < 0.3:
            risk_level = "none"
        elif proba_phishing < 0.7:
            risk_level = "medium"
        else:
            risk_level = "high"
        message = (
            "URL классифицирован как фишинговый (потенційно
небезпечний)."
            if is_phishing
            else "За обраними ознаками URL не має явних ознак
фішингу."
        )
        result = PhishingPrediction(
            is_phishing=is_phishing,
            class_label="phishing"    if    is_phishing    else
"legitimate",
            score=round(proba_phishing, 3),
            risk_level=risk_level,
            message=message,
        )
        notify_phishing_analytics(payload, result)


    from              core.exceptions.user              import
PasswordOrLoginDoesNotMatch,                UserDoesNotExists,
UserWithSameLoginExists
    from core.cache.backend import redis
    router = APIRouter()
    logic = UserLogic(model=Users)
```

```
    auth_handler = auth.AuthHandler()
    cache_manager    =    CacheManager(backend=RedisBackend(),
key_maker=CustomKeyMaker())
    @router.get("/user/refresh_token",          tags=["user"],
response_model=schemes.UserToken)
    async def refresh_token(request: Request, db: Session =
Depends(get_db), user=Depends(auth_handler.auth_wrapper)):
        return                  {"token":                  await
auth_handler.encode_token(user_id=request.user.id)}
    @router.post(
        "/user",
        tags=["user"],
        status_code=status.HTTP_201_CREATED,
        responses={
            409: {"model": Message},
        },
        response_model_exclude={"id"},
        response_model=schemes.User
    )
    async def create_user(user: schemes.UserCreate, db: Session
= Depends(get_db)):
        operation,   res   =   await   logic.create_user(db=db,
user=user, password=user.password)
        if not operation:
            raise             HTTPException(detail=res.message,
status_code=res.error_code)
        return res.__dict__
    @router.post(
        "/user/login",
        tags=["user"],
        responses={401: {"model": Message}},
        status_code=status.HTTP_200_OK,
        response_model=schemes.UserToken
    )
```

```
    async def login(user: schemes.UserLogin, db: Session =
Depends(get_db)):
        user_old     =     await     logic.get_user_by_login(db,
user.username)
        if user_old and await auth_handler.verify_password(
                plain_password=user.password,
hash_password=user_old.password
        ):
            token = await auth_handler.encode_token(user_old.id)
            return {"token": token}
        raise HTTPException(
            status_code=PasswordOrLoginDoesNotMatch.error_code,
            detail=PasswordOrLoginDoesNotMatch.message,
        )


    @router.delete(
        "/user/{username}",
        tags=["user"],
        responses={404: {"model": Message}},
        status_code=status.HTTP_200_OK,
    )
    async def delete_user(username: str, request: Request, db:
Session = Depends(get_db)):
        operation,    res    =    await    logic.delete_user(db,
username=username)
        if not operation:
            raise              HTTPException(detail=res.message,
status_code=res.error_code)
        return res


    @router.get(
        "/user/{username}",         response_model=schemes.User,
tags=["user"], status_code=status.HTTP_200_OK
```

```
    )
    async def get_user_by_username(
            username: str,
            db: Session = Depends(get_db),
    ):
        res = await logic.get_user_by_login(db, username)
        if not res:
            raise
HTTPException(status_code=UserDoesNotExists.error_code,
detail=UserDoesNotExists.message)
        return res
    @router.get(
        "/user",   response_model=schemes.User,   tags=["user"],
status_code=status.HTTP_200_OK
    )
    async def get_myself(
            request: Request,
            user=Depends(auth_handler.auth_wrapper),
            db: Session = Depends(get_db),
    ):
        res       =       await       logic.get_user_by_id(db,
user_id=request.user.id)
        return res
    @router.patch(
        "/user/{username}",
        tags=["user"],
        responses={404: {"model": Message}},
        status_code=status.HTTP_200_OK,
    )
    async def patch_user(
            username: str, user: schemes.UserPatch, db: Session
= Depends(get_db)
    ):
        res = await logic.get_user_by_login(db, user.username)
        if res:
```

```
            raise
HTTPException(status_code=UserWithSameLoginExists.error_code,
detail=UserWithSameLoginExists.message)

        operation, res = await logic.patch_user(db=db, user=user,
username=username)
        if not operation:
            raise              HTTPException(detail=res.message,
status_code=res.error_code)
        return res
```