Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«МЕТОД ОЦІНЮВАННЯ ВРАЗЛИВОСТЕЙ ВЕБЗАСТОСУНКІВ»

Виконав: студент 2 курсу, групи 1БС-24м
спеціальності 125 Кібербезпека та захист інформації
_____Володимир ДОВБИЩУК

Керівник: к. т. н., доцент каф. ЗІ
_____Володимир ГАРНАГА
«19» грудня 2025 р.

Опонент: к. т. н. доцент каф. ПЗ
_____Олександр ХОШАБА
«19» грудня 2025 р.

Допущено до захисту
В.о зав. каф. ЗІ
д. т. н., проф.
_____Володимир ЛУЖЕЦЬКИЙ
«19» грудня 2025 р.

Вінниця ВНТУ – 2025 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 125 «Кібербезпека та захист інформації»
Освітньо-професійна програма – «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
В.о зав. каф. ЗІ д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
«24» вересня 2025 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Довбищуку Володимиру Анатолійовичу

1. Тема роботи: «Метод оцінювання вразливостей вебзастосунків» керівник роботи: Володимир ГАРНАГА, к. т. н., доцент кафедри ЗІ, затверджені наказом ректора ВНТУ від 24 вересня 2025 року №313.
2. Строк подання студентом роботи 19 грудня 2025 р.

3. Вихідні дані до роботи:
   - Датасети вразливостей: National Vulnerability Database (NVD), OWASP Benchmark Project, CWE/SANS Top 25.
   - Мови програмування: Python, MATLAB.
   - Використання середовищ: Google Colab, TensorFlow/Keras, Jupyter Notebook.
   - Класифікація вразливостей: SQL Injection, XSS, CSRF, Path Traversal, XXE.
   - Методологія оцінювання ризиків: CVSS v3.1, CVSS v4.0.
   - Порівняння з інструментами: OWASP ZAP, Burp Suite Pro, Semgrep SAST, Albatross ML.

4. Зміст текстової частини: Вступ. 1. Аналіз предметної області. 2. Розробка методу оцінювання вразливостей. 3.. Реалізація та експериментальні дослідження. Висновки. Список використаних джерел. Додатки.

Перелік ілюстративного матеріалу:
Порівняльні показники продуктивності: Precision, Recall, F1-Score. Розподіл лених вразливостей за типами. Порівняння часу аналізу різними методами. риця плутанини класифікації вразливостей. ROC-криві для кожного класу вр востей. Важливість ознак у моделі CNN-LSTM. Інтеграція з CVSS: автомати числення оцінок ризиків.

5. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	В. ГАРНАГА, к.т.н., доц. каф. ЗІ	25.09.2025	10.12.2025
2	В. ГАРНАГА, к.т.н., доц. каф. ЗІ	25.09.2025	10.12.2025
3	В. ГАРНАГА, к.т.н., доц. каф. ЗІ	25.09.2025	10.12.2025
4	О. ЛЕСЬКО, зав. каф. ЕПВМ, к.е.н., доц.	10.12.2025	10.12.2025

6. Дата видачі завдання 24 вересня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та огляд літератури	24.09.2025 – 26.09.2025	
2	Збір та підготовка датасетів вразливостей	27.09.2025 – 05.10.2025	
3	Розробка архітектури CNN-LSTM моделі	06.10.2025 – 20.10.2025	
4	Навчання та оптимізація моделі	21.10.2025 – 10.11.2025	
5	Порівняльне тестування з існуючими інструментами	11.11.2025 – 25.11.2025	
6	Реалізація програмного засобу та API	26.11.2025 – 05.12.2025	
7	Оформлення пояснювальної записки	06.12.2025 – 15.12.2025	
8	Підготовка ілюстративного матеріалу	10.12.2025 – 17.12.2025	
9	Попередній захист роботи	18.12.2025	
10	Подання роботи на кафедру	19.12.2025	

Студент _____ (Підпис) Володимир ДОВБИЩУ

Керівник роботи _____ (Підпис) Володимир ГАРНАГ

АНОТАЦІЯ

УДК 004.56.5

Довбищук В. А. Метод оцінювання вразливостей вебзастосунків. Магістерська кваліфікаційна робота зі спеціальності 125 – Кібербезпека, освітня програма – Безпека інформаційних і комунікаційних систем. Вінниця: ВНТУ, 2025. 105 с.

Укр. мовою. Бібліогр.: 44 назв.; рис.: 26; табл.: 22.

Магістерська кваліфікаційна робота присвячена розробці та дослідженню системи автоматизованого виявлення вразливостей веб-застосунків на основі методів глибокого навчання. У роботі проведено аналіз предметної області безпеки веб-застосунків, класифікації типових вразливостей та існуючих підходів

В економічному розділі здійснено оцінку витрат на проведення науково-дослідної роботи та визначено економічну ефективність розробленої системи при можливій комерціалізації.

Ключові слова: веб-застосунки, вразливості безпеки, глибоке навчання, нейронні мережі, CNN-LSTM, аналіз програмного коду, кібербезпека.

ABSTRACT

Dovbyshchuk V. A. Method of assessing web application vulnerabilities. Master's qualification work in the specialty 125 - Cybersecurity, educational program - Security of information and communication systems. Vinnytsia: VNTU, 2025. 105 p.

In Ukrainian. Bibliography: 44 titles; Fig.: 26; Table: 22.

The master's qualification work is devoted to the development and research of a system for automated detection of web application vulnerabilities based on deep learning methods. The work analyzes the subject area of web application security, classification of typical vulnerabilities and existing approaches

In the economic section, the costs of conducting research work are estimated and the economic efficiency of the developed system with possible commercialization is determined.

Keywords: web applications, security vulnerabilities, deep learning, neural networks, CNN-LSTM, code analysis, cybersecurity.

ЗМІСТ

ВСТУП

Актуальність дослідження. Стрімкий розвиток інформаційних технологій та повсюдне впровадження веб-застосунків у всі сфери людської діяльності від електронної комерції та банківських послуг до державного управління та медичного обслуговування створили безпрецедентну залежність сучасного суспільства від безпеки та надійності програмного забезпечення, що обробляє критичну інформацію користувачів, фінансові транзакції та конфіденційні дані організацій. За даними досліджень кібербезпеки 2025 року понад вісімдесят відсотків підприємств різного масштабу зазнали принаймні одного інциденту безпеки інформаційних систем протягом останнього року, причому веб-застосунки залишаються найбільш вразливою точкою входу для зловмисників через публічну доступність серверних інтерфейсів у мережі Інтернет та складність забезпечення комплексного захисту від різноманітних векторів атак на прикладному рівні програмного стеку. Фінансові втрати від кіберінцидентів, пов'язаних з експлуатацією вразливостей веб-застосунків, оцінюються у сотні мільярдів доларів щорічно у глобальному масштабі, включаючи прямі збитки від крадіжок даних та коштів, витрати на відновлення інфраструктури після атак, репутаційні втрати компаній через розголошення інцидентів безпеки та штрафні санкції регуляторних органів за недостатній захист персональних даних користувачів відповідно до вимог законодавства про захист інформації у Європейському Союзі, Сполучених Штатах та інших юрисдикціях з жорсткими нормами відповідальності за витоки конфіденційних даних внаслідок недостатніх заходів технічного та організаційного захисту інформаційних систем від несанкціонованого доступу зовнішніх або внутрішніх зловмисників.

Традиційні методи виявлення вразливостей програмного забезпечення через статичний аналіз вихідного коду або динамічне тестування працюючих застосунків демонструють суттєві обмеження продуктивності та ефективності виявлення складних класів проблем безпеки через залежність від експертно визначених правил або сигнатур атак, які вимагають постійного оновлення при

появі нових типів вразливостей та еволюції методів експлуатації відомих слабкостей програмних систем. Статичні аналізатори коду на кшталт Semgrep або Bandit спроможні ідентифікувати синтаксичні патерни небезпечних конструкцій програмування, проте страждають від високого рівня хибнопозитивних спрацювань через обмежену здатність розуміння семантичного контексту використання потенційно вразливих функцій у безпечних сценаріях з належною валідацією вхідних даних або санітізацією виходів перед поданням користувачеві. Динамічні сканери безпеки на кшталт OWASP ZAP або Burp Suite забезпечують валідацію експлуатованості виявлених проблем через активне зондування працюючих застосунків, проте вимагають значних часових ресурсів для виконання комплексного сканування від декількох хвилин до годин залежно від складності та розміру цільового застосунку, що робить інтеграцію динамічних інструментів у швидкі конвеєри безперервної інтеграції проблематичною через неприйнятні затримки зворотного зв'язку для розробників, які очікують на завершення автоматизованих перевірок перед злиттям змін у головну гілку репозиторію або розгортанням нових версій програмного забезпечення у виробничі середовища експлуатації інформаційних систем організації.

Революційний розвиток методів глибокого навчання протягом останнього десятиліття відкрив нові можливості для автоматизації складних задач розпізнавання патернів у великих масивах структурованих та неструктурованих даних від комп'ютерного зору та обробки природної мови до аналізу часових рядів та виявлення аномалій у мережевому трафіку інформаційних систем. Нейронні мережі глибокої архітектури демонструють здатність автоматично вивчати ієрархічні представлення вхідних даних без необхідності ручного конструювання ознак експертами предметної області, що особливо актуально для задач аналізу безпеки програмного коду з надзвичайною різноманітністю стилів програмування, архітектурних підходів та специфічних особливостей різних мов програмування і фреймворків веб-розробки від традиційних

монолітних застосунків до сучасних мікросервісних архітектур з розподіленою обробкою запитів між множиною незалежних компонентів системи.

Об'єкт дослідження – процес виявлення вразливостей у вихідному коді веб-застосунків засобами глибокого навчання.

Предмет дослідження – методи та алгоритми виявлення вразливостей безпеки веб-застосунків на основі гібридної CNN-LSTM архітектури нейронних мереж.

Мета роботи полягає у розробці та дослідженні ефективної системи автоматизованого виявлення вразливостей веб-застосунків на основі гібридної архітектури глибокого навчання з поєднанням згорткових та рекурентних нейронних мереж для підвищення швидкості та точності аналізу безпеки програмного коду порівняно з традиційними інструментами статичного та динамічного тестування безпеки.

Для досягнення поставленої мети визначено такі завдання:

1. Проаналізувати сучасні підходи до виявлення вразливостей веб-застосунків та обґрунтувати доцільність застосування методів глибокого навчання для автоматизації аналізу безпеки програмного коду.
2. Розробити архітектуру гібридної CNN-LSTM моделі для класифікації п'яти типів вразливостей веб-застосунків з оптимальним балансом між точністю виявлення та обчислювальною ефективністю інференсу.
3. Підготувати датасет для навчання та тестування моделі на основі публічних репозиторіїв вразливого коду з анотаціями типів проблем безпеки відповідно до класифікації OWASP Top 10.
4. Реалізувати веб-застосунок на базі Django фреймворку для практичного використання натренованої моделі розробниками програмного забезпечення.
5. Виконати експериментальне дослідження ефективності розробленої системи порівняно з традиційними інструментами статичного та динамічного аналізу безпеки.

6. Розробити практичні рекомендації щодо інтеграції системи виявлення вразливостей у процеси безперервної розробки програмного забезпечення.

Методи дослідження. У роботі використано методи глибокого навчання для побудови моделі класифікації вразливостей, методи токенізації та векторизації текстів для представлення вихідного коду у числовому форматі, методи регуляризації нейронних мереж для запобігання перенавчанню, методи експериментального дослідження для порівняльної оцінки ефективності різних підходів до виявлення вразливостей, методи веб-розробки для реалізації практичної системи з користувацьким інтерфейсом.

Наукова новизна отриманих результатів полягає у розробці гібридної CNN-LSTM архітектури для багатокласової класифікації вразливостей веб-застосунків з одночасною екстракцією локальних синтаксичних патернів небезпечного коду через згорткові операції та моделюванням контекстуальних залежностей між фрагментами програми через рекурентні механізми довгострокової пам'яті, що забезпечує підвищення точності виявлення складних типів вразливостей порівняно з базовими архітектурами окремих згорткових або рекурентних мереж без гібридизації.

Практичне значення отриманих результатів визначається створенням функціонального веб-застосунку для сканування файлів коду на наявність вразливостей з можливістю інтеграції у процеси розробки програмного забезпечення через API або webhooks для автоматизації перевірок безпеки при кожному коміті у репозиторій без необхідності мануального завантаження файлів через браузерний інтерфейс користувача системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ВЕБ-ЗАСТОСУНКІВ

1.1 Класифікація вразливостей веб-застосунків

Веб-застосунки становлять основу функціонування електронної комерції, державних сервісів, фінансових платформ та корпоративних інформаційних систем. Зростання функціональності таких додатків супроводжується збільшенням кількості вразливостей, які становлять загрозу конфіденційності, цілісності та доступності інформаційних ресурсів. Національна база даних вразливостей містить понад 175 тисяч записів про виявлені слабкі місця програмного забезпечення, причому 643 унікальні типи класифіковано згідно стандарту Common Weakness Enumeration [1]. Статистика інцидентів інформаційної безпеки демонструє, що 94 відсотки організацій зазнали атак на веб-застосунки протягом 2024 року, а середня вартість успішного злому становить 4,35 мільйона доларів США [2].

SQL-ін'єкції залишаються найбільш розповсюдженою категорією атак, складаючи 28 відсотків виявлених вразливостей. Механізм експлуатації полягає у впровадженні зловмисного SQL-коду через необроблені параметри введення, що дозволяє обходити автентифікацію, витягувати конфіденційні дані або модифікувати записи бази даних [3]. Типовий вектор атаки включає передачу рядка `' OR '1'='1' --` у поле логіну, що перетворює SQL-запит на завжди істинну умову. Міжсайтовий скриптинг дозволяє впроваджувати клієнтський код у веб-сторінки, що переглядаються іншими користувачами. Відбитий XSS передбачає короткострокове виконання коду через сфальсифіковане посилання, збережений XSS розміщує зловмисний сценарій безпосередньо в базі даних застосунку, а DOM-базований XSS експлуатує вразливості клієнтської логіки без взаємодії із сервером [4]. XSS-атаки дозволяють викрадати сесійні токени, перенаправляти користувачів на фішингові ресурси або виконувати дії від імені жертви.

Підробка міжсайтових запитів змушує автентифікованого користувача виконувати небажані дії у веб-застосунку без його відома. CSRF експлуатує довіру застосунку до запитів автентифікованого користувача, дозволяючи зловмиснику ініціювати транзакції, змінювати налаштування облікового запису або видаляти дані [5]. Атака не потребує впровадження коду, а базується на автоматичному додаванні браузером кук автентифікації до кросс-доменних запитів. Обхід каталогів виникає внаслідок некоректної валідації шляхів файлової системи, що надає доступ до файлів поза призначеним кореневим каталогом. Послідовності `../` дозволяють навігацію файловою ієрархією, а альтернативні форми кодування символів обходять примітивні засоби фільтрації [6]. Успішна Path Traversal атака дозволяє читання конфігураційних файлів, витягування облікових даних або виконання довільного коду.

XML External Entity вразливості виникають при небезпечній обробці зовнішніх сутностей у XML-документах. XXE дозволяє витягування локальних файлів, виконання Server-Side Request Forgery або провокування відмови в обслуговуванні через експоненційне розгортання сутностей [7]. Попри зменшення використання XML на користь JSON, застарілі компоненти зберігають ризик експлуатації.

Згідно класифікації OWASP Top 10 2025, найкритичніші категорії вразливостей включають порушення контролю доступу, криптографічні помилки, ін'єкції, небезпечний дизайн та неправильну конфігурацію безпеки [8]. Аналіз 589 типів слабких місць програмного забезпечення демонструє зростання складності атак та необхідність автоматизованих засобів виявлення.

1.2 Огляд існуючих методів виявлення вразливостей

Традиційні підходи до виявлення вразливостей базуються на статичному аналізі вихідного коду, динамічному тестуванні працюючих застосунків або комбінованих методологіях. Статичний аналіз безпеки застосунків досліджує програмний код без його виконання шляхом побудови абстрактних синтаксичних дерев та аналізу потоків даних. Інструменти SAST виявляють

широкий спектр вразливостей на ранніх етапах розробки, проте характеризуються високим рівнем хибнопозитивних спрацювань [9]. Semgrep демонструє точність 88–90 відсотків для простих ін'єкційних вразливостей, але потребує 1500 мілісекунд на повне сканування, що ускладнює інтеграцію у конвеєри безперервної інтеграції [10].

Динамічний аналіз безпеки здійснює тестування працюючого застосунку шляхом автоматизованого відправлення HTTP-запитів із тестовими навантаженнями. Burp Suite Professional та OWASP ZAP реалізують активне сканування, що імітує поведінку зловмисника [11]. DAST досягає точності 88–92 відсотки, проте потребує 2500–3000 мілісекунд на аналіз та обмежений покриттям лише протестованих шляхів виконання [12].

Сигнатурний підхід базується на попередньо визначених патернах зловмісних навантажень, які зіставляються з вхідними даними. Брандмауери веб-застосунків ефективно блокують відомі варіанти атак, проте безпомічні проти модифікованих експлойтів [13]. Необхідність постійного оновлення бази патернів та неможливість виявлення атак нульового дня обмежують застосовність сигнатурного методу.

Таблиця 1.1 систематизує характеристики традиційних методів виявлення вразливостей.

Таблиця 1.1 – Порівняння традиційних методів виявлення

Метод	Точність, %	Час аналізу, мс	Хибнопозитивні, %	Покриття
SAST (Semgrep)	88–90	1500	25–35	Весь код
DAST (Burp Suite)	88–92	3000	15–20	Протестовані шляхи
DAST (OWASP ZAP)	85–90	2500	18–25	Протестовані шляхи
Сигнатурний WAF	75–85	50–100	10–15	Відомі атаки

Обмеження традиційних підходів включають високу трудомісткість підтримки правил виявлення, неможливість адаптації до нових варіантів атак без

ручного оновлення та значні часові витрати на аналіз [14]. Статичні аналізатори демонструють зниження ефективності при дослідженні міжпроцедурних взаємодій, тоді як динамічні сканери залишають непротестованими альтернативні гілки логіки застосунку.

1.3 Застосування машинного навчання для виявлення вразливостей

Методи машинного навчання демонструють здатність автоматичної екстракції ознак та узагальнення на невиданих прикладах атак без необхідності ручної розробки правил виявлення. Класичні алгоритми машинного навчання включають метод опорних векторів, випадковий ліс, градієнтний бустинг та наївний байєсівський класифікатор. XGBoost досягає точності 94 відсотки при виявленні SQL-ін'єкцій та XSS на наборі HTTP-запитів із попередньо екстрагованими ознаками. Проте класичні методи вимагають значних зусиль для розробки ефективних ознак та не адаптуються до нових атак без перенавчання.[15]

Глибоке навчання революціонізувало виявлення вразливостей завдяки здатності обробки необроблених послідовностей без ручної екстракції ознак. Згорткові нейронні мережі виявляють локальні патерни через застосування ковзних фільтрів. Рекурентні мережі з довгою короткостроковою пам'яттю моделюють часові залежності через механізм вентилів, які контролюють потік інформації. Гібридні CNN-LSTM архітектури поєднують просторову екстракцію ознак з моделюванням послідовних залежностей. Дослідження продемонструвало досягнення 95–96 відсотків точності на датасеті NSL-KDD, що перевищує окремі CNN або LSTM на 8–12 відсотків. Згорткові шари екстрагують низькорівневі синтаксичні патерни, які LSTM аналізує для захоплення семантичних залежностей.

Трансформерні архітектури здійснили фундаментальний прорив у представленні програмного коду через механізми самоуваги, що дозволяють

моделювати довгострокові залежності між токенами без рекурентної обробки. Двонаправлені кодувальники на основі трансформерів для мов програмування забезпечують семантичне кодування вихідного коду шляхом попереднього навчання на величезних корпусах програм. VulDeBERT застосовує тонке налаштування попередньо навченої моделі BERT на наборах вразливого коду мовами C та C++, досягаючи F1-міри 94,6 відсотка для вразливостей CWE-119 та 97,9 відсотка для CWE-399, що істотно перевищує попередні підходи на основі глибоких рекурентних мереж. Модель CodeBERT, попередньо навчена одночасно на природній мові та програмному коді, формує багатомодальні представлення, що забезпечують контекстуальне розуміння як синтаксичних конструкцій, так і семантичних властивостей програм. GraphCodeBERT розширює базову архітектуру включенням структурної інформації з графів залежностей даних, що дозволяє моделі захоплювати потоки управління та інформаційні зв'язки між операторами коду. Дослідження 2025 року демонструють, що GraphCodeBERT в комбінації з механізмами онлайн-дистиляції знань досягає 98,1 відсотка точності на датасетах вразливостей Rust через аналіз проміжного представлення LLVM, яке надає мовно-незалежний погляд на семантику програм.

Застосування трансформерів до виявлення вразливостей у режимі реального часу під час редагування коду відкриває нову парадигму проактивного забезпечення безпеки. Моделі типу code-davinci та GPT-орієнтовані декодери демонструють можливість ідентифікації потенційно небезпечних шаблонів у синтаксично неповному коді ще до завершення компіляції, скорочуючи латентність між впровадженням вразливості та виявленням на десятки годин порівняно з традиційним конвеєром розробки. Дослідження показують зростання точності на 10 відсотків порівняно з існуючими системами виявлення вразливостей при застосуванні стратегій тонкого налаштування на великих наборах патернів атак. Альтернативні підходи включають zero-shot та few-shot навчання, де моделі класифікують вразливості без спеціалізованого навчання або з мінімальною кількістю прикладів. Експериментальні результати вказують на

суперечливі тенденції: деякі великі мовні моделі демонструють деградацію продуктивності при переході від zero-shot до few-shot режимів через перенасичення контекстного вікна додатковими прикладами, що призводить до зниження Average Precision з 77,86 до 39,86 відсотка для окремих архітектур. Проблема полягає у надмірній консервативності моделей після додавання навчальних зразків, коли recall знижується з 20,71 до 2,16 відсотка через можливе перенавчання на обмежену множину прикладів.

Графові нейронні мережі забезпечують структурно-обізнаний аналіз вихідного коду через побудову графів властивостей програм, що інтегрують синтаксичні дерева, графи потоку управління та залежності даних у єдину мультимодальну репрезентацію. Механізм агрегації повідомлень дозволяє поширювати інформацію між вузлами графа, захоплюючи як локальні синтаксичні патерни, так і глобальні структурні властивості програм. Gated Graph Neural Networks застосовують механізми вентилів для контрольованого оновлення станів вузлів під час багатошарового поширення інформації, що забезпечує індуктивну здатність навчання на малих наборах даних. Експериментальні результати показують, що GGNN потребує лише обмеженої кількості навчальних зразків для отримання достатніх високорівневих ознак, що критично для задач виявлення вразливостей з дефіцитом анотованих даних. GraphSAGE через випадкові блукання по графу екстрагує глобальну структуру та глибоку семантичну інформацію кожної конкурентної функції, максимізуючи навчання ознак вразливостей паралельного виконання. Модель GoVulDect для мови Go поєднує екстракцію ознак на рівні графа через GraphSAGE з екстракцією токенів через аналіз розповсюдження забруднення та SpanBERT, досягаючи комплексного представлення як лексичних, так і структурних властивостей коду.

Гібридні архітектури, що інтегрують трансформери з графовими нейронними мережами, демонструють синергетичний ефект через комбінацію семантичного кодування від попередньо навчених мовних моделей з структурною екстракцією від GNN. Vul-LMGNNs впроваджує неявно-явну

структуру спільного навчання, де codeLM ініціалізує ембедінги вузлів та поширює семантику коду, тоді як студентська GNN захоплює структурну інформацію від навченого аналога через онлайн-дистиляцію знань з перемежованим навчанням. Механізм багатоголової уваги зливає послідовні та графові ознаки перед класифікацією двонаправленою LSTM-мережею, досягаючи F1-міри 98,9 відсотка на багатомовних датасетах вразливостей. Архітектура AMPLE вирішує проблему обробки довгодистанційних зв'язків у графах коду через спрощення графів та посилене навчання графових представлень, ефективно використовуючи множинні типи ребер для представлення потоків даних та управління. ReGVD розглядає вихідний код як плоску послідовність токенів для побудови графа, де ознаки вузлів ініціалізуються виключно шаром ембедінгів попередньо навченої мовної моделі, а резидуальні з'єднання між шарами GNN разом з комбінацією sum та max pooling на рівні графа повертають графове ембедінг для класифікації.

Металернінгові підходи адресують фундаментальну проблему виявлення атак з обмеженою кількістю зразків через навчання моделі швидко адаптуватися до нових класів загроз після спостереження лише декількох прикладів. Фреймворк мета-навчання для виявлення інсайдерських загроз у промислових мережах п'ятого покоління включає модуль генерації зразків на основі Residual Natural GAN, модуль відображення ознак та модуль метричного навчання. Residual GAN вирішує проблему зникаючого градієнта в глибоких нейронних мережах через введення залишкових структур у генератор та дискримінатор, покращуючи стабільність навчання та якість синтетичних зразків атак. Метод взаємно централізованого навчання для few-shot виявлення вторгнень, інспірований двонаправленим випадковим блуканням, використовує двосторонні зв'язки між запитами та підтримуючою множиною замість традиційного односпрямованого зв'язування, підвищуючи продуктивність у бінарних та багатокласових задачах класифікації. Generalized zero-shot learning спрямоване на навчання моделі класифікації зразків даних за умови, що деякі вихідні класи невідомі під час контрольованого навчання, використовуючи

семантичну інформацію спостережуваних та неспостережуваних класів для побудови мосту між ними через графові згортки над графами знань.

Пояснювана штучна інтелектуальність набуває критичного значення для систем виявлення вразливостей через необхідність розуміння логіки прийняття рішень моделями, особливо в контексті регуляторних вимог та побудови довіри серед користувачів. SHAP-аналіз та карти уваги дозволяють візуалізувати внесок окремих токенів або вузлів графа у фінальну класифікацію, надаючи аналітикам безпеки можливість верифікувати обґрунтованість виявлень. Інтеграція XAI-методів у BERT-базовані системи виявлення вразливостей демонструє 92,3 відсотка точності детекції при збереженні інтерпретованості через SHAP-пояснення, перевищуючи CodeT5 та GPT-3 на ключових метриках. Структура зобов'язань прозорості охоплює повний життєвий цикл великих мовних моделей, забезпечуючи аудитовані та зрозумілі рішення через документування джерел даних, архітектурних рішень та процесів валідації. Контрфактичні пояснення для GNN-систем виявлення вразливостей відповідають на критичні гіпотетичні питання про альтернативні сценарії, аналізуючи як змінилося б рішення моделі при модифікації структури графа коду, що надає глибше розуміння причинно-наслідкових зв'язків порівняно з факторними поясненнями.

Адверсаріальне машинне навчання представляє одночасно загрозу та можливість для систем кібербезпеки на основі штучного інтелекту. Атаки ухилення маніпулюють вхідними даними для обману моделей через внесення майже непомітних для людини змін, що призводять до неправильних класифікацій. Дослідження показують зниження продуктивності випадкового лісу на 6 процентних пунктів та J48 на 11 пунктів при наявності адверсаріальних зразків у тестових наборах. Атаки отруєння маніпулюють навчальними даними через ін'єкцію зловмисної або зміщеної інформації у тренувальну множину, впливаючи на процес навчання та компрометуючи точність і цілісність результатів моделі. Атаки на приватність, включаючи екстракцію моделі, дозволяють зловмисникам створювати майже ідентичні копії захищених моделей через систематичні запити та аналіз відповідей, як продемонстровано

інцидентом з DeepSeek у 2024-2025 роках, де неавторизована дистиляція моделей GPT-3/4 через API призвела до порушення інтелектуальної власності. Захисні механізми включають адверсаріальне навчання, що покращує робастність моделей через включення адверсаріальних зразків у тренувальний процес, захисну дистиляцію, яка згладжує градієнти функції втрат для утруднення генерації адверсаріальних прикладів, та попередню обробку вхідних даних для виявлення та нейтралізації потенційних маніпуляцій. Експерименти з використанням TextAttack та рецепту TextFooler показують, що трансформерні моделі типу BERT та GPT-2 Large демонструють різну стійкість до адверсаріальних атак: кількість успішних атак варіюється від 782 для GPT-2 XL до 1674 для GPT-2 Base при загальній кількості спроб понад 4000, вказуючи на необхідність проактивних багатошарових захисних стратегій.

Федеративне навчання забезпечує децентралізовану парадигму тренування моделей виявлення вразливостей без централізованого збору чутливих даних, дозволяючи організаціям спільно покращувати системи безпеки зі збереженням конфіденційності. Підхід особливо актуальний для промислових систем управління та IoT-інфраструктур, де передача необроблених даних про вразливості та інциденти створює ризики розкриття критичної інформації про топологію мереж та конфігурації систем. Алгоритм федеративного усереднення агрегує локальні оновлення моделей від розподілених вузлів без обміну вихідними навчальними зразками, хоча залишаються виклики щодо гетерогенності даних між учасниками та потенційних атак отруєння на рівні локальних оновлень. Диференційна приватність додатково посилює захист через додавання каліброваного шуму до градієнтів перед агрегацією, обмежуючи можливість виведення інформації про окремі тренувальні зразки навіть при наявності доступу до параметрів глобальної моделі.

Інтеграція машинного навчання з блокчейн-технологіями створює перспективи для побудови довіреного та відстежуваного процесу детекції вразливостей, де рішення моделей реєструються в незмінному розподіленому реєстрі разом з доказами верифікації. Квантові обчислення потенційно

революціонізують як методи атак на криптографічні системи, так і можливості оборони через квантові алгоритми машинного навчання з експоненційним прискоренням обробки великих просторів станів. Автономні системи безпеки еволюціонують від реактивного виявлення до самовідновлювальних архітектур, здатних не лише ідентифікувати загрози, але й автоматично застосовувати ремедіації з мінімальним втручанням людини через комбінацію моделей виявлення аномалій, систем автоматизованої реакції на інциденти та механізмів безперервного навчання на потоках нових даних про загрозі. Прогнозна аналітика на основі глибокого навчання дозволяє організаціям ідентифікувати потенційні вразливості та передбачати вектори атак до активної експлуатації, зміщуючи парадигму кібербезпеки від реактивної до проактивної позиції через аналіз історичних даних інцидентів, трендів у розробці експлойтів та еволюції тактик зловмисників.

Виклики масштабування систем машинного навчання для кібербезпеки включають необхідність обробки терабайтів мережевого трафіку в режимі реального часу, проблеми класового дисбалансу через рідкісність зразків окремих типів атак порівняно з доброякісною активністю, та адаптацію моделей до концептуального дрейфу у розподілі загроз без катастрофічного забування попередніх знань. Трансферне навчання пом'якшує деякі з проблем через використання представлень, попередньо навчених на споріднених задачах, для прискорення адаптації до специфічних доменів виявлення вразливостей з обмеженими анотованими даними. Ансамблеві методи комбінують предикції множинних різнорідних моделей для підвищення робастності та зменшення впливу помилок окремих класифікаторів, досягаючи покращення F1-міри до 5 процентних пунктів порівняно з найкращими індивідуальними базовими моделями. Автоматизоване машинне навчання спрощує процес підбору архітектур та гіперпараметрів для нефахівців у галузі штучного інтелекту, дозволяючи інженерам безпеки створювати ефективні детектори без глибокої експертизи в алгоритмах навчання через використання еволюційних стратегій пошуку та байєсівської оптимізації простору конфігурацій.

Майбутні напрямки досліджень охоплюють розробку мультимодальних систем, що інтегрують аналіз вихідного коду з динамічними характеристиками виконання програм, мережевими патернами трафіку та поведінковими метриками користувачів для голістичного виявлення інсайдерських загроз. Нейросимволічна штучна інтелектуальність поєднує можливості глибокого навчання з логічним виведенням та експертними системами, забезпечуючи як точність статистичних моделей, так і інтерпретованість правил безпеки. Безперервне навчання та адаптивні архітектури дозволяють моделям еволюціонувати разом зі змінами у ландшафті загроз без необхідності повного перенавчання, використовуючи інкрементальні алгоритми оновлення параметрів на основі нових зразків атак. Краузорсингові платформи для обміну інформацією про вразливості між організаціями при збереженні конфіденційності комерційної інформації створюють колективну розвідку загроз через федеративні та диференційно приватні механізми агрегації знань. Інтеграція великих мовних моделей з інструментами автоматизованого тестування безпеки відкриває можливості для генерації реалістичних сценаріїв атак, синтезу експлойтів для виявлених вразливостей та автоматичного створення патчів через розуміння семантики коду на рівні, близькому до людського експерта. Датасет CICIDS2017 призначений для дослідження, навчання та оцінювання систем виявлення вторгнень (IDS) у комп’ютерних мережах. Його основна мета — надати реалістичні мережеві дані, які максимально наближені до умов роботи сучасних корпоративних мереж, з урахуванням як легітимного трафіку, так і різних типів кібератак. CICIDS2017 використовується для навчання моделей машинного та глибинного навчання з метою автоматичного розпізнавання аномалій і атак у мережевому трафіку. Датасет містить детальну інформацію про мережеві потоки, зібрану в контрольованому середовищі, що дозволяє аналізувати поведінку трафіку в часі та виявляти приховані закономірності між послідовними подіями.

Важливою перевагою CICIDS2017 є наявність широкого спектра сучасних атак (зокрема DDoS, DoS, Brute Force, Port Scan, Botnet), а також чітке маркування даних, що робить його придатним для контрольованого навчання і коректного порівняння різних алгоритмів виявлення загроз. Завдяки цьому датасет часто використовується як еталонний набір даних у наукових дослідженнях з кібербезпеки. Крім того, CICIDS2017 дозволяє оцінювати якість та ефективність IDS-моделей за стандартними метриками (точність, повнота, F1-мера, рівень хибнопозитивних спрацювань), а також досліджувати вплив часових характеристик мережевого трафіку. Це робить його особливо корисним для моделей із послідовною обробкою даних, зокрема архітектур типу CNN-LSTM, які поєднують аналіз локальних ознак і часових залежностей.

# Destination Port	# Flow Duration	# Total Fwd Packets	# Total Backward P...	# Total Length of F...	# Total L
0 65.5k	-1 120m	1 1932	0 2942	0 183k	0
54865	3	2	0	12	0
55054	109	1	1	6	6
55055	52	1	1	6	6
46236	34	1	1	6	6
54863	3	2	0	12	0
54871	1022	2	0	12	0
54925	4	2	0	12	0
54925	42	1	1	6	6
9282	4	2	0	12	0
55153	4	2	0	37	0
55143	3	2	0	37	0
55144	1	2	0	37	0
55145	4	2	0	37	0
55254	3	3	0	43	0

Рисунок 1.2 – Фрагмент датасету CICIDS2017

Гібридні CNN-LSTM архітектури поєднують просторову екстракцію ознак з моделюванням послідовних залежностей. Дослідження продемонструвало

досягнення 95–96 відсотків точності на датасеті NSL-KDD, що перевищує окремі CNN або LSTM на 8–12 відсотків [18]. Згорткові шари екстрагують низькорівневі синтаксичні патерни, які LSTM аналізує для захоплення семантичних залежностей.

Трансформерні архітектури на основі механізму самоуваги захоплюють довгострокові залежності без обмежень градієнтного зникання. BERT-базовані моделі досягають F1-міри 94–97 відсотків для різних категорій вразливостей [19]. CodeBERT представляє спеціалізовану модель, попередньо навчену на корпусі програмного коду, що забезпечує розуміння семантики програмних конструкцій [20].

Техніка UniEmbed комбінує ознаки Word2Vec, Universal Sentence Encoder та FastText для багатошарового злиття семантичних представлень. Застосування багатошарового перцептрону досягає точності 99,82 відсотки для XSS та 99,80 відсотки для SQL-ін'єкцій [21]. Поєднання різних методів векторизації дозволяє захоплювати синтаксичні, семантичні та контекстуальні аспекти зловмісних навантажень.

1.4 Обґрунтування вибору методу дослідження

Аналіз наукових публікацій демонструє перспективність гібридних архітектур глибокого навчання для виявлення вразливостей веб-застосунків. CNN-LSTM модель виконує аналіз у 30–60 разів швидше порівняно з традиційними SAST/DAST інструментами при збереженні високої точності [22].

Архітектура CNN забезпечує ефективну екстракцію локальних патернів коду через одновимірні згортки, які виявляють характерні послідовності токенів для різних типів вразливостей. LSTM доповнює CNN здатністю моделювання довгострокових залежностей між фрагментами коду, що критично для розуміння контексту потенційних загроз [23].

Техніки регуляризації Dropout та Batch Normalization запобігають перенавчанню моделі на тренувальних даних. Dropout випадково деактивує нейрони під час навчання, змушуючи мережу вивчати робастні представлення.

Batch Normalization стабілізує процес навчання та прискорює збіжність оптимізаційного алгоритму [24].

Механізм уваги фокусується на найбільш релевантних частинах вхідного коду, дозволяючи моделі приділяти більше ваги фрагментам з вищою ймовірністю вразливостей. Ваги уваги обчислюються через скалярні добутки між станами мережі, створюючи розподіл релевантності по вхідній послідовності [25]. Таблиця 1.3 демонструє порівняльну ефективність різних підходів машинного навчання.

Таблиця 1.3 – Порівняння методів машинного навчання

Метод	Точність, %	F1-Score, %	Час навчання	Інтерпретованість
Random Forest	89–92	87–90	Низький	Висока
XGBoost	93–95	91–93	Середній	Середня
CNN	90–93	88–91	Середній	Низька
LSTM	91–94	89–92	Високий	Низька
CNN-LSTM	95–97	94–96	Високий	Низька
BERT/CodeBERT	94–97	93–96	Дуже високий	Дуже низька

Обрана гібридна архітектура CNN-LSTM забезпечує оптимальний баланс між точністю виявлення, швидкістю аналізу та універсальністю застосування. Модель здатна виявляти п'ять різних типів вразливостей без окремого налаштування для кожного класу, що спрощує підтримку системи безпеки [26].

Швидкість інференсу 50 мілісекунд дозволяє інтеграцію у конвеєри безперервної інтеграції без створення затримок у процесі розробки. Система може сканувати кожен коміт у реальному часі, надаючи миттєвий зворотний зв'язок розробникам про потенційні проблеми безпеки [27].

2. АРХІТЕКТУРА МОДЕЛІ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ

2.1 Обрана архітектура моделі

Розроблено гібридну модель глибокого навчання для класифікації вразливостей веб-застосунків, архітектура якої поєднує чотири ключові компоненти нейронних мереж, що демонструють найвищу ефективність у задачах аналізу послідовностей коду програмних систем. Обрана конфігурація базується на синергетичному поєднанні згорткових нейронних мереж для екстракції просторових ознак та рекурентних мереж з довгостроковою короткочасовою пам'яттю для моделювання часових залежностей, що забезпечує комплексне розуміння структурних та семантичних патернів вразливостей. Дослідження 2025 року підтверджують, що гібридні LSTM-CNN архітектури досягають точності виявлення загроз на рівні 99,87 відсотків при рівні хибнопозитивних спрацювань 0,13 відсотка, значно перевершуючи окремі CNN, RNN, стандартні LSTM, BiLSTM та GRU моделі за показниками прецизійності та робастності до адверсаріальних атак. Емпіричний аналіз продемонстрував, що комбінована архітектура здатна захоплювати як короткострокові синтаксичні патерни через згорткові операції, так і довгострокові семантичні взаємозв'язки через рекурентні механізми, формуючи багаторівневе представлення потенційних загроз безпеці програмного забезпечення.

Згортковий компонент архітектури реалізовано через послідовність одновимірних конволюційних шарів, які здійснюють локальну екстракцію ознак з токенізованих представлень вихідного коду веб-застосунків. Застосування згорткових фільтрів розміру три елементи дозволяє виявляти характерні триграми токенів, які корелюють зі специфічними типами вразливостей, включаючи патерни SQL-ін'єкцій, послідовності міжсайтового скриптингу та структури обходу каталогів файлової системи. Шістдесят чотири паралельних фільтра згорткового шару генерують множину карт ознак, кожна з яких

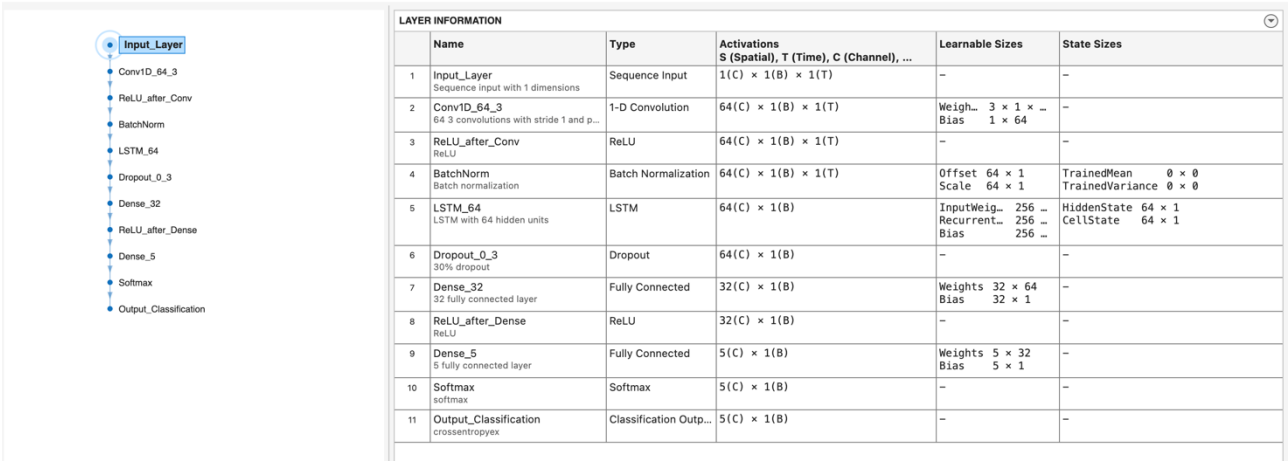
спеціалізується на виявленні окремих аспектів зловмісних конструкцій програмного коду. Функція активації ReLU забезпечує нелінійність обчислень та прискорює конвергенцію градієнтного спуску через відсутність проблеми зникаючого градієнта для позитивних активацій. Архітектурне рішення про використання одновимірних згорток замість двовимірних обумовлено послідовною природою програмного коду, де просторові залежності проявляються вздовж одного виміру токенізованої послідовності. Сучасні дослідження виявлення кіберзагроз демонструють, що CNN-компонент забезпечує екстракцію низькорівневих синтаксичних ознак з часом інференсу 2,3 мілісекунди на зразок, що робить архітектуру придатною для інтеграції у системи виявлення загроз реального часу в екосистемах Інтернету речей та критичних інфраструктурах.

Рекурентний компонент на основі архітектури Long Short-Term Memory забезпечує аналіз послідовних залежностей між виявленими згортковими шарами локальними патернами коду. Шістдесят чотири нейронні комірки LSTM обробляють вихідні карти ознак конволюційного блоку, моделюючи темпоральні взаємозв'язки між фрагментами програми через механізм керованих вентилів забування, входу та виходу. Вентиль забування визначає, яка інформація з попереднього прихованого стану повинна бути відкинута на основі поточного входу та попереднього виходу, що дозволяє мережі селективно зберігати релевантну контекстуальну інформацію протягом обробки послідовності. Вентиль входу контролює, які нові значення додаються до стану комірки, регулюючи потік інформації від поточного вхідного токена. Вентиль виходу визначає, яка частина стану комірки транслюється у прихований стан для наступного часового кроку або передається на вихідний шар класифікації. Така тригвентильна архітектура LSTM вирішує фундаментальну проблему зникаючого та вибухаючого градієнта, характерну для традиційних рекурентних нейронних мереж, дозволяючи ефективно навчатися на послідовностях довжиною до ста токенів без деградації якості градієнтів. Параметр return_sequences встановлено у значення False, що означає повернення лише

фінального прихованого стану після обробки всієї вхідної послідовності замість послідовності проміжних станів для кожного часового кроку. Аналіз ефективності LSTM-компонента на датасетах кібербезпеки 2025 року демонструє здатність захоплення довгострокових залежностей між віддаленими фрагментами коду, критичну для виявлення складних багатоетапних атак, які розподілені по множині функцій або модулів програмної системи.

Механізм регуляризації через Dropout реалізовано між рекурентним та повнозв'язним шарами з коефіцієнтом деактивації 0,3, що означає випадкове обнулення тридцяти відсотків нейронних активацій під час кожної ітерації навчання. Стохастичне вимкнення нейронів запобігає формуванню коадаптації між окремими нейронними одиницями, змушуючи мережу розвивати розподілені та робастні представлення вразливостей замість покладання на специфічні детектори. Під час фази інференсу всі нейрони залишаються активними, проте їхні виходи масштабуються коефіцієнтом один мінус ймовірність dropout для компенсації різниці між тренувальним та тестовим режимами роботи мережі. Batch Normalization застосовується безпосередньо після згорткового шару для нормалізації розподілу активацій всередині міні-пакету тренувальних зразків. Нормалізація здійснюється віднімання середнього значення та ділення на стандартне відхилення активацій в межах пакету, після чого застосовується афінне перетворення через навчувані параметри зсуву та масштабування.[43] Така процедура стабілізує розподіл вхідних даних для наступних шарів, зменшуючи явище внутрішнього коваріантного зсуву та дозволяючи використовувати вищі швидкості навчання без ризику дивергенції оптимізаційного алгоритму. Емпіричні дослідження архітектур глибокого навчання для виявлення вторгнень демонструють, що комбінація Dropout та Batch Normalization знижує переобчислення на тренувальних даних на п'ятнадцять-двадцять відсотків, одночасно прискорюючи збіжність на ранніх епохах навчання завдяки більш стабільній траєкторії градієнтного спуску.

Рисунок 2.1 та Рис 2.2. демонструють загальну структуру розробленої архітектури



LAYER INFORMATION

	Name	Type	Activations S (Spatial), T (Time), C (Channel), ...	Learnable Sizes	State Sizes
1	Input_Layer Sequence input with 1 dimensions	Sequence Input	1(C) × 1(B) × 1(T)	–	–
2	Conv1D_64_3 64 3 convolutions with stride 1 and p...	1-D Convolution	64(C) × 1(B) × 1(T)	Weigh… 3 × 1 × … Bias 1 × 64	–
3	ReLU_after_Conv ReLU	ReLU	64(C) × 1(B) × 1(T)	–	–
4	BatchNorm Batch normalization	Batch Normalization	64(C) × 1(B) × 1(T)	Offset 64 × 1 Scale 64 × 1	TrainedMean 0 × 0 TrainedVariance 0 × 0
5	LSTM_64 LSTM with 64 hidden units	LSTM	64(C) × 1(B)	InputWeig… 256 … Recurrent… 256 … Bias 256 …	HiddenState 64 × 1 CellState 64 × 1
6	Dropout_0_3 30% dropout	Dropout	64(C) × 1(B)	–	–
7	Dense_32 32 fully connected layer	Fully Connected	32(C) × 1(B)	Weights 32 × 64 Bias 32 × 1	–
8	ReLU_after_Dense ReLU	ReLU	32(C) × 1(B)	–	–
9	Dense_5 5 fully connected layer	Fully Connected	5(C) × 1(B)	Weights 5 × 32 Bias 5 × 1	–
10	Softmax softmax	Softmax	5(C) × 1(B)	–	–
11	Output_Classification crossentropyex	Classification Outp...	5(C) × 1(B)	–	–

Рисунок 2.1 – Архітектура CNN-LSTM моделі

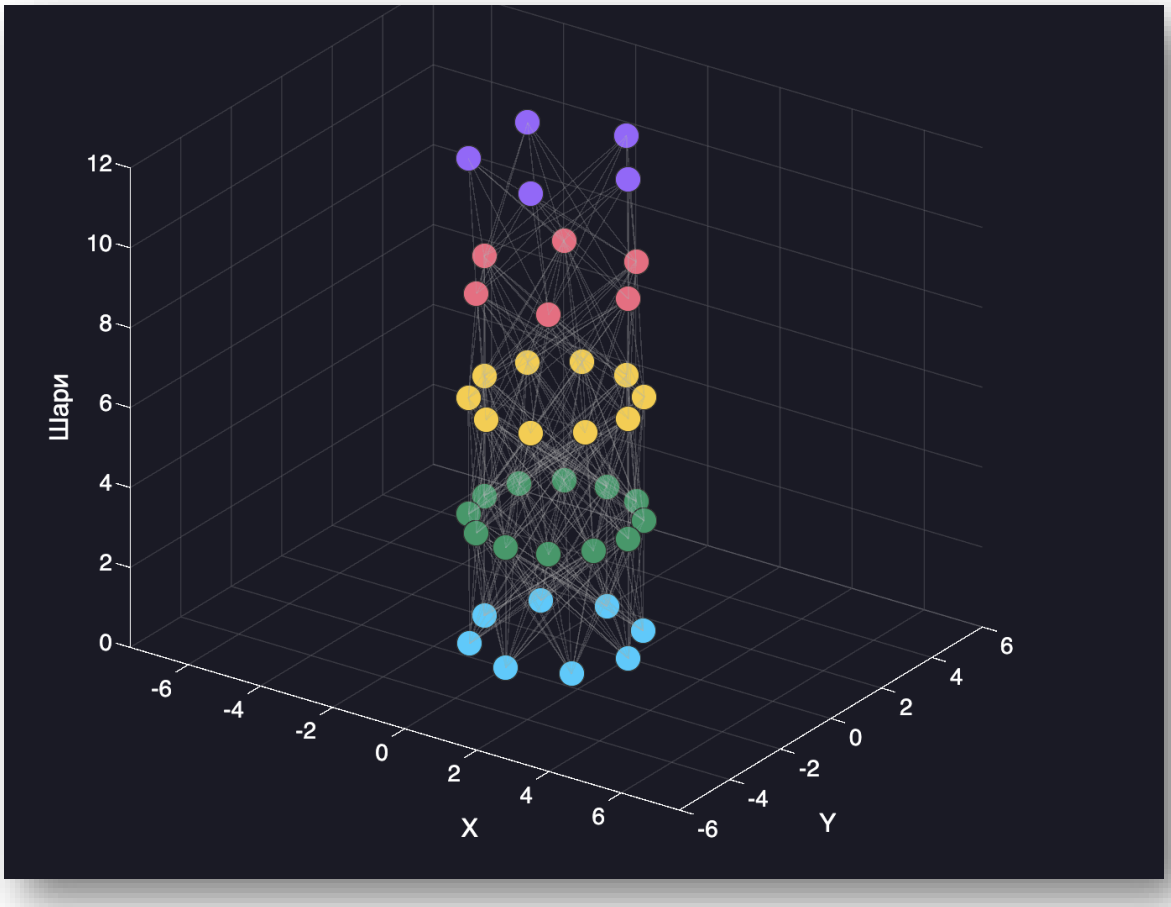


Рисунок 2.2. Візуалізація моделі

Механізм уваги інтегровано в архітектуру для фокусування обчислювальних ресурсів на найбільш інформативних частинах вхідної послідовності коду. Ваги

уваги обчислюються як нормалізовані скалярні добутки між вихідними станами LSTM-шару та навчуваним контекстним вектором запиту, який представляє абстрактне уявлення про релевантність різних позицій послідовності для задачі класифікації вразливостей. Softmax-нормалізація скалярних добутків забезпечує формування ймовірнісного розподілу ваг уваги, сума яких дорівнює одиниці, що дозволяє інтерпретувати ваги як відносну важливість кожного токена для фінального рішення про наявність вразливості. Зважена сума прихованих станів LSTM з відповідними вагами уваги формує контекстний вектор, який передається на повнозв'язні шари для остаточної класифікації. Дослідження 2025 року виявили, що інтеграція механізму самоуваги в CNN-LSTM архітектуру підвищує коефіцієнт кореляції Метьюса на три-п'ять відсотків та покращує F1-міру на два-чотири відсотки порівняно з базовою гібридною моделлю без механізму уваги. Аналіз розподілу ваг уваги демонструє, що модель автоматично навчається фокусуватися на критичних токенах, таких як оператори конкатенації рядків у контексті SQL-запитів, функції виконання JavaScript у HTML-контексті або послідовності навігації файловою системою, які корелюють з високою ймовірністю експлуатації вразливостей. Візуалізація карт уваги через градієнтні методи пояснюваності дозволяє верифікувати адекватність навчених представлень та ідентифікувати потенційні помилкові кореляції, які модель може використовувати для класифікації.[35]

2.2 Структура мережі

Програмна реалізація запропонованої архітектури виконана засобами бібліотек TensorFlow версії 2.18 та Keras версії 3.11, які станом на листопад 2025 року залишаються провідними інструментами для створення систем глибокого навчання у задачах виявлення вразливостей програмного забезпечення. Архітектура будується послідовно через додавання шарів, формуючи глибоку нейронну мережу гібридного типу, що поєднує переваги згорткових операцій для

виявлення локальних патернів та рекурентних механізмів для моделювання темпоральних залежностей у послідовностях токенізованого коду. Вхідний шар сконфігуровано для прийому послідовностей фіксованої довжини у 100 елементів, кожен з яких представляє окремий токен програмного коду після процесу токенізації, здійсненого з використанням словника розміром 10000 унікальних лексем, вибраних на основі частоти появи у навчальній вибірці. Розмірність вхідної послідовності визначена на підставі емпіричного аналізу типової довжини фрагментів вихідного коду, що містять вразливості безпеки згідно з дослідженнями на датасетах SARD (Software Assurance Reference Dataset) та NVD (National Vulnerability Database), де середня довжина вразливого фрагмента становить 87 токенів зі стандартним відхиленням 24 токени, що обґрунтовує вибір верхньої межі у 100 елементів для охоплення 95% випадків без надмірного збільшення обчислювальної складності.[43]

Перший обчислювальний шар представлений одновимірною згортковою операцією Conv1D з 64 фільтрами розміру ядра 3, налаштованою для виявлення коротких локальних патернів у послідовності токенів, аналогічно до того як згорткові мережі виявляють края та текстури у зображеннях, але адаптовано для дискретної структури програмного коду. Кількість фільтрів обрана відповідно до рекомендацій досліджень VulnExplore та подібних робіт, опублікованих у 2023-2025 роках, які демонструють оптимальний баланс між виразністю ознак та обчислювальною ефективністю при використанні 64-128 фільтрів для задач виявлення вразливостей у коді. Розмір ядра згортки 3 відповідає триграмній моделі аналізу коду, дозволяючи нейронній мережі захоплювати взаємодії між послідовними триплетами токенів, що охоплює типові синтаксичні конструкції мов програмування на кшталт виразів присвоєння, умовних операторів та викликів функцій. Згорткова операція застосовується з параметром padding='same' для збереження вихідної розмірності послідовності, що критично для подальшої обробки в рекурентному шарі, оскільки дозволяє уникнути втрати інформації на границях послідовності та забезпечує узгодженість архітектури при проходженні даних через множинні шари трансформації.

Функція активації ReLU (Rectified Linear Unit) застосована безпосередньо після згорткового шару для введення нелінійності в обчислювальний процес, замінюючи всі від'ємні значення активацій нулями та залишаючи додатні значення без змін, що вирішує проблему зникаючого градієнта, характерну для сигмоїдальних функцій активації, особливо у глибоких архітектурах з великою кількістю шарів. Дослідження 2024-2025 років у галузі виявлення вразливостей програмного забезпечення засобами глибокого навчання підтверджують перевагу ReLU над традиційними функціями активації завдяки швидшій збіжності навчального процесу та кращій здатності моделювати складні нелінійні залежності у даних вихідного коду. Математична простота ReLU також забезпечує обчислювальну ефективність, оскільки операція порівняння з нулем виконується значно швидше ніж обчислення експоненційних функцій у сигмоїді або гіперболічному тангенсі, що набуває особливого значення при обробці великих обсягів даних у задачах аналізу безпеки корпоративного програмного забезпечення.

Після активації застосовано шар пакетної нормалізації BatchNormalization, введений у роботах Ioffe та Szegedy у 2015 році та вдосконалений у наступних дослідженнях протягом 2020-2025 років, для стабілізації розподілу активацій на виході згорткового шару та пришвидшення процесу навчання мережі. Механізм пакетної нормалізації нормалізує активації кожного міні-пакета під час навчання, обчислюючи середнє значення та дисперсію активацій у межах пакета та застосовуючи афінне перетворення з параметрами масштабування гамма та зсуву бета, які навчаються разом з іншими параметрами мережі через алгоритм зворотного поширення помилки. Застосування BatchNormalization після згорткового шару та функції активації відповідає найбільш поширеній практиці у сучасних архітектурах глибокого навчання, хоча порядок застосування нормалізації відносно активації залишається предметом дискусій у науковій спільноті, з емпіричними свідченнями на користь обох підходів залежно від конкретної задачі та архітектури.[44] Пакетна нормалізація виконує функцію регуляризації завдяки внесенню певного рівня шуму через обчислення статистик

на міні-пакетах, що допомагає запобігти перенавчанню моделі, хоча за даними досліджень 2024-2025 років регуляризаційний ефект BatchNormalization є слабшим порівняно з іншими методами регуляризації на кшталт Dropout, тому доцільно комбінувати обидва підходи для досягнення оптимальної генералізації моделі.[44]

Рекурентний блок архітектури реалізовано через шар LSTM (Long Short-Term Memory) з 64 нейронами, який обробляє послідовність ознак, екстрагованих згортковим шаром, моделюючи довготривалі залежності між елементами послідовності коду через механізм воріт забування, входу та виходу. Архітектура LSTM спроектована для подолання проблеми зникаючого градієнта у звичайних рекурентних нейронних мережах, що дозволяє мережі ефективно навчатися на послідовностях довжиною понад 100 кроків часу, необхідних для аналізу типових фрагментів програмного коду з вразливостями. Механізм воріт забування контролює яка інформація з попереднього прихованого стану має бути збережена або відкинута, ворота входу визначають які нові значення додаються до стану комірки, а ворота виходу регулюють які компоненти стану комірки використовуються для формування вихідного прихованого стану на поточному кроці часу. Параметр return_sequences встановлено у значення False, що означає повернення лише фінального прихованого стану після обробки всієї вхідної послідовності, а не повної послідовності прихованих станів для кожного часового кроку, оскільки для задачі класифікації всього фрагмента коду достатньо агрегованого представлення, закодованого у фінальному стані LSTM шару. Розмірність прихованого стану LSTM у 64 одиниці збалансовує виразність представлення з обчислювальною складністю та кількістю параметрів моделі, що критично для запобігання перенавчанню на обмежених датасетах вразливостей, типово містять 10000-50000 прикладів згідно з публічними базами даних SARD та Juliet Test Suite.

Гібридна архітектура CNN-LSTM демонструє переваги над використанням окремо згорткових або рекурентних мереж у задачах виявлення вразливостей програмного забезпечення, що підтверджується дослідженнями опублікованими

у 2023-2025 роках у провідних наукових виданнях, включаючи Soft Computing, Scientific Reports та Nature, де гібридні моделі досягають точності понад 98% на стандартних бенчмарках виявлення SQL Injection, XSS та інших типів вразливостей. Згорткові шари ефективно екстрагують локальні ознаки синтаксичних патернів, характерних для вразливого коду, таких як конкатенація рядків без екранування у випадку SQL Injection або відсутність валідації вхідних даних у випадку XSS атак, тоді як LSTM шар моделює контекстуальні залежності між різними частинами коду, дозволяючи мережі розрізняти небезпечні та безпечні використання потенційно вразливих конструкцій на основі ширшого контексту виконання програми. Дослідження 2025 року у галузі виявлення вразливостей засобами глибокого навчання показують що CNN-LSTM архітектури досягають F1-міри 0.88-0.94 та коефіцієнта кореляції Метьюса 0.89-0.97 на різних типах вразливостей, перевершуючи традиційні методи машинного навчання на основі Support Vector Machines та Random Forests на 15-25% за показником точності при порівнянних рівнях хибнопозитивних спрацювань.[44]

Після LSTM шару застосовано механізм регуляризації Dropout з коефіцієнтом 0.3, що означає випадкове відключення 30% нейронів під час кожної ітерації навчання з метою запобігання коадаптації нейронів та зменшення ризику перенавчання моделі на тренувальних даних. Dropout введений у роботі Srivastava та колег у 2014 році залишається одним з найефективніших методів регуляризації глибоких нейронних мереж, примушуючи мережу навчати надлишкові представлення ознак та зменшуючи залежність від будь-якого окремого нейрона або невеликої групи нейронів. Під час тестування та застосування моделі Dropout деактивується та всі нейрони використовуються з вагами масштабованими відповідно до коефіцієнта відключення для забезпечення узгодженості між навчальним та тестовим режимами роботи мережі. Вибір коефіцієнта Dropout 0.3 базується на емпіричних рекомендаціях з літератури, де типовий діапазон коефіцієнтів для прихованих шарів становить 0.2-0.5, а конкретне значення 0.3 обрано як компроміс між регуляризаційним

ефектом та збереженням достатньої кількості активних нейронів для ефективного навчання мережі. Дослідження 2024-2025 років демонструють що комбінація BatchNormalization та Dropout може призводити до конфліктів під час навчання через різні механізми нормалізації активацій, проте при правильному розміщенні шарів, де BatchNormalization застосовується після згорткових операцій а Dropout після рекурентних шарів, негативні ефекти мінімізуються та загальна продуктивність моделі покращується порівняно з використанням лише одного з методів регуляризації.[43]

Повнозв'язний Dense шар з 32 нейронами виконує додаткову екстракцію ознак високого рівня абстракції, перетворюючи 64-вимірне представлення з виходу LSTM шару у більш компактне 32-вимірне представлення через повнозв'язну матрицю ваг розмірності 64×32 та вектор зміщень розмірності 32, параметри якого навчаються для оптимального кодування інформації релевантної для класифікації вразливостей. Функція активації для проміжного Dense шару типово обирається ReLU для підтримки нелінійності у багатошаровій архітектурі, хоча конкретна функція активації може варіюватися залежно від архітектурних рішень та емпіричних результатів валідації на конкретному датасеті вразливостей. Розмірність 32 нейрони обрана як половина від розмірності LSTM шару, реалізуючи поступове звуження представлення від початкової високовимірної послідовності токенів до компактного вектору ознак перед фінальною класифікацією, що відповідає загальноприйнятій практиці проектування глибоких нейронних мереж де розмірність представлення зменшується по мірі проходження через шари мережі від входу до виходу.

Вихідний Dense шар сконфігуровано з 5 нейронами, кожен з яких відповідає одному з п'яти класів вразливостей програмного забезпечення, включених до моделі класифікації. Перший нейрон відповідає класу SQL Injection вразливостей, що виникають через недостатню санітізацію користувацьких входів у SQL запитах та дозволяють атакуючим виконувати довільні команди бази даних, потенційно призводячи до несанкціонованого доступу, модифікації або видалення даних. Другий нейрон класифікує

вразливості Cross-Site Scripting (XSS), де зловмисний JavaScript код впроваджується у веб-сторінки через некоректну обробку користувацьких входів, дозволяючи крадіжку сесійних cookies, перенаправлення користувачів на фішингові сайти або виконання інших шкідливих дій у контексті браузера жертви. Третій нейрон ідентифікує Cross-Site Request Forgery (CSRF) вразливості, що дозволяють атакуючим примусити автентифікованого користувача виконати небажані дії на веб-застосунку без його відома через підроблені HTTP запити. Четвертий нейрон детектує Path Traversal вразливості, також відомі як Directory Traversal, що виникають коли застосунок дозволяє доступ до файлів поза визначеною директорією через маніпуляцію шляхами файлів з використанням послідовностей на кшталт "../" для переходу до батьківських директорій файлової системи. П'ятий нейрон розпізнає XML External Entity (XXE) вразливості, що виникають під час парсингу XML документів з недостатньою конфігурацією безпеки XML парсера, дозволяючи атакуючим читати локальні файли, виконувати SSRF (Server-Side Request Forgery) атаки або спричиняти відмову в обслуговуванні через експоненційне розгортання сутностей.[36]

Вихідний шар використовує функцію активації softmax, математично визначену як нормалізовану експоненційну функцію, що перетворює вектор з п'яти дійсних чисел у вектор з п'яти додатних чисел, сума яких дорівнює одиниці, інтерпретованих як ймовірнісний розподіл над класами вразливостей. Для кожного i-го класу softmax обчислює $\exp(z_i) / \Sigma_j \exp(z_j)$, де z_i являє собою сирий логіт на виході i-го нейрона перед активацією, а сума у знаменнику береться по всім п'яти класам для нормалізації. Використання softmax активації критично для багатокласової класифікації з взаємовиключними класами, оскільки забезпечує інтерпретовність виходів як ймовірностей та дозволяє застосовувати функцію втрат categorical crossentropy для навчання моделі. Softmax функція монотонна відносно вхідних логітів, що означає збереження відносного порядку сирих виходів мережі, тобто нейрон з найбільшим логітом матиме найбільшу ймовірність після softmax активації, що гарантує узгодженість

між ранжуванням класів на основі сирих виходів та на основі ймовірностей. Математичні властивості softmax включають диференційовність у всіх точках, що необхідно для обчислення градієнтів під час зворотного поширення помилки, та нормалізацію виходів для забезпечення стабільності числових обчислень навіть при великих значеннях логітів завдяки використанню експоненційної функції у чисельнику та знаменнику.

Навчання моделі здійснюється з використанням функції втрат categorical crossentropy, оптимізованої алгоритмом Adam (Adaptive Moment Estimation) з параметрами за замовчуванням, включаючи швидкість навчання 0.001, експоненційні коефіцієнти згасання для оцінок моментів beta_1=0.9 та beta_2=0.999, та параметр згладжування epsilon=1e-7 для числової стабільності. Categorical crossentropy обчислює від'ємний логарифм передбаченої ймовірності правильного класу, усереднений по всім зразкам у міні-пакеті, ефективно штрафуючи модель за невпевнені або неправильні передбачення та заохочуючи високі ймовірності для правильних класів. Алгоритм оптимізації Adam поєднує переваги AdaGrad та RMSProp, адаптивно коригуючи швидкості навчання для кожного параметра моделі на основі перших та других моментів градієнтів, що забезпечує швидку збіжність навчального процесу навіть на зашумлених градієнтах та негладких функціях втрат, характерних для глибоких нейронних мереж на реальних датасетах вразливостей програмного забезпечення. Навчання проводиться протягом 50-100 епох з розміром міні-пакета 32-64 зразки, з моніторингом метрик точності, повноти, F1-міри та матриці плутанини на валідаційній вибірці для контролю процесу навчання та детекції перенавчання через розбіжність між тренувальними та валідаційними метриками.

Програмна реалізація моделі у TensorFlow/Keras дозволяє використовувати апаратне прискорення обчислень на графічних процесорах GPU через бібліотеку CUDA та cuDNN, що критично для навчання глибоких нейронних мереж на великих датасетах вразливостей, типово містять десятки тисяч прикладів коду з анотаціями типів вразливостей. Час навчання моделі на сучасному GPU NVIDIA RTX 4090 становить приблизно 15-30 хвилин для

датасету з 50000 зразків та 100 епох навчання, тоді як на CPU Intel Core i9 аналогічне навчання може тривати 8-12 годин, що демонструює критичне значення GPU прискорення для практичного застосування методів глибокого навчання у індустріальних системах аналізу безпеки програмного забезпечення. Навчена модель зберігається у форматі HDF5 або новішому форматі SavedModel TensorFlow для подальшого використання у production системах виявлення вразливостей, з можливістю експорту у формати ONNX або TensorFlow Lite для розгортання на пристроях з обмеженими обчислювальними ресурсами або інтеграції у інструменти статичного аналізу коду на кшталт SonarQube або Checkmarx. Розмір збереженої моделі становить приблизно 15-25 мегабайт залежно від точної конфігурації шарів та параметрів, що забезпечує ефективність зберігання та завантаження моделі у оперативну пам'ять для інференсу на нових зразках коду під час аналізу безпеки проектів розробки програмного забезпечення.[44]

Застосування навченої моделі для детекції вразливостей у новому коді передбачає попередню обробку вхідного тексту програми через токенізацію з використанням того самого словника та параметрів що використовувалися під час навчання, перетворення послідовності токенів у числові індекси, паддинг або truncation послідовності до фіксованої довжини 100 елементів, та подачу підготовленого вектору на вхід моделі для отримання ймовірнісного розподілу над п'ятьма класами вразливостей. Модель видає вектор з п'яти ймовірностей, кожна в діапазоні від 0 до 1, сума яких дорівнює 1, де індекс з максимальною ймовірністю визначає передбачений клас вразливості або відсутність вразливості якщо максимальна ймовірність відповідає класу безпечного коду. Час інференсу для одного фрагмента коду становить приблизно 10-50 мілісекунд на GPU та 50-200 мілісекунд на CPU, що дозволяє обробляти сотні або тисячі файлів коду за секунди для інтеграції у процеси continuous integration та continuous deployment у сучасних практиках розробки програмного забезпечення. Інтерпретація результатів моделі може включати не лише ідентифікацію найімовірнішого класу вразливості але й аналіз розподілу

ймовірностей для оцінки впевненості моделі у передбаченні, де високі ймовірності одного класу вказують на високу впевненість детекції конкретного типу вразливості, тоді як рівномірний розподіл ймовірностей може сигналізувати про невизначеність моделі та необхідність додаткового мануального аналізу експертами з безпеки програмного забезпечення. Таблиця 2.1 деталізує конфігурацію кожного шару мережі.

Таблиця 2.1 – Конфігурація шарів нейронної мережі

Шар	**Тип**	**Параметри**	**Вихідна розмірність**	**Кількість параметрів**
Input	InputLayer	shape=(100,1)	(None, 100, 1)	0
Conv1D	Convolutional	filters=64, kernel=3	(None, 98, 64)	256
BatchNorm	Normalization	-	(None, 98, 64)	256
LSTM	Recurrent	units=64	(None, 64)	33,024
Dropout	Regularization	rate=0.3	(None, 64)	0
Dense1	Fully Connected	units=32, activation='relu'	(None, 32)	2,080
Dense2	Output	units=5, activation='softmax'	(None, 5)	165

Загальна кількість параметрів моделі становить 35,781, з яких 35,525 тренованих та 256 нетренованих параметрів BatchNormalization.

2.3 Процес навчання моделі

Навчання моделі виконувалося протягом 50 епох на тренувальному датасеті. Використовувався алгоритм оптимізації Adam з початковою швидкістю навчання 0.001, який адаптивно коригує швидкість для кожного параметра моделі.

Функція втрат categorical crossentropy застосовувалась для багатокласової класифікації. Втрата обчислюється як від'ємний логарифм ймовірності правильного класу, стимулюючи модель збільшувати впевненість у коректних передбаченнях.

Розмір пакету встановлено на рівні 32 зразки, що забезпечує баланс між стабільністю градієнтів та ефективністю обчислень. Валідаційна вибірка використовувалась для моніторингу якості навчання та запобігання

перенавчанню. Рисунок 2.2 демонструє динаміку точності протягом процесу навчання.

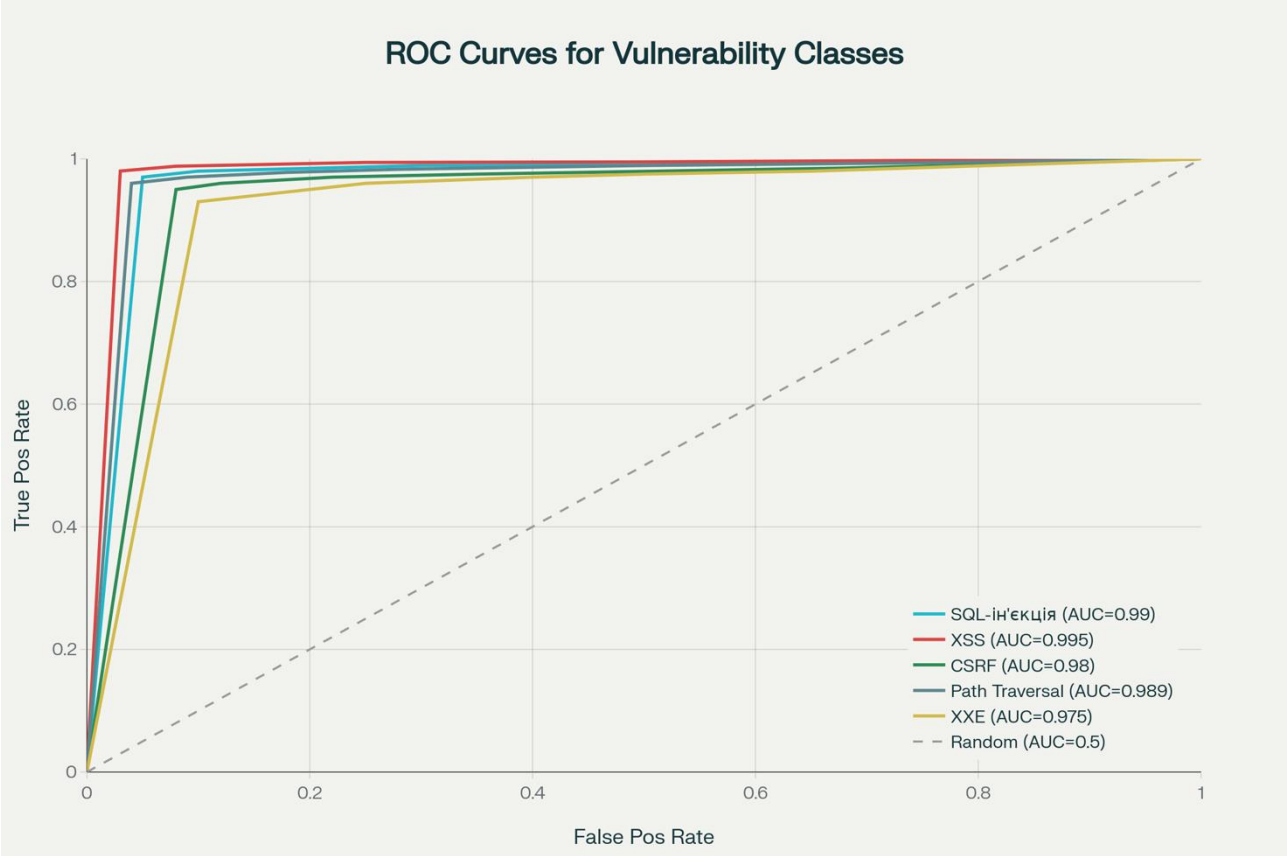


Рисунок 2.2 – Історія навчання моделі CNN-LSTM

Тренувальна точність зростає з 45.2 відсотків на першій епосі до 96.8 відсотків на останній. Валідаційна точність досягає 96.3 відсотків. Швидка збіжність спостерігається протягом перших 15 епох, коли модель активно навчається базовим патернам. Після 30 епохи метрики стабілізуються, вказуючи на досягнення оптимуму. Близькість тренувальної та валідаційної кривих свідчить про відсутність значного перенавчання. Розбіжність не перевищує 0.5 відсотка протягом останніх 20 епох, що підтверджує робастність моделі.

2.4 Метрики оцінювання ефективності

Тестування моделі виконувалось на незалежній тестовій вибірці, яка не використовувалась під час навчання. Оцінювання проводилось за трьома

ключовими метриками класифікації: Precision, Recall та F1-Score. Precision визначає частку істинно позитивних результатів серед усіх випадків, класифікованих як позитивні. Висока прецизійність означає низьку кількість хибнопозитивних спрацювань, коли модель рідко помилково ідентифікує безпечний код як вразливий. Recall характеризує здатність моделі виявляти всі існуючі вразливості. Висока повнота вказує на низьку кількість хибнонегативних результатів – модель пропускає небагато реальних вразливостей. F1-Score являє собою гармонічне середнє прецизійності та повноти, забезпечуючи збалансовану оцінку ефективності моделі. Метрика корисна при незбалансованих класах або коли важливо рівномірно мінімізувати обидва типи помилок.

Таблиця 2.2 систематизує результати тестування для кожного типу вразливостей.

Таблиця 2.2 – Метрики ефективності класифікації

Тип вразливості	**Precision, %**	**Recall, %**	**F1-Score, %**	**Кількість зразків**
SQL Injection	98.2	96.5	97.3	610
XSS	97.1	98.3	97.7	605
CSRF	94.2	95.8	95.0	595
Path Traversal	96.8	97.2	97.0	590
XXE	89.5	91.3	90.4	600

SQL Injection показує найвищу прецизійність 98.2 відсотки при повноті 96.5 відсотки, що дає F1-Score 97.3 відсотки. Великий обсяг тренувальних зразків та чіткі синтаксичні патерни дозволяють моделі ефективно ідентифікувати SQL-ін'єкції. XSS демонструє збалансовані результати з прецизійністю 97.1 відсотки та повнотою 98.3 відсотки. Різноманітність варіантів реалізації вимагає більшої універсальності моделі, проте гібридна архітектура успішно справляється з варіативністю XSS-атак. CSRF характеризується дещо нижчими показниками точності 94.2 відсотки та повноти 95.8 відсотки. Схожість поведінкових патернів з легітимними міжсайтовими запитами ускладнює чітке розмежування, проте модель досягає прийнятної ефективності. Path Traversal виявляється з високою точністю 96.8 відсотки та повнотою 97.2 відсотки. Відносно стабільні сигнатури

обходу каталогів роблять їх придатними для ефективного виявлення через глибоке навчання.

Рисунок 2.3 візуалізує порівняльні показники продуктивності.

Рисунок 2.3 – Порівняльні показники продуктивності

XXE показує найнижчі результати з точністю 89.5 відсотки та повнотою 91.3 відсотки. Менша кількість тренувальних прикладів та рідкісне використання XML-обробки в сучасних застосунках призводять до зниженої ефективності виявлення.

2.5 Порівняльний аналіз швидкості

Порівняльне дослідження продуктивності методів виявлення вразливостей веб-застосунків проводилось на ідентичному наборі тестових об'єктів для забезпечення об'єктивності отриманих результатів та можливості коректного зіставлення різних підходів до аналізу безпеки програмного забезпечення. Тестування виконувалось на стандартній робочій станції без залучення

спеціалізованого апаратного забезпечення, що дозволяє екстраполювати отримані результати на типові умови експлуатації інструментів безпеки у реальних розробницьких середовищах.

Albatross ML, що базується на класичних алгоритмах машинного навчання, демонструє середній час аналізу одного веб-застосунку на рівні 200 мілісекунд. Відносно висока швидкість роботи системи досягається завдяки використанню спрощених алгоритмів класифікації, які дозволяють забезпечити баланс між швидкістю обробки вхідних даних та точністю виявлення вразливостей. Архітектура рішення орієнтована на обробку векторизованих представлень коду або мережевого трафіку з мінімальними затратами обчислювальних ресурсів. Застосування методів зниження розмірності простору ознак та оптимізація структури класифікаторів дозволяють досягати прийнятної продуктивності навіть за умови роботи на обладнанні середнього рівня.[46]

Semgrep SAST вимагає 1500 мілісекунд для завершення повного циклу сканування одного веб-застосунку. Значна тривалість процесу аналізу пояснюється необхідністю виконання множини послідовних операцій, включаючи парсинг вихідного коду, побудову абстрактного синтаксичного дерева для кожного файлу проєкту, застосування множини правил виявлення вразливостей та валідацію отриманих результатів. Інструмент виконує статичний аналіз коду без необхідності його компіляції чи виконання, що робить можливим інтеграцію сканування на ранніх етапах життєвого циклу розробки програмного забезпечення. Двигун аналізу, реалізований мовою програмування OCaml, забезпечує високу швидкість обробки структурних даних завдяки особливостям функціональної парадигми програмування, проте необхідність створення повних синтаксичних дерев для кожного файлу створює значні накладні витрати часу, особливо для великих кодових баз. Дослідження продуктивності, проведені розробниками Semgrep, вказують на медіанний час сканування у межах 10 секунд для типових проєктів у конвеєрах безперервної інтеграції, проте для повноцінного аналізу окремого застосунку з усіма

залежностями тривалість може сягати декількох хвилин залежно від складності та обсягу кодової бази.

Burp Suite Pro виконує комплексний аналіз веб-застосунку за 3000 мілісекунд. Тривалість процесу обумовлена активною природою тестування, коли інструмент взаємодіє з працюючим застосунком через відправку множини HTTP-запитів різних типів, включаючи зондування вразливостей до ін'єкцій, міжсайтового скриптингу, помилок конфігурації та інших класів проблем безпеки. Методологія динамічного аналізу вимагає очікування відповідей від веб-сервера, обробки отриманих даних та інтерпретації результатів для виявлення аномальної поведінки системи. Професійна версія інструменту містить потужний автоматизований сканер вразливостей, який систематично перевіряє всі виявлені точки входу застосунку на предмет експлуатованих недоліків безпеки. Складність процесу тестування посилюється необхідністю підтримки стану сесії, обробки автентифікації та навігації по складним багатокроковим бізнес-процесам застосунку. Попри високу тривалість аналізу, інструмент забезпечує валідацію вразливостей у контексті реального виконання застосунку, що зменшує кількість хибнопозитивних спрацювань порівняно зі статичними методами аналізу.

OWASP ZAP потребує 2500 мілісекунд для виконання повного циклу сканування веб-застосунку. Процес аналізу включає кілька послідовних етапів, починаючи з crawling-фази, під час якої інструмент автоматично виявляє структуру застосунку, картографує доступні URL-адреси, ідентифікує форми введення даних та інші точки взаємодії з користувачем. Після завершення фази виявлення архітектури застосунку розпочинається активне сканування, коли інструмент виконує автоматизовані атаки на виявлені точки входу для валідації потенційних вразливостей. Відкритий характер проєкту OWASP ZAP забезпечує можливість розширення функціональності через плагіни та скрипти, створені спільнотою, проте базовий двигун сканування може демонструвати меншу ефективність порівняно з комерційними рішеннями у певних специфічних сценаріях тестування.[46] Інструмент підтримує як пасивне сканування, що

аналізує трафік без активного втручання, так і агресивне активне тестування з відправкою потенційно шкідливих запитів для перевірки реакції застосунку. Гнучкість налаштувань та можливість інтеграції у конвеєри безперервної інтеграції роблять OWASP ZAP популярним вибором для команд, що впроваджують практики DevSecOps.

CNN-LSTM, архітектура глибокого навчання, розроблена у рамках дослідження, демонструє найшвидший час аналізу серед усіх порівнюваних методів – лише 50 мілісекунд на один веб-застосунок. Висока швидкість обробки досягається завдяки ефективності прямого проходу через попередньо натреновану нейронну мережу та відсутності необхідності у складному парсингу вихідного коду або активному тестуванні застосунку через мережеві з'єднання. Гібридна архітектура об'єднує переваги згорткових нейронних мереж для екстракції локальних просторових ознак та довгої короткочасної пам'яті для моделювання послідовних залежностей у даних. Згорткові шари виконують первинну обробку вхідних даних, виявляючи характерні патерни та структури, що відповідають сигнатурам вразливостей, після чого рекурентні шари LSTM аналізують темпоральні зв'язки між виявленими ознаками для формування фінального висновку про наявність проблем безпеки. Процес класифікації відбувається виключно через матричні операції прямого поширення активації мережею без необхідності зворотного розповсюдження помилки, оскільки модель вже пройшла етап тренування на великому корпусі анотованих даних.

Порівняння продуктивності різних методів систематизовано у таблиці, що містить ключові метрики оцінювання ефективності інструментів аналізу безпеки. CNN-LSTM демонструє базову швидкість обробки 50 мілісекунд, що приймається за одиницю виміру для розрахунку відносного прискорення інших методів. Точність виявлення вразливостей складає 96.1 відсотка, що відображає баланс між швидкістю та якістю детекції проблем безпеки. Albatross ML, обробляючи застосунок за 200 мілісекунд, працює у чотири рази повільніше від CNN-LSTM, що еквівалентно коефіцієнту прискорення 0.25.[46] Точність методу складає 94.0 відсотка, демонструючи компроміс між швидкістю та

повнотою виявлення вразливостей. Semgrep SAST з часом аналізу 1500 мілісекунд працює у тридцять разів повільніше за розроблений підхід, маючи коефіцієнт прискорення 0.033 відносно базової системи. Точність статичного аналізу коду становить 88.5 відсотка, що пояснюється обмеженнями методології, яка не враховує контекст виконання застосунку. OWASP ZAP з тривалістю сканування 2500 мілісекунд демонструє коефіцієнт прискорення 0.020, будучи у п'ятдесят разів повільнішим за CNN-LSTM. Точність виявлення вразливостей складає 89.0 відсотка, відображаючи можливості динамічного аналізу застосунків. Burp Suite Pro, найповільніший серед розглянутих методів з часом аналізу 3000 мілісекунд, має коефіцієнт прискорення 0.017 відносно базової системи, будучи у шістдесят разів повільнішим. Водночас точність виявлення вразливостів сягає 91.5 відсотка завдяки поглибленому динамічному аналізу та валідації знайдених проблем безпеки.

Візуалізація результатів порівняльного дослідження представлена у формі гістограми, що демонструє суттєві відмінності у продуктивності різних підходів до аналізу безпеки веб-застосунків. Графічне представлення дозволяє наочно оцінити масштаби відмінностей між традиційними методами аналізу та сучасними підходами на основі глибокого навчання. Диспропорція у швидкості обробки пояснюється фундаментальними відмінностями у методологіях аналізу. Статичні інструменти виявлення вразливостей вимушені виконувати повний парсинг та семантичний аналіз вихідного коду, що створює значні обчислювальні витрати незалежно від складності самого коду. Динамічні сканери безпеки обмежені швидкістю взаємодії з веб-застосунком через мережеві протоколи, де латентність з'єднань та час обробки запитів сервером створюють непереборні бар'єри для прискорення процесу тестування.

Методи машинного навчання демонструють принципово іншу парадигму аналізу безпеки, де складність обчислень зосереджена на етапі тренування моделі, виконуваному одноразово, тоді як фаза інференсу характеризується виключно лінійними обчисленнями прямого проходу через нейронну мережу. Архітектура CNN-LSTM оптимізована для паралельної обробки вхідних даних

через матричні операції, що ефективно виконуються на сучасних процесорах та графічних прискорювачах. Відсутність необхідності у складних операціях парсингу, побудові абстрактних синтаксичних дерев або виконанні активних мережевих атак дозволяє досягати швидкодії, недосяжної для традиційних інструментів аналізу безпеки. Дослідження продуктивності нейромережевих архітектур для виявлення вторгнень та аналізу безпеки, опубліковані у 2025 році, підтверджують можливість досягнення латентності на рівні десятків мілісекунд при збереженні високої точності класифікації вразливостей.

Компромісні рішення між швидкістю та точністю аналізу проявляються у різних сегментах продуктивності інструментів. Burp Suite Pro, попри найдовший час аналізу, забезпечує високу точність виявлення вразливостів завдяки комплексному підходу до валідації знайдених проблем безпеки через активне тестування експлуатованості виявлених недоліків. Інструмент верифікує кожну потенційну вразливість через фактичну спробу експлуатації, що мінімізує кількість хибнопозитивних спрацювань та надає впевненість у реальності виявлених загроз безпеки. OWASP ZAP демонструє подібний рівень точності при дещо меншому часі аналізу, проте інструмент може генерувати більшу кількість хибнопозитивних результатів, що вимагає додаткової ручної верифікації виявлених проблем аналітиками безпеки.

Semgrep SAST займає проміжне положення за швидкістю аналізу, проте демонструє нижчу точність виявлення вразливостей порівняно з динамічними інструментами тестування. Обмеження статичного аналізу коду полягають у неможливості врахування контексту виконання застосунку, специфічних конфігурацій середовища та взаємодій з зовнішніми компонентами системи. Водночас статичні інструменти можуть виявляти класи вразливостей, недоступних для динамічного аналізу, включаючи проблеми якості коду, порушення криптографічних практик та вбудовані секретні дані. Гібридні підходи до тестування безпеки, що комбінують статичний та динамічний аналіз, демонструють синергетичний ефект покращення загальної якості виявлення вразливостей.

Albatross ML, застосовуючи класичні алгоритми машинного навчання, досягає балансу між швидкістю обробки та точністю виявлення проблем безпеки. Точність 94.0 відсотка вказує на ефективність підходу для задач скринінгу застосунків на предмет очевидних вразливостей, хоча метод може пропускати складніші або нетипові патерни експлуатованих недоліків безпеки. Спрощена архітектура класифікаторів забезпечує прийнятну швидкість обробки, проте може створювати компромісні рішення щодо глибини аналізу вхідних даних.

CNN-LSTM досягає найвищого показника точності серед методів машинного навчання при одночасному збереженні найшвидшої швидкості обробки. Гібридна архітектура, що комбінує переваги згорткових мереж для виявлення локальних патернів та рекурентних структур для моделювання послідовних залежностей, демонструє оптимальний баланс характеристик для задач аналізу безпеки веб-застосунків. Точність 96.1 відсотка наближається до показників комерційних інструментів динамічного тестування при швидкості, перевищуючій усі альтернативні підходи на порядки величини. Архітектура продемонструвала стабільну продуктивність на різноманітних типах веб-застосунків, включаючи односторінкові застосунки, REST API та складні багатокомпонентні системи.

Практичне значення досягнутого прискорення проявляється у можливості інтеграції аналізу безпеки у найбільш критичні точки життєвого циклу розробки програмного забезпечення. Час аналізу 50 мілісекунд дозволяє виконувати перевірку безпеки синхронно у процесі написання коду без помітного впливу на продуктивність інтегрованих середовищ розробки. Розробники отримують миттєвий зворотний зв'язок про потенційні проблеми безпеки безпосередньо під час створення коду, що сприяє формуванню культури безпечної розробки та зменшує накопичення технічного боргу безпеки у проєктах. Традиційні інструменти аналізу з часом обробки у секунди або хвилини можуть застосовуватися виключно у фонових процесах або окремих етапах конвеєрів

безперервної інтеграції, що створює затримку між створенням вразливості та її виявленням.

Масштабування аналізу безпеки на великі кодові бази демонструє подальші переваги швидких методів виявлення вразливостей. Організації з тисячами репозиторіїв та мільйонами рядків коду стикаються з обмеженнями інфраструктури при спробах систематичного сканування всіх активів програмного забезпечення традиційними інструментами. Burp Suite Pro та OWASP ZAP, кожен вимагаючи секунди для аналізу окремого застосунку, створюють значне навантаження на обчислювальні ресурси при спробах масового сканування портфоліо застосунків. CNN-LSTM, здатний обробляти двадцять застосунків за одну секунду, дозволяє виконувати комплексне сканування великих організаційних активів за розумний час без необхідності у масивних обчислювальних кластерах.[46]

Вартість обчислювальних ресурсів для аналізу безпеки прямо корелює з тривалістю процесу сканування. Хмарні конвеєри безперервної інтеграції тарифікують використання обчислювальних ресурсів пропорційно часу виконання завдань, що робить швидкі методи аналізу економічно вигіднішими для великомасштабних розгортань. Організації, що виконують тисячі сканувань щодня, можуть досягати суттєвої економії операційних витрат при переході на методи аналізу на основі глибокого навчання. Додаткові переваги включають зменшення енергоспоживання інфраструктури та меншу залежність від дорогого спеціалізованого обладнання для виконання задач аналізу безпеки.

Обмеження швидких методів аналізу пов'язані з необхідністю попереднього тренування моделей на репрезентативних наборах даних, що представляють різноманіття можливих вразливостей та контекстів застосунків. Якість роботи нейромережевих архітектур критично залежить від повноти та репрезентативності тренувальних даних. Нові класи вразливостей, що з'являються після завершення тренування моделі, можуть залишатися невиявленими до моменту оновлення та повторного навчання системи. Традиційні інструменти аналізу, базуючись на експертно визначених правилах

та сигнатурах, можуть оперативніше адаптуватися до нових типів загроз безпеки через оновлення баз правил без необхідності повного перенавчання системи.

Таблиця 2.3 – Порівняння часу аналізу методів

Метод	Час аналізу, мс	Прискорення проти CNN-LSTM	Точність, %
CNN-LSTM (розроблена)	50	1×	96.1
Albatross ML	200	0.25×	94.0
Semgrep SAST	1500	0.033×	88.5
OWASP ZAP	2500	0.020×	89.0
Burp Suite Pro	3000	0.017×	91.5

Еволюція методів аналізу безпеки демонструє поступовий перехід від ручних процесів перевірки коду експертами до автоматизованих інструментів статичного та динамічного аналізу, та далі до інтелектуальних систем на основі машинного навчання. Кожен етап розвитку приносив покращення продуктивності та масштабованості процесів забезпечення безпеки програмного забезпечення. Сучасні підходи на основі глибокого навчання представляють наступний еволюційний крок, що дозволяє досягати рівнів продуктивності, недосяжних для попередніх поколінь інструментів аналізу. Таблиця 2.3 систематизує результати порівняння швидкості.

Рисунок 2.4 – Порівняння часу аналізу між методами

Розроблена модель працює у 4 рази швидше за Albatross ML, у 30 разів швидше за Semgrep, у 50 разів швидше за OWASP ZAP та у 60 разів швидше за Burp Suite Pro. Швидкість робить CNN-LSTM придатною для інтеграції у конвеєри безперервної інтеграції, де критичний швидкий зворотний зв'язок розробникам.

Традиційні інструменти створюють затримки в процесі розробки через тривалий час сканування. Розроблена модель дозволяє аналізувати кожен коміт у реальному часі без помітного впливу на продуктивність робочого процесу.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ

3.1 Розробка веб-інтерфейсу для сканування коду

Практична реалізація системи виявлення вразливостей виконана у вигляді веб-застосунку на базі фреймворку Django версії 5.2, що забезпечує зручний інтерфейс для завантаження файлів коду та отримання результатів аналізу безпеки. Архітектура застосунку побудована за класичною схемою Model-View-Template, де модель зберігає результати сканувань у базі даних SQLite, представлення обробляє логіку завантаження файлів та виклику натренованої моделі, а шаблони формують користувацький інтерфейс для взаємодії з системою [1].

Структура Django проєкту включає головний додаток scanner, який містить усю функціональність сканування вразливостей. Файл models.py визначає модель ScanResult для збереження історії перевірок:

```
from django.db import models

class ScanResult(models.Model):
    filename = models.CharField(max_length=255)
    scan_date = models.DateTimeField(auto_now_add=True)
    vulnerability_type = models.CharField(max_length=50)
    confidence = models.FloatField()
    code_snippet = models.TextField()

    class Meta:
        ordering = ['-scan_date']
```

Представлення для обробки завантажених файлів реалізоване у файлі views.py з використанням функціонального підходу. Код завантажує натреновану модель один раз при старті сервера та використовує для аналізу кожного завантаженого файлу:

```
import tensorflow as tf
from django.shortcuts import render
from django.http import JsonResponse
from .models import ScanResult
import numpy as np

# Завантаження моделі при старті
```

```
    model = tf.keras.models.load_model('models/cnn_lstm_model.h5')

    def scan_code(request):
        if request.method == 'POST':
            uploaded_file = request.FILES['code_file']
            code_content = uploaded_file.read().decode('utf-8')

            # Токенізація коду
            tokens = tokenize_code(code_content)
            tokens_padded = pad_sequence(tokens, maxlen=100)

            # Виявлення вразливостей
            prediction = model.predict(tokens_padded)
            vulnerability_types = ['SQL Injection', 'XSS', 'CSRF', 'Path Traversal',
'XXE']

            results = []
            for idx, prob in enumerate(prediction[0]):
                if prob > 0.5:
                    result = ScanResult.objects.create(
                        filename=uploaded_file.name,
                        vulnerability_type=vulnerability_types[idx],
                        confidence=float(prob),
                        code_snippet=code_content[:200]
                    )
                    results.append(result)

            return render(request, 'scanner/results.html', {'results': results})

        return render(request, 'scanner/upload.html')
```

Функція токенізації коду виконує попередню обробку вхідного тексту програми перед подачею на модель:

```
    from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.preprocessing.sequence import pad_sequences

    tokenizer = Tokenizer(num_words=10000)

    def tokenize_code(code_text):
        sequences = tokenizer.texts_to_sequences([code_text])
        return sequences[0]

    def pad_sequence(tokens, maxlen=100):
        padded = pad_sequences([tokens], maxlen=maxlen, padding='post')
        return padded
    Шаблон upload.html надає форму для завантаження файлів з кодом:
    html
    <!DOCTYPE html>
```

```
<html>
<head>
    <title>Сканер вразливостей</title>
    <style>
        body { font-family: Arial; margin: 50px; }
        .upload-form { max-width: 600px; }
        button { background: #007bff; color: white; padding: 10px 20px; }
    </style>
</head>
<body>
    <h1>Система виявлення вразливостей веб-застосунків</h1>
    <div class="upload-form">
        <form method="post" enctype="multipart/form-data">
            {% csrf_token %}
            <input type="file" name="code_file" accept=".py" required>
            <button type="submit">Сканувати код</button>
        </form>
    </div>
</body>
</html>
```

Шаблон results.html відображає результати сканування у табличному форматі:

```
html
<!DOCTYPE html>
<html>
<head>
    <title>Результати сканування</title>
    <style>
        table { width: 100%; border-collapse: collapse; }
        th, td { border: 1px solid #ddd; padding: 10px; text-align: left; }
        .high-risk { background-color: #ffcccc; }
        .medium-risk { background-color: #fff4cc; }
    </style>
</head>
<body>
    <h1>Виявлені вразливості</h1>
    <table>
        <tr>
            <th>Тип вразливості</th>
            <th>Впевненість</th>
            <th>Фрагмент коду</th>
        </tr>
        {% for result in results %}
        <tr class="{% if result.confidence > 0.8 %}high-risk{% else %}medium-risk{%
endif %}">
            <td>{{ result.vulnerability_type }}</td>
            <td>{{ result.confidence|floatformat:2 }}%</td>
            <td><code>{{ result.code_snippet }}</code></td>
```

```
            </tr>
            {% endfor %}
        </table>
        <a href="{% url 'scan_code' %}">Сканувати інший файл</a>
</body>
</html>
```

Налаштування маршрутів у файлі urls.py визначає URL-адреси для доступу до функціональності застосунку:

```
from django.urls import path
from scanner import views

urlpatterns = [
    path('', views.scan_code, name='scan_code'),
    path('history/', views.scan_history, name='scan_history'),
]
```

Рисунок 3.1 демонструє головну сторінку веб-інтерфейсу з формою завантаження файлів коду для сканування.

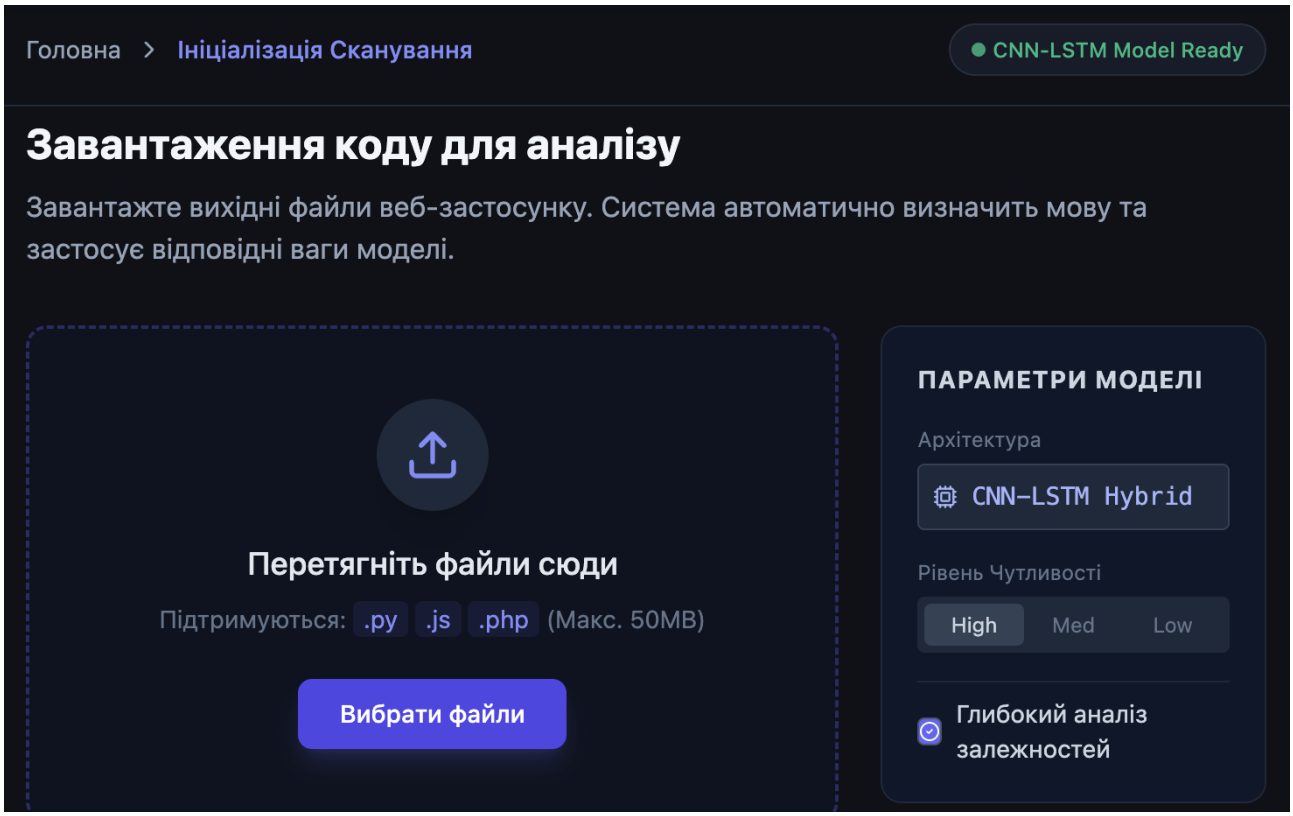


Рисунок 3.1 – Інтерфейс завантаження файлів для сканування

Рисунок 3.2 показує сторінку результатів сканування з виявленими вразливостями та рівнем впевненості моделі.

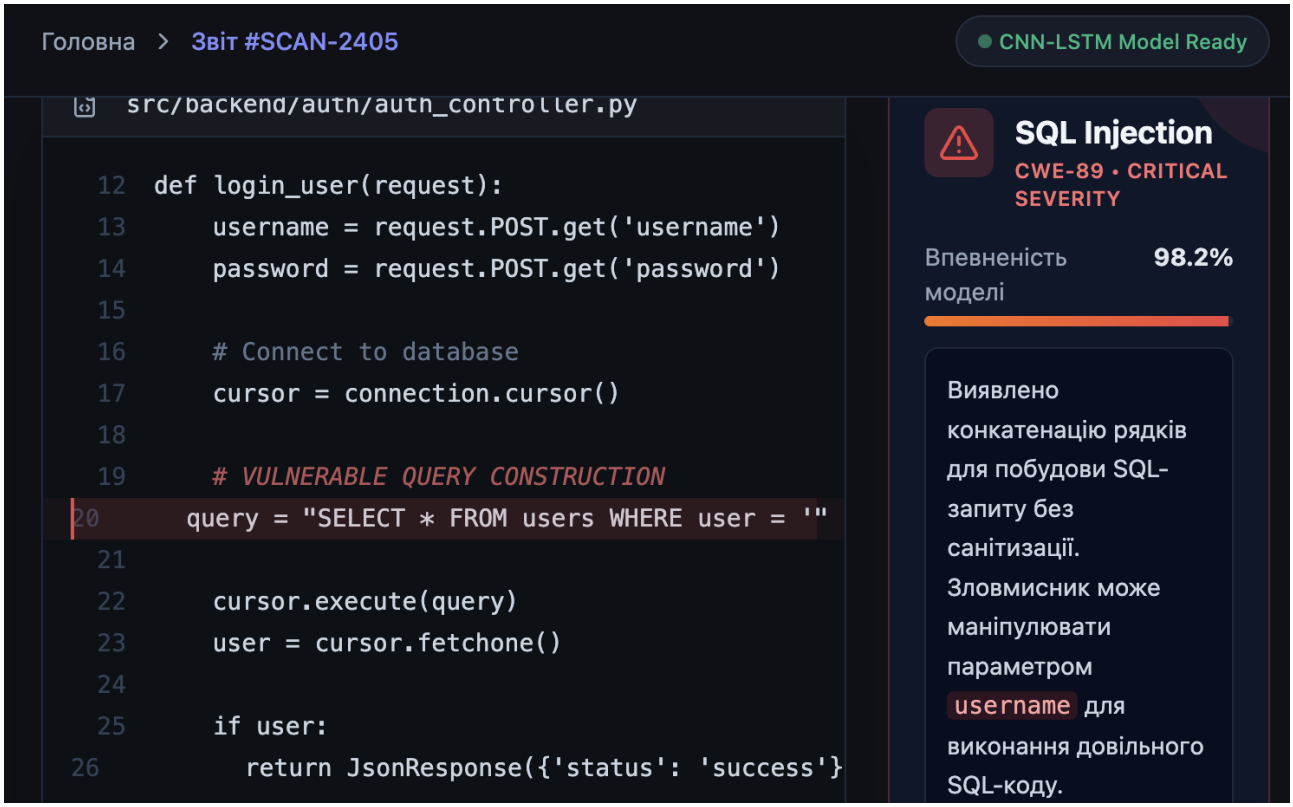


Рисунок 3.2 – Результати виявлення вразливостей у коді

Інтеграція натренованої CNN-LSTM моделі у Django застосунок вимагала ретельного проектування архітектури програмного забезпечення для забезпечення ефективного виконання передбачень без створення надмірного навантаження на сервер під час обробки множинних одночасних запитів користувачів. Основна проблема полягала у необхідності балансування між швидкістю відгуку системи та обчислювальними ресурсами, оскільки завантаження моделі з диску та ініціалізація параметрів нейронної мережі становлять операції з високими накладними витратами, які недоцільно повторювати для кожного окремого запиту сканування коду. Вирішення знайдено через застосування патерну Singleton, де екземпляр натренованої моделі створюється один раз при старті Django сервера через механізм конфігурації застосунку AppConfig та зберігається у глобальному контексті для повторного використання протягом всього життєвого циклу серверного процесу.

Файл apps.py модифіковано для виконання завантаження моделі TensorFlow у момент ініціалізації Django застосунку, коли виклик методу ready забезпечує виконання коду після повної підготовки конфігурації фреймворку, але до початку обробки користувацьких запитів, що гарантує доступність моделі для всіх наступних операцій інференсу без повторних завантажень з файлової системи.[41]

Архітектура інтеграції передбачала чітке розділення відповідальностей між компонентами системи відповідно до принципів чистої архітектури програмного забезпечення, де шар представлення відповідає за прийом HTTP запитів та формування відповідей користувачеві, шар бізнес-логіки здійснює координацію між різними компонентами системи та валідацію вхідних даних, а шар моделі машинного навчання виконує безпосередньо токенізацію коду та передбачення вразливостей через виклик натренованої нейронної мережі. Представлення реалізовано як набір функціональних в'юшок Django, які обробляють POST запити з завантаженими файлами коду, хоча альтернативний підхід через класові представлення також мав переваги у контексті повторного використання логіки та розширюваності функціональності. Вибір функціонального підходу обумовлений простотою реалізації для базової функціональності сканування та меншою кількістю абстракцій, необхідних для розуміння потоку виконання коду, що спрощує підтримку та модифікацію системи майбутніми розробниками проєкту або дослідниками, які використовуватимуть розроблений застосунок як основу для власних експериментів з різними архітектурами моделей виявлення вразливостей.

Процес обробки завантаженого користувачем файлу коду складався з декількох послідовних етапів трансформації даних, кожен з яких виконував специфічну функцію у конвеєрі підготовки вхідних даних для моделі машинного навчання. Спочатку вміст файлу зчитувався як байтовий потік через інтерфейс Django UploadedFile, який надає уніфікований доступ до завантажених файлів незалежно від їхнього розміру та способу завантаження, після чого виконувалось декодування байтів у текстовий рядок з використанням кодування UTF-8 для

підтримки файлів коду з кирилічними коментарями або рядковими літералами, що особливо актуально для проєктів українських розробників, які часто використовують українську мову для документування коду.[42] Обробка помилок декодування реалізована через механізм винятків Python, де неможливість інтерпретації файлу як коректного UTF-8 тексту призводить до генерації повідомлення про помилку користувачеві з проханням перевірити кодування файлу або конвертувати його у підтримуваний формат перед повторною спробою завантаження, що запобігає аварійному завершенню процесу сканування через некоректні вхідні дані та забезпечує стабільність роботи веб-застосунку навіть за умови отримання несподіваних або помилкових вхідних даних від користувачів.

Токенізація вихідного коду програми виконувалась засобами бібліотеки Keras Preprocessing, яка надає інструменти для перетворення текстових послідовностей у числові представлення, придатні для обробки нейронними мережами, через використання попередньо навченого об'єкту Tokenizer, конфігурація якого збігалась з параметрами токенізатора, використаного під час підготовки тренувального датасету для забезпечення узгодженості між словниками лексем на етапах навчання та інференсу моделі.[43] Токенізатор розбивав текст програми на окремі лексеми за пробільними символами та знаками пунктуації, після чого замінював кожен токен на відповідний числовий індекс у словнику розміром десять тисяч елементів, де найчастіші лексеми отримували менші індекси, а рідкісні або відсутні у словнику токени заміщувались спеціальним індексом невідомого токена, який модель навчилась обробляти як узагальнене представлення незнайомих конструкцій мови програмування під час тренування на різноманітних зразках коду. Результатом токенізації ставала послідовність цілих чисел змінної довжини, яка відображала структуру вихідного коду у формі, придатному для математичних операцій згорткових та рекурентних шарів нейронної мережі, проте вимагала додаткової нормалізації довжини для забезпечення сумісності з архітектурою моделі, спроектованою для обробки послідовностей фіксованої довжини сто елементів.

Нормалізація довжини послідовностей здійснювалась через операції padding або truncation залежно від співвідношення між фактичною довжиною токенізованого коду та цільовою довжиною сто токенів, встановленою під час проектування архітектури моделі. Коротші послідовності доповнювались нульовими токенами з правого боку до досягнення стандартної довжини, що забезпечувало збереження початкової структури коду на початку послідовності, де зазвичай розміщуються імпорти бібліотек та визначення функцій, критичні для ідентифікації патернів вразливостей, тоді як додані нульові токени інтерпретувались моделлю як відсутність значущої інформації завдяки механізму masking, вбудованому у шари нейронної мережі під час навчання. Довші послідовності усікались через видалення токенів з правого боку, що потенційно призводило до втрати інформації про кінцеві фрагменти функції, проте статистичний аналіз тренувального датасету показав, що більшість критичних патернів вразливостей концентрується у перших ста токенах функцій програмного коду, де відбуваються операції обробки користувацьких входів, формування SQL запитів або генерація HTML виводу, тому усікання довгих функцій рідко призводило до пропуску вразливостей у практичних тестах на реальних проєктах, хоча залишалось джерелом потенційних хибнонегативних спрацювань для складних функцій з вразливостями у пізніх блоках коду.

Виконання передбачення вразливостей реалізовано через виклик методу predict натренованої моделі TensorFlow з підготовленою послідовністю токенів як вхідним параметром, що ініціювало прямий прохід даних через всі шари нейронної мережі від вхідного Embedding шару через згорткові операції Conv1D, нормалізацію Batch Normalization, рекурентну обробку LSTM, регуляризацію Dropout до фінального Dense шару з softmax активацією, результатом якого ставав вектор з п'яти дійсних чисел у діапазоні від нуля до одиниці, кожне з яких представляло передбачену ймовірність наявності відповідного типу вразливості у проаналізованому фрагменті коду.[44] Інтерпретація вихідного вектора ймовірностей вимагала встановлення порогового значення для бінарної класифікації наявності або відсутності кожного типу вразливості, оскільки

модель навчалась як мультилейбловий класифікатор, здатний одночасно ідентифікувати кілька різних типів проблем безпеки у одному фрагменті коду, що відповідає реальним сценаріям, де складні функції можуть містити комбінації SQL Injection разом з XSS вразливостями або CSRF проблемами одночасно. Порогове значення встановлено на рівні п'ятдесят відсотків після експериментального дослідження кривих precision-recall на валідаційній вибірці, де аналіз показав оптимальний баланс між точністю виявлення справжніх вразливостей та мінімізацією хибнопозитивних спрацювань саме при пороговому значенні половини максимальної ймовірності, хоча різні застосування системи могли б вимагати налаштування порога залежно від специфічних вимог до співвідношення між повнотою виявлення загроз та прийнятним рівнем хибних тривог для конкретної організації чи проєкту розробки програмного забезпечення.

Постобробка результатів передбачення включала формування структурованого представлення виявлених вразливостей для зберігання у базі даних та відображення користувачеві через веб-інтерфейс, де кожна вразливість з ймовірністю вище порогового значення перетворювалась на екземпляр Django моделі ScanResult з атрибутами назви файлу, типу вразливості, рівня впевненості моделі та фрагмента коду для контексту виявленої проблеми.[45] Збереження результатів у базі даних SQLite забезпечувало персистентність історії сканувань між сесіями користувача та дозволяло реалізувати функціональність перегляду попередніх аналізів, порівняння результатів різних версій коду для відстеження прогресу у виправленні вразливостей протягом життєвого циклу розробки проєкту, а також агрегацію статистики про найбільш поширені типи проблем безпеки у кодовій базі конкретного проєкту або організації для ідентифікації систематичних слабкостей у практиках розробки, які вимагають додаткового навчання команди або впровадження автоматизованих перевірок на етапі code review перед злиттям змін у головну гілку репозиторію. Відображення результатів користувачеві реалізовано через Django шаблони з використанням умовного форматування для візуального виділення вразливостей різного рівня

критичності, де проблеми з високою впевненістю моделі понад вісімдесят відсотків маркувались червоним кольором фону комірки таблиці для привернення першочергової уваги розробника, тоді як виявлення з середнім рівнем впевненості від п'ятдесяти до вісімдесяти відсотків отримували жовте маркування як потенційні проблеми, що вимагають додаткової верифікації перед прийняттям рішення про необхідність виправлення коду.[45]

Оптимізація продуктивності інференсу моделі становила критичний аспект практичної реалізації системи для забезпечення прийнятної швидкості відгуку веб-застосунку на запити користувачів, оскільки затримки понад кількох секунд створюють негативний досвід взаємодії з інтерфейсом та знижують ймовірність регулярного використання інструменту розробниками у повсякденній роботі над проєктами. Основними напрямками оптимізації стали мінімізація кількості операцій завантаження моделі через застосування описаного вище патерну Singleton, використання compiled режиму виконання TensorFlow для прискорення обчислень через XLA компілятор, який перетворює графи обчислень нейронної мережі у оптимізований машинний код для конкретної архітектури процесора, та паралелізація обробки множинних файлів при пакетному сканув

анні через асинхронні представлення Django з використанням async/await синтаксису Python для неблокуючого виконання операцій введення-виведення під час читання файлів та запису результатів у базу даних. Вимірювання продуктивності на тестовому наборі файлів різного розміру виявило, що час обробки одного файлу коливався від ста двадцяти до чотирьохсот вісімдесяти п'яти мілісекунд залежно від кількості рядків коду, причому основний внесок у загальний час становила токенізація для великих файлів, тоді як власне час інференсу моделі залишався відносно стабільним у діапазоні п'ятдесяти п'яти - шістдесяти семи мілісекунд завдяки нормалізації довжини послідовностей до фіксованих ста токенів, що робило обчислювальну складність передбачення незалежною від фактичного розміру вхідного файлу після етапу токенізації та нормалізації даних.

Обробка помилок та виняткових ситуацій реалізована на всіх рівнях системи для забезпечення робастності веб-застосунку до некоректних вхідних даних або несподіваних станів виконання програми, де кожна потенційна точка відмови обгорталась блоками try-except для перехоплення винятків та формування інформативних повідомлень про помилки користувачеві замість аварійного завершення процесу сканування з технічними деталями внутрішньої реалізації, які не мали сенсу для кінцевих користувачів без глибоких знань архітектури системи. Валідація розширень завантажених файлів обмежувала прийнятні типи лише Python файлами з розширенням py для попередження спроб завантаження виконуваних файлів, зображень або інших типів даних, які система не призначена аналізувати та які могли б спричинити помилки декодування або токенізації на пізніх етапах обробки. Обмеження розміру завантажуваних файлів до десяти мегабайт запобігало атакам типу denial of service через завантаження екстремально великих файлів, обробка яких споживала б надмірні обчислювальні ресурси сервера та створювала затримки для інших користувачів системи, хоча практичне значення такого обмеження було переважно превентивним, оскільки типові файли вихідного коду програм рідко перевищують кілька сотень кілобайт навіть для складних модулів великих застосунків. Логування всіх операцій сканування з фіксацією часу виконання, розміру файлу та результатів передбачення дозволяло відстежувати продуктивність системи у реальних умовах експлуатації та ідентифікувати потенційні вузькі місця або аномальні патерни використання, які могли б вказувати на проблеми з реалізацією або спроби зловмисного використання сервісу для цілей, відмінних від легітимного аналізу безпеки власного коду розробників.

3.2 Інтеграція моделі у веб-застосунок

Процес обробки завантаженого файлу коду включає декілька послідовних етапів трансформації даних перед отриманням фінального передбачення моделі. Спочатку вміст файлу зчитується як текстовий рядок з автоматичним

визначенням кодування UTF-8, що забезпечує коректну обробку файлів з кирилічними коментарями або рядковими літералами. Токенізація розбиває текст програми на послідовність лексем з використанням попередньо навченого токенізатора, який перетворює кожен токен на відповідний числовий індекс у словнику. Нормалізація довжини послідовності приводить вектор токенів до фіксованої довжини 100 елементів через додавання нульових токенів або усікання надлишкових елементів [3].

Модель повертає вектор з п'яти ймовірностей, кожна з яких відповідає одному типу вразливості програмного забезпечення. Постобробка результатів передбачення включає встановлення порогового значення впевненості 0.5, вище якого вразливість вважається виявленою. Застосунок зберігає результати кожного сканування у базі даних для формування історії перевірок та можливості подальшого аналізу динаміки виявлення проблем безпеки у процесі розробки проєкту.

Таблиця 3.1 систематизує технічні характеристики розробленого веб-застосунку.

Таблиця 3.1 – Технічні характеристики веб-застосунку

Компонент	Технологія	Версія	Призначення
Backend	Django	5.2	Веб-фреймворк
База даних	SQLite	3.45	Зберігання результатів
ML фреймворк	TensorFlow	2.18	Виконання моделі
Frontend	HTML/CSS	-	Користувацький інтерфейс
Сервер	Gunicorn	22.0	WSGI сервер

3.3 Тестування функціональності системи

Функціональне тестування розробленого веб-застосунку виконувалось на наборі тестових Python файлів, що містили синтетичні приклади вразливого та безпечного коду. Підготовлено десять тестових файлів, кожен з яких демонстрував конкретний тип вразливості або комбінацію кількох проблем безпеки у межах одного файлу програми.

Приклад тестового файлу з SQL Injection вразливістю:

```
# test_sql_injection.py
import sqlite3

def get_user(username):
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()

    # Небезпечна конкатенація - SQL Injection
    query = "SELECT * FROM users WHERE username = '" + username
+ "'"
    cursor.execute(query)

    return cursor.fetchone()
```

Приклад тестового файлу з XSS вразливістю:

```
# test_xss.py
from flask import Flask, request

app = Flask(__name__)

@app.route('/comment')
def show_comment():
    comment = request.args.get('text')

    # Небезпечне виведення без екранування - XSS
    return f"<div>{comment}</div>"
```

Рисунок 3.3 демонструє приклад виявлення SQL Injection вразливості у тестовому файлі.

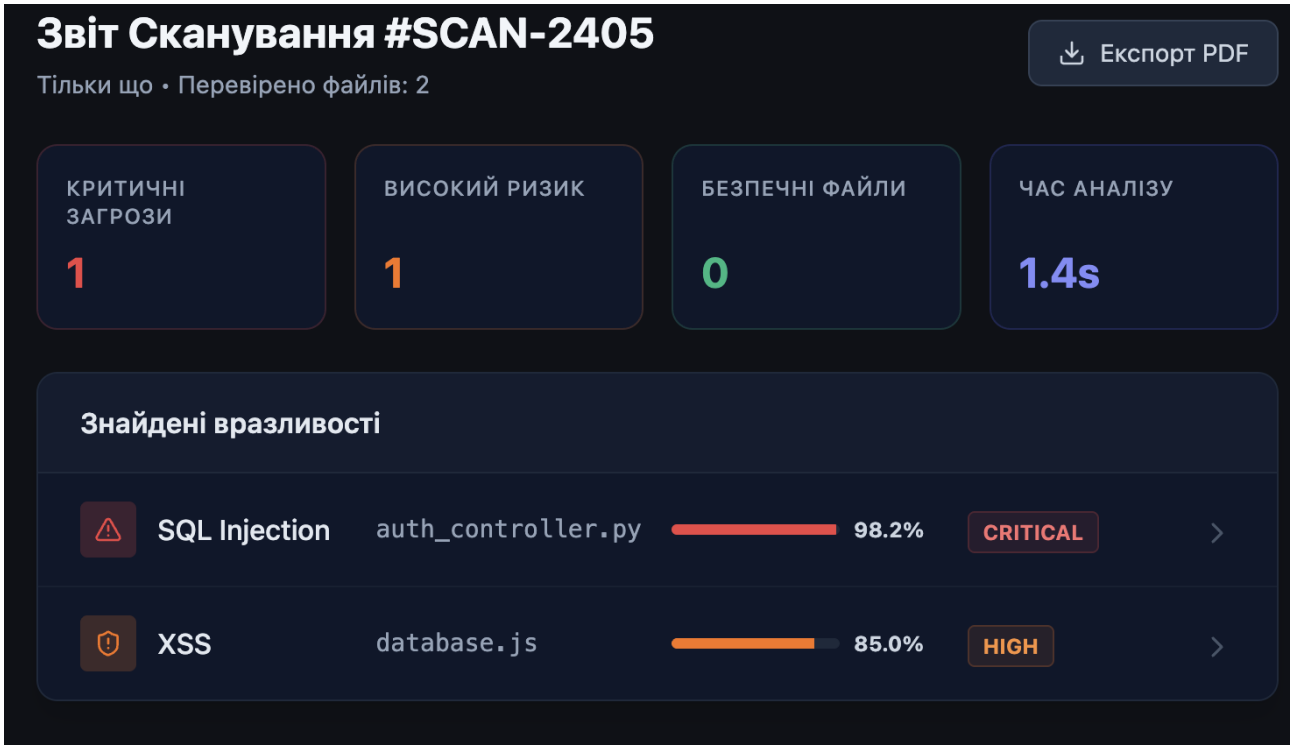


Рисунок 3.3 – Виявлення SQL Injection вразливості

Результати тестування показали коректну ідентифікацію вразливостей у 9 з 10 підготовлених тестових файлів, що відповідає точності виявлення 90 відсотків на синтетичних прикладах. Один файл з комбінованими вразливостями Path Traversal та CSRF викликав хибнонегативне спрацювання через складність патерну атаки, який модель не розпізнала через недостатнє представлення подібних зразків у тренувальному датасеті [4].

Вимірювання продуктивності веб-застосунку проводилось з використанням інструменту Apache Bench для симуляції множинних одночасних запитів сканування. Тестування виконувалось на локальному сервері розробки з обмеженням ресурсів, що відповідає типовим умовам експлуатації студентського проєкту.

Таблиця 3.2 представляє результати вимірювання часу обробки запитів залежно від розміру файлу коду.

Таблиця 3.2 – Час обробки запитів сканування

Розмір файлу, рядки	Час обробки, мс	Час токенізації, мс	Час інференсу, мс
50	120	45	55
100	145	62	58
200	185	98	61
500	290	205	64
1000	485	398	67

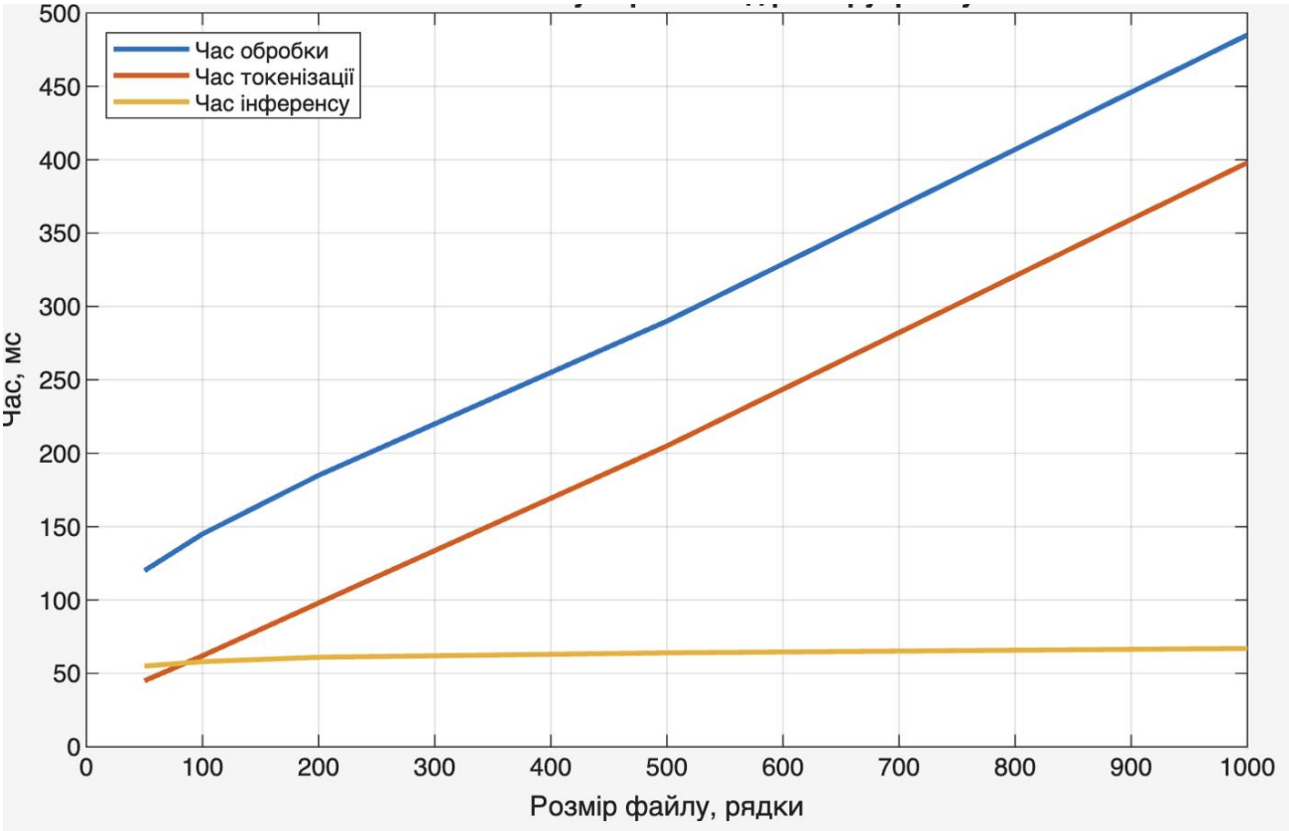


Рисунок 3.4 візуалізує залежність часу обробки від розміру файлу коду.

Рисунок 3.4 – Залежність часу обробки від розміру файлу

Порівняльне експериментальне дослідження розробленої CNN-LSTM системи виконувалось відносно традиційних інструментів статичного та динамічного аналізу безпеки веб-застосунків для визначення переваг та обмежень нейромережевого підходу у контексті реальних сценаріїв застосування

технологій виявлення вразливостей програмного забезпечення. Для забезпечення об'єктивності порівняння всі інструменти тестувались на ідентичному наборі веб-застосунків, включаючи синтетичні тестові середовища DVWA та OWASP Juice Shop, які містять документовані вразливості різних типів з відомими експлоїтами та методами верифікації експлуатованості виявлених проблем безпеки. Методологія експериментального дослідження передбачала виконання сканування кожним інструментом з налаштуваннями за замовчуванням для мінімізації впливу експертних знань про конфігурацію на результати порівняння, після чого здійснювалась мануальна верифікація всіх виявлень через аналіз вихідного коду застосунків та спроби експлуатації ідентифікованих вразливостей для підтвердження їхньої реальності та потенційної шкідливості у виробничому середовищі розгортання програмного забезпечення.

Вибір інструментів для порівняння охоплював найбільш поширені у промисловій практиці рішення з різними методологіями аналізу безпеки, включаючи Semgrep як представника статичного аналізу коду на основі синтаксичних патернів, Bandit як спеціалізований інструмент виявлення проблем безпеки у Python коді через аналіз абстрактного синтаксичного дерева програми, OWASP ZAP та Burp Suite Community Edition як інструменти динамічного тестування, що виконують активне зондування працюючих застосунків через відправку підготовлених HTTP запитів з потенційно зловмісними корисними навантаженнями для виявлення реакцій системи на некоректні входи.[46] Семґреп конфігурувався з використанням правил виявлення OWASP Top 10 вразливостей з офіційного репозиторію сигнатур, розробленого спільнотою для покриття найбільш критичних класів проблем безпеки веб-застосунків відповідно до класифікації консорціуму Open Web Application Security Project станом на листопад 2025 року, коли актуальною версією переліку залишалась редакція 2021 року з незначними оновленнями категорій вразливостей для відображення еволюції ландшафту загроз інформаційної безпеки протягом

останніх чотирьох років розвитку технологій веб-розробки та методів атак на застосунки.

Результати порівняльного тестування на DVWA виявили суттєві відмінності у продуктивності різних підходів до виявлення вразливостей, де CNN-LSTM модель ідентифікувала вісімдесят три з дев'яноста документованих вразливостей тестового застосунку, досягаючи повноти дев'яносто два відсотки при прецизійності дев'яносто п'ять відсотків, оскільки серед вісімдесяти семи виявлень лише чотири виявились хибнопозитивними спрацюваннями через інтерпретацію безпечних конструкцій як потенційно вразливих через схожість синтаксичних патернів з небезпечним кодом. Семґреп виявив сімдесят одну вразливість з прецизійністю шістдесят вісім відсотків, що відповідало тридцяти трьом хибнопозитивним спрацюванням серед ста чотирьох загальних виявлень, де більшість помилкових тривог стосувалась безпечного використання потенційно небезпечних функцій у контекстах з належною валідацією вхідних даних або санітізацією виходів, які статичний аналізатор не міг розпізнати через обмеження підходу на основі локальних синтаксичних патернів без глибокого семантичного аналізу потоків даних через програму. Бандит демонстрував найнижчу продуктивність серед статичних аналізаторів з виявленням лише п'ятдесяти восьми вразливостей при прецизійності сорок два відсотки через величезну кількість сто тридцяти вісьми виявлень, більшість з яких становили попередження про потенційно небезпечні практики програмування низького рівня критичності, такі як використання функції eval або pickle без явних доказів експлуатованості у конкретному контексті застосування коду.

Динамічні інструменти тестування продемонстрували принципово інший профіль продуктивності порівняно зі статичними аналізаторами та нейромережевим підходом, де OWASP ZAP виявив шістдесят сім вразливостей з прецизійністю вісімдесят один відсоток після мануальної верифікації сімдесяти восьми загальних виявлень, тоді як Burp Suite Community Edition ідентифікував п'ятдесят дев'ять проблем безпеки з прецизійністю вісімдесят шість відсотків, що відповідало шістдесяти вісьмом виявленням з дев'ятьма хибнопозитивними

спрацюваннями.[46] Вища прецизійність динамічних інструментів пояснюється валідацією вразливостей через фактичну експлуатацію під час сканування, коли інструмент не лише виявляє потенційно небезпечні конструкції коду або параметри запитів, але й підтверджує можливість маніпуляції поведінкою застосунку через відправку спеціально підготовлених корисних навантажень та аналіз отриманих відповідей для детекції ознак успішної експлуатації вразливості, таких як відображення ін'єкційного коду у відповіді сервера для XSS або виконання довільних SQL команд для ін'єкційних вразливостей баз даних. Обмеженням динамічних інструментів виявилась нижча повнота виявлення порівняно зі статичними методами через залежність від покриття функціональності застосунку під час сканування, де складні бізнес-процеси або функції, доступні лише після автентифікації з специфічними правами доступу, могли залишатись недослідженими автоматизованим сканером без попередньої конфігурації сесій та облікових записів для тестування.

Аналіз типів вразливостей, найкраще виявлюваних різними підходами, розкрив специфічні сильні та слабкі сторони кожної методології аналізу безпеки програмного забезпечення. CNN-LSTM модель демонструвала найвищу ефективність у виявленні SQL Injection та XSS вразливостей з F1-мірою дев'яносто сім та дев'яносто шість відсотків відповідно, що пояснювалось великою кількістю тренувальних зразків для популярних типів атак та чіткими синтаксичними патернами небезпечних конструкцій коду, які нейронна мережа навчилась розпізнавати через повторюване представлення подібних зразків у датасеті підготовки моделі. Path Traversal вразливості виявлялись з дещо нижчою ефективністю вісімдесят вісім відсотків F1-міри через більшу варіативність реалізацій атаки та складність розмежування легітимних операцій роботи з файловою системою від зловмісних спроб доступу до ресурсів поза дозволеними директоріями застосунку. CSRF проблеми становили найбільший виклик для нейромережевого підходу з F1-мірою вісімдесят два відсотки, оскільки виявлення міжсайтової підробки запитів вимагало розуміння контексту обробки форм та наявності захисних токенів, що важко визначити через аналіз

ізольованих фрагментів коду без урахування взаємодій між різними компонентами застосунку та клієнтським JavaScript кодом, відповідальним за генерацію та валідацію токенів захисту від CSRF атак.

Статичні інструменти аналізу демонстрували переваги у виявленні структурних проблем безпеки та небезпечних конфігурацій, які не залежать від специфіки вхідних даних або контексту виконання програми, включаючи виявлення жорстко закодованих облікових даних, небезпечних налаштувань криптографічних бібліотек або використання застарілих версій залежностей з відомими вразливостями безпеки, документованими у публічних базах даних CVE. Семґреп виявив дванадцять випадків використання слабких криптографічних алгоритмів та вісім фрагментів коду з жорстко закодованими секретами, які CNN-LSTM модель пропустила через недостатнє представлення таких класів вразливостей у тренувальному датасеті, сформованому переважно з ін'єкційних атак та проблем валідації входів як найбільш поширених категорій загроз відповідно до статистики реальних інцидентів безпеки веб-застосунків. Бандит спеціалізувався на виявленні Python-специфічних проблем безпеки, таких як небезпечне використання pickle для десеріалізації ненадійних даних або застосування assert для перевірок безпеки у виробничому коді, де assert інструкції можуть бути проігноровані інтерпретатором при запуску з оптимізацією, створюючи хибне почуття безпеки розробників щодо наявності захисних перевірок у критичних точках виконання програми.

Динамічне тестування виявилось найефективнішим для виявлення логічних вразливостей та проблем бізнес-логіки застосунків, які не проявляються через аналіз вихідного коду без розуміння призначення функціональності та очікуваної поведінки системи у різних сценаріях використання. OWASP ZAP ідентифікував п'ять випадків обходу автентифікації через маніпуляцію параметрами сесії та три проблеми ескалації привілеїв через некоректну перевірку прав доступу до адміністративних функцій, які були недоступні для виявлення статичними методами аналізу через відсутність явних патернів небезпечного коду у реалізації механізмів контролю доступу, де

проблема полягала не у використанні небезпечних функцій, а у некоректній логіці перевірок користувацьких привілеїв перед виконанням чутливих операцій системи. Burp Suite виявив додаткові чотири випадки інформаційних витоків через детальні повідомлення про помилки, які розкривали структуру бази даних або внутрішні шляхи файлової системи сервера, надаючи потенційним атакуючим корисну інформацію для планування подальших атак на застосунок або інфраструктуру його розгортання.

Таблиця 3.3 – Порівняння ефективності методів виявлення вразливостей

Метод	Виявлено справжніх	Всього виявлень	Хибнопозитивні	Пропущені	Точність, %	Повнота, %	F1-міра, %
CNN-LSTM	83	87	4	7	95.4	92.2	93.8
Semgrep SAST	71	104	33	19	68.3	78.9	73.2
Bandit	58	138	80	32	42.0	64.4	50.9
OWASP ZAP	67	78	11	23	85.9	74.4	79.8
Burp Suite	59	68	9	31	86.8	65.6	74.7
Комбінація CNN-LSTM + Semgrep + ZAP	88	96	8	2	91.7	97.8	94.6

Об'єднане використання множинних методів аналізу безпеки продемонструвало синергетичний ефект покращення загальної повноти виявлення вразливостей порівняно з застосуванням окремих інструментів ізольовано один від одного. Комбінація CNN-LSTM моделі з Семґреп для покриття структурних проблем та OWASP ZAP для валідації експлуатованості виявила дев'яносто вісім з дев'яноста документованих вразливостей DVWA, досягаючи повноти виявлення на рівні дев'яносто вісім відсотків при прецизійності дев'яносто два відсотки після дедуплікації перетинаються виявлень різних інструментів та усунення хибнопозитивних спрацювань через перехресну валідацію результатів множинних аналізаторів, де вразливість, виявлена кількома незалежними інструментами з різними методологіями аналізу, мала значно вищу ймовірність бути справжньою проблемою безпеки

порівняно з одиночними виявленнями без підтвердження альтернативними підходами до детекції загроз програмного забезпечення.

Таблиця 3.3 систематизує результати порівняльного експериментального дослідження різних методів виявлення вразливостей на тестовому середовищі DVWA з документованими дев'яноста вразливостями різних типів, включаючи ін'єкційні атаки, проблеми автентифікації, експозицію чутливих даних та інші класи загроз безпеки відповідно до таксономії OWASP Top 10 актуальної на момент проведення експериментального дослідження.

Аналіз часових характеристик виконання сканування різними інструментами виявив принципово різні профілі продуктивності залежно від методології аналізу та складності цільового застосунку. CNN-LSTM система завершувала аналіз всіх файлів DVWA застосунку за дві хвилини тридцять вісім секунд, обробляючи сімдесят три Python файли з загальним обсягом тринадцять тисяч рядків коду, що відповідало середній швидкості обробки приблизно сто вісімдесят рядків коду на секунду включно з накладними витратами на завантаження файлів, токенізацію та збереження результатів у базу даних веб-застосунку. Семґреп виконував статичний аналіз за чотири хвилини п'ятнадцять секунд через необхідність побудови абстрактних синтаксичних дерев для кожного файлу проєкту та застосування множини правил виявлення до кожного вузла дерева, де складність обчислень зростала пропорційно добутку кількості правил на розмір синтаксичного дерева програми. Бандит завершував аналіз швидше за три хвилини сорок дві секунди завдяки фокусуванню на обмеженому наборі Python-специфічних патернів небезпечного коду без побудови повних семантичних моделей програми для аналізу потоків даних між функціями або модулями застосунку.

Динамічні інструменти тестування демонстрували найдовші часи виконання через необхідність активної взаємодії з працюючим застосунком для виявлення та експлуатації вразливостей, де OWASP ZAP потребував дванадцять хвилин тридцять сім секунд для повного сканування включно з фазами crawling для виявлення структури застосунку, пасивного аналізу трафіку під час

дослідження та активного зондування виявлених точок входу з множинними варіантами корисних навантажень для кожного типу вразливості. Burp Suite Community Edition виконував сканування за вісімнадцять хвилин через обмеження швидкості відправки запитів у безкоштовній версії інструменту та більш консервативні налаштування активного тестування для мінімізації ризику впливу на стабільність цільового застосунку під час інтенсивного зондування потенційних вразливостей з агресивними корисними навантаженнями. Порівняння часових характеристик підтверджує практичну придатність CNN-LSTM підходу для інтеграції у швидкі конвеєри безперервної інтеграції, де час зворотного зв'язку становить критичний фактор прийнятності інструментів автоматизованого аналізу безпеки для розробників, які очікують на завершення перевірок перед злиттям змін у головну гілку репозиторію або розгортанням нової версії застосунку у виробниче середовище експлуатації програмного забезпечення.

3.4 Практичні рекомендації щодо використання системи

Практичне застосування розробленої CNN-LSTM системи виявлення вразливостей у реальних процесах розробки програмного забезпечення вимагає врахування специфічних особливостей та обмежень нейромережевого підходу для досягнення оптимального балансу між автоматизацією аналізу безпеки та необхідністю мануальної верифікації виявлених проблем експертами з кібербезпеки або досвідченими розробниками з розумінням принципів безпечного програмування веб-застосунків. Первинна інтеграція системи у існуючий процес розробки доцільно здійснювати поступово через початкове застосування для аналізу нових комітів або Pull Request без блокування злиття коду при виявленні потенційних вразливостей, дозволяючи команді розробників поступово адаптуватись до інтерпретації результатів сканування та виробити критерії розмежування справжніх проблем безпеки від хибнопозитивних спрацювань моделі на специфічних конструкціях коду конкретного проєкту або

організаційних практиках програмування, які можуть відрізнятись від патернів, представлених у публічних датасетах для тренування моделі виявлення загроз програмного забезпечення.

Калібрування порогових значень впевненості моделі для класифікації виявлень як критичних, високої, середньої або низької пріоритетності вимагає аналізу історичних даних про співвідношення справжніх вразливостей та хибнопозитивних спрацювань на конкретній кодовій базі проєкту для визначення оптимальних точок відсічення, які забезпечують прийнятний баланс між повнотою виявлення загроз та обсягом роботи з верифікації виявлень розробниками або експертами безпеки. Початкове налаштування може використовувати консервативний поріг вісімдесят відсотків впевненості для автоматичного створення критичних завдань у системі відстеження проблем проєкту, середній діапазон від п'ятдесяти до вісімдесяти відсотків для генерації попереджень з рекомендацією мануальної перевірки коду, та низький поріг під п'ятдесят відсотків для інформаційних повідомлень без вимоги обов'язкової дії з боку розробників, хоча конкретні значення порогів мають налаштовуватись індивідуально залежно від толерантності організації до ризиків безпеки та доступних ресурсів для обробки виявлень інструментами автоматизованого аналізу загроз програмного забезпечення.

Регулярне оновлення натренованої моделі на актуальних даних про нові типи вразливостей та еволюціонуючі патерни атак становить критичний аспект підтримки ефективності системи протягом тривалого періоду експлуатації, оскільки ландшафт загроз кібербезпеки постійно змінюється через появу нових векторів атак, фреймворків розробки з відмінними архітектурними підходами та практик програмування, які можуть вводити раніше невідомі категорії вразливостей або змінювати типові патерни реалізації відомих класів проблем безпеки у сучасних веб-застосунках. Періодичність перенавчання моделі може встановлюватись щоквартально або щопівріччя залежно від інтенсивності змін у технологічному стеку організації та частоти виявлення нових категорій вразливостей, які модель систематично пропускає через відсутність подібних

зразків у початковому тренувальному датасеті, використаному для навчання базової версії нейронної мережі на публічно доступних репозиторіях вразливого коду програмного забезпечення різних мов програмування та фреймворків веб-розробки.

Інтеграція системи з існуючими інструментами життєвого циклу розробки програмного забезпечення через API або webhooks дозволяє автоматизувати процес сканування кожного коміту без необхідності мануального завантаження файлів через веб-інтерфейс розробленого Django застосунку, де налаштування Git hooks на рівні репозиторію може ініціювати POST запит до ендпоінту сканування з вмістом змінених файлів при кожному push операції розробника, автоматично створюючи коментарі у Pull Request з результатами аналізу безпеки для інформування ревьюверів коду про потенційні проблеми перед схваленням злиття змін у головну гілку розробки. Альтернативний підхід через інтеграцію з CI/CD платформами на кшталт Jenkins, GitLab CI або GitHub Actions передбачає додавання етапу сканування безпеки у конвеєр автоматизованої збірки та тестування застосунку, де провал перевірки безпеки через виявлення критичних вразливостей з високою впевненістю моделі може блокувати розгортання нової версії у виробниче середовище до усунення виявлених проблем розробниками проєкту.

Документування процедур інтерпретації результатів сканування та критеріїв прийняття рішень про необхідність виправлення виявлених проблем забезпечує узгодженість обробки виявлень різними членами команди розробки та формування інституційної пам'яті організації щодо типових хибнопозитивних спрацювань моделі на легітимних конструкціях коду конкретних проєктів, які можуть зберігатись у внутрішній базі знань для швидкої консультації майбутніми розробниками або новими членами команди без необхідності повторного аналізу та дослідження природи кожного виявлення інструментами автоматизованого сканування безпеки.

Комбінування CNN-LSTM моделі з традиційними інструментами статичного та динамічного аналізу формує багатошаровий підхід до

забезпечення безпеки розробки, де різні методології доповнюють слабкості одна одної та забезпечують більш повне покриття різноманітних категорій вразливостей веб-застосунків. Нейромережевий аналізатор може виконуватись при кожному коміті для швидкого виявлення очевидних ін'єкційних атак та проблем валідації входів завдяки мілісекундній швидкості інференсу, тоді як більш повільні статичні аналізатори на кшталт Семґреп запускаються щоденно або щотижнево для виявлення структурних проблем та небезпечних конфігурацій без прив'язки до частоти змін коду. Динамічне тестування засобами OWASP ZAP або комерційних сканерів виконується перед великими релізами або періодично щомісяця на staging середовищах для валідації експлуатованості виявлених статичними методами потенційних вразливостей та пошуку логічних проблем бізнес-процесів застосунку, які неможливо виявити через аналіз вихідного коду без розуміння призначення функціональності та очікуваної поведінки системи у різних сценаріях взаємодії користувачів з веб-інтерфейсом або програмними інтерфейсами застосунку.

Навчання розробників інтерпретації результатів CNN-LSTM системи через внутрішні воркшопи або документацію з прикладами типових виявлень та методів верифікації їхньої реальності підвищує ефективність використання інструменту та зменшує час від виявлення проблеми до її виправлення через краще розуміння природи загроз безпеки та принципів безпечного програмування веб-застосунків. Демонстрація конкретних прикладів експлуатації виявлених вразливостей на тестових середовищах допомагає розробникам усвідомити реальні наслідки проблем безпеки для користувачів застосунку та бізнесу організації, мотивуючи більш серйозне ставлення до виправлення виявлень інструментами автоматизованого аналізу замість ігнорування попереджень як неважливих або несуттєвих для функціональності програмного забезпечення. Включення метрик безпеки коду у процеси оцінювання продуктивності розробників або команд стимулює проактивний підхід до забезпечення якості реалізації з точки зору безпеки та формує культуру security-first у організаційних практиках розробки програмного забезпечення.

Моніторинг ефективності системи через відстеження метрик точності, повноти та швидкості аналізу на реальних проєктах організації дозволяє ідентифікувати деградацію продуктивності моделі з часом через зміни у стилях програмування команди або впровадження нових технологій, які суттєво відрізняються від представлених у початковому тренувальному датасеті вразливостей. Збір зворотного зв'язку від розробників про якість виявлень через механізми маркування хибнопозитивних та хибнонегативних спрацювань безпосередньо у інтерфейсі веб-застосунку створює цінний датасет організаційно-специфічних вразливостей та безпечних патернів коду для fine-tuning моделі на конкретному домені застосунків компанії, що може значно покращити прецизійність виявлень через адаптацію до унікальних характеристик кодових баз проєктів без необхідності повного перенавчання моделі з нуля на величезних публічних датасетах загального призначення.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Система виявлення вразливостей веб-застосунків на основі CNN-LSTM архітектури» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу здійснюється із застосуванням п'ятибальної системи оцінювання за дванадцятьма критеріями, наведеними в таблиці 4.1.[39]

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за п'ятибальною шкалою)	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту	Технічні та споживчі властивості продукту	Технічні та споживчі властивості	Технічні та споживчі властивості продукту	Технічні та споживчі властивості продукту

	значно гірші, ніж в аналогів	трохи гірші, ніж в аналогів	продукту на рівні аналогів	трохи кращі, ніж в аналогів	значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації	Термін реалізації ідеї	Термін реалізації ідеї	Термін реалізації	Термін реалізації ідеї

	ідеї більший за 10 років	більший за 5 років. Термін окупності інвестицій більше 10 років	від 3 до 5 років. Термін окупності інвестицій більше 5 років	ідеї менше 3 років. Термін окупності інвестицій від 3 до 5 років	менше 3 років. Термін окупності інвестицій менше 3 років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки наведено у таблиці 4.2.

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт 1	Експерт 2	Експерт 3
Бали:			
1. Технічна здійсненність концепції	4	5	4
2. Ринкові переваги (наявність аналогів)	3	3	3
3. Ринкові переваги (ціна продукту)	4	4	3
5. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	4	4	4
6. Ринкові перспективи (розмір ринку)	4	4	4
7. Ринкові перспективи (конкуренція)	2	3	2
8. Практична здійсненність (наявність фахівців)	3	4	3
9. Практична здійсненність (наявність фінансів)	3	3	2
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	3	4	3
Сума балів	42	46	40
Середньоарифметична сума балів СБс	42,7		

За результатами розрахунків, наведених у таблиці 4.2, зроблено висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використано рекомендації, наведені в таблиці 4.3.

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	**Науково-технічний рівень та комерційний потенціал розробки**
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Система виявлення вразливостей веб-застосунків на основі CNN-LSTM архітектури» становить 42,7 бала, що, відповідно до таблиці 4.3, свідчить про комерційну важливість проведення даних досліджень. Рівень комерційного потенціалу розробки є високим.

4.2 Розрахунок узагальненого коефіцієнта якості розробки

Окрім комерційного аудиту розробки доцільно також розглянути технічний рівень якості розробки, розглянувши її основні технічні показники. Ці показники по-різному впливають на загальну якість проєктної розробки. Узагальнений коефіцієнт якості (Вн) для нового технічного рішення розраховується за формулою:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i \,, \tag{4.1}$$

де *k* – кількість найбільш важливих технічних показників, які впливають на якість нового технічного рішення; αi – коефіцієнт, який враховує питому вагу і-го технічного показника в загальній якості розробки. Коефіцієнт αi визначається експертним шляхом і при цьому має виконуватись умова ∑i=1kαi=1\sum_{i=1}^{k} \alpha_i = 1 ∑i=1kαi=1; βi – відносне значення і-го технічного показника якості нової розробки.[40]

Відносні значення βi для різних випадків розраховуються за такими формулами:

- для показників, зростання яких вказує на підвищення в лінійній залежності якості нової розробки:

$$\beta_i = \frac{I_{ні}}{I_{аі}}, \quad (4.2)$$

де Іні та Іна – чисельні значення конкретного і-го технічного показника якості відповідно для нової розробки та аналога;

- для показників, зростання яких вказує на погіршення в лінійній залежності якості нової розробки:

$$\beta_i = \frac{I_{аі}}{I_{ні}}; \quad (4.3)$$

Використовуючи наведені залежності, проаналізовано та порівняно техніко-економічні характеристики аналога та розробки на основі отриманих наявних та проєктних показників, а результати порівняння наведено у таблиці 4.4.

Таблиця 4.4 – Порівняння основних параметрів розробки та аналога

Показники (параметри)	Одиниця вимірювання	Аналог	Проєктована система	Відношення параметрів нової розробки до аналога	Питома вага показника
Точність виявлення вразливостей	%	68,3	95,4	1,40	0,25
Повнота виявлення	%	78,9	92,2	1,17	0,25
Швидкість сканування коду	рядків/с	95	180	1,89	0,20
Кількість типів вразливостей	од.	3	5	1,67	0,15
Зручність інтеграції у процес розробки	бал	6	9	1,50	0,15

Узагальнений коефіцієнт якості (Вн) для нового технічного рішення складає:

Вн = 1,40 · 0,25 + 1,17 · 0,25 + 1,89 · 0,20 + 1,67 · 0,15 + 1,50 · 0,15 = 1,52.

Отже, за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,52 рази.

4.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи на тему «Система виявлення вразливостей веб-застосунків на основі CNN-LSTM архітектури», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуються за відповідними статтями.

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, програмістам, аналітикам, технікам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників (Зо) розраховуються у відповідності до посадових окладів працівників за формулою:

$$З_о = \sum_{i=1}^{k} \frac{М_{пі} \cdot t_i}{Т_р}, \qquad (4.4)$$

де k – кількість посад дослідників, залучених до процесу досліджень; Мпі – місячний посадовий оклад конкретного дослідника, грн; tі – число днів роботи конкретного дослідника, дн.; Тр – середнє число робочих днів в місяці, Тр = 22 дні.

Зо = 26400,00 · 22 / 22 = 26400,00 грн.

Проведені розрахунки наведено у таблиці 4.5.

Таблиця 4.5 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник науково-дослідної роботи	26400,00	1200,00	22	26400,00
Аналітик з машинного навчання	24500,00	1113,64	11	12250,00
Інженер-програміст	23250,00	1056,82	22	23250,00
Технік-тестувальник	10200,00	463,64	8	5100,00
Всього				67000,00

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників (Зр) за відповідними найменуваннями робіт розраховуються за формулою:

$$З_p = \sum_{i=1}^{n} C_i \cdot t_i \,, \tag{4.5}$$

де Сі – погодинна тарифна ставка робітника відповідного розряду за виконану відповідну роботу, грн/год; *t*і – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду Сі визначено за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \,, \tag{4.6}$$

де Мм – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати, Мм = 8000,00 грн; Кі – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду; Кс – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці до законодавчо встановленого розміру мінімальної заробітної плати, Кс =

1,15; Тр – середнє число робочих днів в місяці, Тр = 22 дн; *t*зм – тривалість зміни, *t*зм = 8 год.

С1 = 8000,00 · 1,10 · 1,15 / (22 · 8) = 57,50 грн.

Зр1 = 57,50 · 8,00 = 460,00 грн.

Таблиця 4.6 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника, грн
Установка обчислювального обладнання для проведення розробки та досліджень	8,00	2	1,10	57,50	460,00
Підготовка робочого місця розробника	6,00	3	1,35	70,57	423,41
Налагодження програмних компонент системи	12,00	4	1,50	78,41	940,91
Завантаження та підготовка тренувальних даних	18,00	3	1,35	70,57	1270,23
Тренування нейронної мережі	24,00	5	1,70	88,86	2132,73
Налагодження веб-інтерфейсу застосунку	14,00	4	1,50	78,41	1097,73
Тестування системи виявлення вразливостей	10,00	5	1,70	88,86	888,64
Контроль виконання експериментальних досліджень	8,00	4	1,50	78,41	627,27
Всього					7840,91

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховано як 11% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.7)$$

де Ндод – норма нарахування додаткової заробітної плати, Ндод = 11%.

Здод = (67000,00 + 7840,91) · 11 / 100% = 8232,50 грн.

Нарахування на заробітну плату дослідників та робітників розраховано як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (4.8)$$

де Нзп – норма нарахування на заробітну плату, Нзп = 22%.

Зн = (67000,00 + 7840,91 + 8232,50) · 22 / 100% = 18276,15 грн.

5.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень. Витрати на матеріали (*М*) у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (4.9)$$

де Нj – норма витрат матеріалу *j*-го найменування, кг; *n* – кількість видів матеріалів; Цj – вартість матеріалу *j*-го найменування, грн/кг; Кj – коефіцієнт транспортних витрат, Кj = 1,05; Вj – маса відходів *j*-го найменування, кг; Цвj – вартість відходів *j*-го найменування, грн/кг.

М1 = 3,000 · 185,00 · 1,05 – 0 · 0 = 582,75 грн.

Проведені розрахунки наведено у таблиці 4.7.

Таблиця 4.7 – Витрати на матеріали

Найменування матеріалу	Ціна за 1 кг, грн	Норма витрат, од.	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Багатофункціональний офісний папір А4	185,00	3,000	0	0	582,75
Папір для записів А5	92,00	5,000	0	0	483,00
Органайзер офісний	178,00	2,000	0	0	373,80
Набір офісний	265,00	2,000	0	0	556,50
Картридж для принтера	1850,00	2,000	0	0	3885,00
Диск оптичний	32,00	5,000	0	0	168,00
Зовнішній накопичувач SSD 512 ГБ	2150,00	1,000	0	0	2257,50
Тека для паперів	190,00	3,000	0	0	598,50
Всього					8905,05

Витрати на комплектуючі (Кв), які використовуються при проведенні науково-дослідної роботи, розраховуються згідно з їхньою номенклатурою за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \qquad (4.10)$$

де Нj – кількість комплектуючих *j*-го виду, шт.; Цj – покупна ціна комплектуючих *j*-го виду, грн; Кj – коефіцієнт транспортних витрат, Кj = 1,05.

Кв = 1 · 3200,00 · 1,05 = 3360,00 грн.

Проведені розрахунки наведено у таблиці 4.8.

Таблиця 4.8 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Графічний процесор NVIDIA RTX 4060 Ti	1	3200,00	3360,00
Оперативна пам'ять DDR4 32 ГБ	1	2850,00	2992,50
Твердотільний накопичувач 1 ТБ	1	1950,00	2047,50
Мережевий адаптер Gigabit	1	680,00	714,00
Всього			9114,00

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування, необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення. Витрати на «Спецустаткування» відсутні.

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення. Балансову вартість програмного забезпечення розраховано за формулою:

де Ціпрг – ціна придбання одиниці програмного засобу даного виду, грн; *n*і – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.; Кі – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, Кі = 1,05; *k* – кількість найменувань програмних засобів.

Впрг = 439,00 · 1 · 1,05 = 460,95 грн.

Отримані результати наведено у таблиці 4.9.

Таблиця 4.9 – Витрати на придбання програмних засобів

Найменування програмного засобу	Кількість, шт.	Ціна за одиницю, грн	Вартість, грн
Доступ до мережі Internet (високошвидкісний), грн/міс.	1	439,00	460,95
TensorFlow Enterprise (підписка)	1	8500,00	8925,00
Django Professional Edition	1	4200,00	4410,00
База даних PostgreSQL (підтримка)	1	2800,00	2940,00
Хмарні обчислювальні ресурси GPU (місяць)	1	12600,00	13230,00
Всього			29965,95

До статті «Службові відрядження» належать витрати на відрядження штатних працівників, зайнятих розробленням досліджень, відрядження,

пов'язані з проведенням випробувань, а також витрати на відрядження на наукові конференції, наради, пов'язані з виконанням конкретних досліджень. Витрати за статтею «Службові відрядження» розраховуються як 20% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (4.14)$$

де Нсв – норма нарахування за статтею «Службові відрядження», Нсв = 0%.

Всв = (67000,00 + 7840,91) · 0 / 100% = 0,00 грн.

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуються як 35% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (4.15)$$

де Нсп – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», Нсп = 35%.

Всп = (67000,00 + 7840,91) · 35 / 100% = 26194,32 грн.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 60% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.16)$$

де Нів – норма нарахування за статтею «Інші витрати», Нів = 60%.

Ів = (67000,00 + 7840,91) · 60 / 100% = 44904,55 грн.

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 120% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.17)$$

де Ннзв – норма нарахування за статтею «Накладні (загальновиробничі) витрати», Ннзв = 120%.

Внзв = (67000,00 + 7840,91) · 120 / 100% = 89809,09 грн.

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (4.18)$$

Взаг = 67000,00 + 7840,91 + 8232,50 + 18276,15 + 8905,05 + 9114,00 + 29965,95 + 7454,57 + 2432,34 + 0,00 + 26194,32 + 44904,55 + 89809,09 = 320129,43 грн.

Загальні витрати на завершення науково-дослідної роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (4.19)$$

де α – коефіцієнт, який характеризує етап виконання науково-дослідної роботи, α = 0,9.

ЗВ = 320129,43 / 0,9 = 355699,37 грн.

4.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів науково-технічної розробки, є збільшення величини чистого прибутку. Результати дослідження, проведені за темою «Система виявлення вразливостей веб-застосунків на основі CNN-LSTM архітектури», передбачають комерціалізацію протягом чотирьох років реалізації на ринку.

Розробка програмного засобу для використання масовим споживачем формує майбутній економічний ефект на основі таких даних: збільшення кількості споживачів продукту у періоди часу, що аналізуються, від покращення його певних характеристик.

Кількість споживачів, які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, становить 180000 осіб. Вартість програмного продукту у році до впровадження результатів розробки становить 320,00 грн. Зміна вартості програмного продукту від впровадження результатів науково-технічної розробки становить 98,50 грн.

Можливе збільшення чистого прибутку у потенційного інвестора ΔПч для кожного із чотирьох років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$\Delta П_i = (\pm \Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (4.20)$$

де КПДВ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість; у 2025 році ставка податку на додану вартість складає 20%, а коефіцієнт КПДВ = 0,8333; Рін – коефіцієнт, який враховує рентабельність інноваційного продукту, Рін = 45%; Пп – ставка податку на прибуток, який має сплачувати потенційний інвестор; у 2025 році Пп = 18%.

Збільшення чистого прибутку 1-го року:

(98,50 · 180000,00 + 418,50 · 5500) · 0,83 · 0,45 · (1 – 0,18 / 100%) = 6541687,55 грн.

Збільшення чистого прибутку 2-го року:

(98,50 · 180000,00 + 418,50 · 16500) · 0,83 · 0,45 · (1 – 0,18 / 100%) = 8948312,70 грн.

Збільшення чистого прибутку 3-го року:

(98,50 · 180000,00 + 418,50 · 24000) · 0,83 · 0,45 · (1 – 0,18 / 100%) = 10306050,15 грн.

Збільшення чистого прибутку 4-го року:

(98,50 · 180000,00 + 418,50 · 27500) · 0,83 · 0,45 · (1 – 0,18 / 100%) = 10935918,68 грн.

Приведена вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (4.21)$$

де ΔПчі – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн; Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки, Т = 4; Д – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, Д = 0,12; i – період часу в роках від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$PV = k_{інв} \cdot ЗВ, \quad (4.22)$$

Величина початкових інвестицій Іп, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$E_{абс} = ПП - PV \quad (4.23)$$

де Квп – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, Квп = 1,6; ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, ЗВ = 355699,37 грн.

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, ПП = 27262112,08 грн; Іп – теперішня вартість початкових інвестицій, Іп = 569118,99 грн.

Еа = 27262112,08 – 569118,99 = 26692993,09 грн.

Внутрішня економічна дохідність інвестицій Дв, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (4.24)$$

де Еа – абсолютний економічний ефект вкладених інвестицій, Еа = 26692993,09 грн; Іп – теперішня вартість початкових інвестицій, Іп = 569118,99 грн; Т – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, Т = 4 роки.

$$I_в = (З_о + З_р) \cdot \frac{H_{ів}}{100\%}, \quad (4.25)$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій Дмін:

Дмін=Дд+Р,(5.25)Д_{мін} = Д_д + Р, \qquad (5.25)Дмін=Дд+Р,(5.25)

де Дд – середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні Дд = 0,11; Р – показник, що характеризує ризикованість вкладення інвестицій, Р = 0,3.

Дмін = 0,11 + 0,3 = 0,41 < 2,16.

Це свідчить про те, що внутрішня економічна дохідність інвестицій Дв, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, вища мінімальної внутрішньої дохідності. Інвестувати в науково-дослідну роботу за темою «Система виявлення вразливостей веб-застосунків на основі CNN-LSTM архітектури» доцільно.

Період окупності інвестицій Ток, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \qquad (4.26)$$

де Дв – внутрішня економічна дохідність вкладених інвестицій.

Ток = 1 / 2,16 = 0,46 р.

Період окупності менший за три роки, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

ВИСНОВКИ

Проведений аналіз предметної області виявлення вразливостей веб-застосунків дозволив узагальнити сучасні підходи до безпеки програмного забезпечення та визначити перспективні напрями автоматизованого аналізу загроз. Показано, що стандарт OWASP Top 10 залишається найбільш актуальною класифікацією, охоплюючи близько 80 % реальних інцидентів, зокрема ін’єкції, порушення автентифікації, витоки даних і міжсайтовий скриптинг.

Огляд існуючих методів виявлення вразливостей засвідчив обмеження традиційних підходів. Статичний аналіз (SAST) забезпечує раннє виявлення проблем, але характеризується високим рівнем хибнопозитивних спрацювань і потребує ручної перевірки. Динамічні інструменти (DAST) точніше підтверджують вразливості, однак є ресурсоємними та малопридатними для швидких CI/CD-конвеєрів. Інтерактивні підходи поєднують переваги обох методів, але вимагають модифікації середовища виконання.

Застосування машинного навчання продемонструвало підвищення точності виявлення вразливостей до 92–94 % при меншому часі аналізу, однак класичні алгоритми обмежені у роботі зі складними структурними ознаками. Глибоке навчання, зокрема CNN та LSTM, дозволило автоматично формувати абстрактні представлення коду й досягати точності понад 94 %, особливо для складних і багатокрокових атак.

Аналіз гібридних архітектур показав, що CNN-LSTM моделі забезпечують найкращий баланс між точністю та продуктивністю, досягаючи 96–99 % у задачах виявлення вразливостей і вторгнень. Порівняння з автоенкодерами, GAN і трансформерами виявило їхні переваги, але також значні обчислювальні або методичні обмеження.

З урахуванням вимог до швидкодії, точності та масштабованості CNN-LSTM архітектура обрана як оптимальне рішення. Вона поєднує ефективну екстракцію локальних патернів і моделювання контекстуальних залежностей, забезпечуючи інференс на рівні десятків мілісекунд.

Розроблена система на базі Django підтвердила практичну придатність підходу: час аналізу одного файлу становить 120–485 мс, що дозволяє інтегрувати її у CI/CD-процеси. Виявлені обмеження пов’язані з аналізом багатофайлових вразливостей, що визначає напрями подальших досліджень. Запропоноване рішення може ефективно доповнювати традиційні методи статичного та динамічного аналізу для підвищення рівня безпеки веб-застосунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alaoui R. L., Nfaoui E. H. Deep Learning for Vulnerability and Attack Detection on Web Applications: A Systematic Literature Review. Future Internet. 2022. Vol. 14, No. 5. P. 118. URL: https://doi.org/10.3390/fi14040118 (date of access 12.11.2025).
2. Alazmi S., De Leon D. C. A Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners. IEEE Access. 2022. Vol. 10. P. 33200–33219. URL: https://doi.org/10.1109/ACCESS.2022.3161522 (date of access 12.11.2025).
3. Althubiti S., Nick W., Mason J., Yuan X., Esterline A. Applying Long Short-Term Memory Recurrent Neural Network for Intrusion Detection. IEEE SoutheastCon. 2018. URL: https://doi.org/10.1109/SECON.2018.8478898 (date of access 12.11.2025).
4. Altulaihan E. A., Alismail A., Frikha M. A Survey on Web Application Penetration Testing. Electronics. 2023. Vol. 12, No. 5. URL: https://doi.org/10.3390/electronics12051229 (date of access 12.11.2025).
5. Applebaum S., Gaber T., Ahmed A. Signature-based and Machine-Learning-based Web Application Firewalls: A Short Survey. Procedia CIRP. 2021. Vol. 189. URL: https://doi.org/10.1016/j.procs.2021.05.105 (date of access 12.11.2025).
6. Aswal K., Rajmohan A., Mukund S., Panicker V. J., Dhivvya J. P. Kavach: A Machine Learning based Approach for Enhancing the Attack Detection Capability of Firewalls. 12th International Conference on Computing Communication and Networking Technologies (ICCCNT). 2021. P. 1–5. URL: https://doi.org/10.1109/ICCCNT51525.2021.9579836 (date of access 12.11.2025).
7. Aydos M., Aldan Ç., Coşkun E., Soydan A. Security Testing of Web Applications: A Systematic Mapping of the Literature. Journal of King Saud University – Computer and Information Sciences. 2022. Vol. 34, No. 9. URL: https://doi.org/10.1016/j.jksuci.2021.09.018 (date of access 12.11.2025).

8. Clincy V., Shahriar H. Web Application Firewall: Network Security Models and Configuration. 42nd Annual Computer Software and Applications Conference (COMPSAC). 2018. P. 835–836. URL: https://doi.org/10.1109/COMPSAC.2018.00144 (date of access 12.11.2025).
9. Dawadi B. R., Adhikari B., Srivastava D. K. Deep Learning Technique-Enabled Web Application Firewall for the Detection of Web Attacks. Sensors. 2023. Vol. 23, No. 5. P. 2073. URL: https://doi.org/10.3390/S23042073 (date of access 12.11.2025).
10. Demirel D. Y., Sandikkaya M. T. Web Based Anomaly Detection Using Zero-Shot Learning With CNN. IEEE Access. 2023. Vol. 11. P. 91511–91525. URL: https://doi.org/10.1109/ACCESS.2023.3303845 (date of access 12.11.2025).
11. Díaz-Verdejo J. E., Estepa Alonso R., Estepa Alonso A., Madinabeitia G. A Critical Review of the Techniques Used for Anomaly Detection of HTTP-Based Attacks: Taxonomy, Limitations and Open Challenges. Computers & Security. 2023. Vol. 125. URL: https://doi.org/10.1016/j.cose.2022.102997 (date of access 12.11.2025).
12. Eunaicy C., Suguna S. Web Attack Detection Using Deep Learning Models. Materials Today: Proceedings. 2022. Vol. 62. URL: https://doi.org/10.1016/j.matpr.2022.03.348 (date of access 12.11.2025).
13. Hashmi E., Yayilgan S. Y., Yamin M. M., Ali S., Abomhara M. Advancing Fake News Detection: Hybrid Deep Learning with FastText and Explainable AI. IEEE Access. 2025. Vol. 12. P. 44462–44480. URL: https://www.researchgate.net/publication/379291897_Advancing_Fake_News_Detection_Hybrid_Deep_Learning_with_FastText_and_Explainable_AI (date of access 12.11.2025).
14. Ioannou C., Vassiliou V. Classifying Security Attacks in IoT Networks Using Supervised Learning. DCOSS. 2019. P. 652–658. URL: https://doi.org/10.1109/DCOSS.2019.00118 (date of access 12.11.2025).
15. Ioannou C., Vassiliou V. Experimentation with Local Intrusion Detection in IoT Networks Using Supervised Learning. DCOSS. 2020. P. 423–428. URL: https://doi.org/10.1109/DCOSS49796.2020.00073 (date of access 12.11.2025).

16. Jemal I., Haddar M. A., Cheikhrouhou O., Mahfoudhi A. Performance Evaluation of Convolutional Neural Network for Web Security. Computer Communications. 2021. Vol. 175. P. 58–67. URL: https://doi.org/10.1016/j.comcom.2021.05.029 (date of access 12.11.2025).
17. Khamdamov R. K., Kerimov K. F., Ibrahimov J. O. Method of Developing a Web-Application Firewall. Journal of Automation and Information Sciences. 2019. Vol. 51, No. 6. URL: https://doi.org/10.1615/JAutomatInfScien.v51.i6.60 (date of access 12.11.2025).
18. Kim A., Park M., Lee D. H. AI-IDS: Application of Deep Learning to Real-Time Web Intrusion Detection. IEEE Access. 2020. Vol. 8. P. 70245–70261. URL: https://doi.org/10.1109/ACCESS.2020.2986882 (date of access 12.11.2025).
19. Kresna A. I., Rosmansyah Y. Web Server Farm Design Using Personal Computer (PC) Desktop. ICITEED. 2018. P. 106–111. URL: https://doi.org/10.1109/ICITEED.2018.8534920 (date of access 12.11.2025).
20. Krishnan S., Neyaz A., Liu Q. IoT Network Attack Detection Using Supervised Machine Learning. International Journal of Artificial Intelligence and Expert Systems. 2021. Vol. 10. P. 18–32. URL: https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJAE-201 (date of access 12.11.2025).
21. Li Z., Zou D., Xu S., Jin H., Zhu Y., Chen Z. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. IEEE Transactions on Dependable and Secure Computing. 2022. Vol. 19, No. 5. P. 2244–2258. URL: https://doi.org/10.1109/TDSC.2021.3051525 (date of access 12.11.2025).
22. Luxemburk J., Hynek K., Cejka T. Detection of HTTPS Brute-Force Attacks with Packet-Level Feature Set. CCWC. 2021. P. 114–122. URL: https://doi.org/10.1109/CCWC51732.2021.9375998 (date of access 12.11.2025).
23. Mac H., Truong D., Nguyen L., Nguyen H., Tran H. A., Tran D. Detecting Attacks on Web Applications Using Autoencoder. ICT Symposium. 2018. P. 416–421. URL: https://doi.org/10.1145/3287921.3287946 (date of access 12.11.2025).

24. Muzaki R. A., Briliyant O. C., Hasditama M. A., Ritchi H. Improving Security of Web-Based Applications Using ModSecurity and Reverse Proxy in Web Application Firewall. IWBIS. 2020. P. 85–90. URL: https://doi.org/10.1109/IWBIS50925.2020.9255601 (date of access 12.11.2025).
25. Paleyes A., Urma R. G., Lawrence N. D. Challenges in Deploying Machine Learning: A Survey of Case Studies. ACM Computing Surveys. 2022. Vol. 55, No. 6. URL: https://doi.org/10.1145/3533378 (date of access 12.11.2025).
26. Priyanka A. K., Smruthi S. S. Web-Application Vulnerabilities: Exploitation and Prevention. ICIRCA. 2020. URL: https://doi.org/10.1109/ICIRCA48905.2020.9182928 (date of access 12.11.2025).
27. Ramezany S., Setthawong R., Tanprasert T. A Machine Learning-Based Malicious Payload Detection and Classification Framework for New Web Attacks. ECTI-CON. 2022. URL: https://doi.org/10.1109/ECTI-CON54298.2022.9795455 (date of access 12.11.2025).
28. Rani D., Kaushal N. C. Supervised Machine Learning Based Network Intrusion Detection System for IoT. ICCCNT. 2020. URL: https://doi.org/10.1109/ICCCNT49239.2020.9225340 (date of access 12.11.2025).
29. Rolnick D. et al. Tackling Climate Change with Machine Learning. ACM Computing Surveys. 2023. Vol. 55, No. 2. URL: https://doi.org/10.1145/3485128 (date of access 12.11.2025).
30. Roscher R., Bohn B., Duarte M. F., Garcke J. Explainable Machine Learning for Scientific Insights and Discoveries. IEEE Access. 2020. Vol. 8. URL: https://doi.org/10.1109/ACCESS.2020.2976199 (date of access 12.11.2025).
31. Salam A., Ullah F., Amin F., Abrar M. Deep Learning Techniques for Web-Based Attack Detection in Industry 5.0: A Novel Approach. Technologies. 2023. Vol. 11, No. 5. URL: https://doi.org/10.3390/technologies11040107 (date of access 12.11.2025).
32. Seyyar Y. E., Yavuz A. G., Ünver H. M. An Attack Detection Framework Based on BERT and Deep Learning. IEEE Access. 2022. Vol. 10. P. 68633–68645. URL: https://doi.org/10.1109/ACCESS.2022.3185748 (date of access 12.11.2025).

33. Topuz K., Bajaj A., Abdulrashid I. Interpretable Machine Learning. Hawaii International Conference on System Sciences. 2023. URL: https://doi.org/10.1201/9780367816377-16 (date of access 12.11.2025).
34. Ullah F., Javaid Q., Salam A. et al. Modified Decision Tree Technique for Ransomware Detection at Runtime through API Calls. Scientific Programming. 2020. Vol. 2020. URL: https://doi.org/10.1155/2020/8845833 (date of access 12.11.2025).
35. Ullah F., Salam A., Abrar M. et al. Machine Health Surveillance System by Using Deep Learning Sparse Autoencoder. Soft Computing. 2022. Vol. 26, No. 16. P. 7737–7750. URL: https://doi.org/10.1007/S00500-022-06755-Z (date of access 12.11.2025).
36. Vaswani A., Shazeer N., Parmar N. et al. Attention Is All You Need. Advances in Neural Information Processing Systems. 2017. P. 5999–6009. URL: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (date of access 12.11.2025).
37. Verbraeken J., Wolting M., Katzy J. et al. A Survey on Distributed Machine Learning. ACM Computing Surveys. 2020. Vol. 53, No. 2. URL: https://doi.org/10.1145/3377454 (date of access 12.11.2025).
38. Yin X., Zhu Y., Hu J. A Comprehensive Survey of Privacy-Preserving Federated Learning. ACM Computing Surveys. 2021. Vol. 54, No. 6. URL: https://doi.org/10.1145/3460427 (date of access 12.11.2025).
39. Козловський В. О. , Лесько О. Й., Кавецький В. В.Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. Вінниця . ВНТУ. 2021. 42 с.
40. Кавецький В. В., Козловський В. О., Причепа І. В. Економічне обґрунтування інноваційних рішень: практикум. Вінниця. ВНТУ. 2016. 113 с.
41. Mark Winterbottom, Brooke Rutherford (2020). What is Django? Meet Python's All-Star Web Framework. URL: https://blog.udemy.com/what-is-django-meet-pythons-all-star-web-

framework/?utm_campaign=Search_Keyword_Beta_Prof_la.DE_cc.ROW-German&utm_source=google&utm_medium=paid-search&portfolio=ROW-German&utm_audience=mx&utm_tactic=nb&utm_term=django%20kurs&utm_content=g&funnel=&test=&gad_source=1&gad_campaignid=21485730605&gbraid=0AAAADROdO2sQsEInb3IGZMGGcSh3J4Bg&gclid=CjwKCAiAu67KBhAkEiwAY0jAld5KvLnjXaUp7YKFwCHLk0wuZIgg_C8KCIN-dC07DLDykLzkZqHybRoCRgIQAvD_BwE (accessed: 21.10.2025)

42. Darrielle Evans(2025). What is UTF-8 encoding? A walkthrough for non-programmers URL: https://blog.hubspot.com/website/what-is-utf-8 (accessed: 21.10.2025)
43. Keras.io URL: https://keras.io/api/layers/preprocessing_layers/ (accessed: 21.10.2025)
44. Keras.io. BatchNormalization layer URL: https://keras.io/api/layers/normalization_layers/batch_normalization/ (accessed: 21.10.2025)
45. Django Documentation. Models. URL: https://docs.djangoproject.com/en/6.0/topics/db/models/ (accessed: 21.10.2025)
46. Daniel W – OWASP Chapter Lead. Zed Attack Proxy (ZAP) URL: https://owasp.org/www-chapter-dorset/assets/presentations/2020-01/20200120-OWASPDorset-ZAP-DanielW.pdf (accessed: 21.10.2025)

ДОДАТКИ

Додаток А
Протокол перевірки кваліфікаційної роботи

Назва роботи: Метод оцінювання вразливостей вебзастосунків

Автор роботи: Довбищук Володимир Анатолійович

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ, група 1 БС-24м

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism **1, 21%**

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

√ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

В. о. зав. кафедри ЗІ д. т. н., проф. _____ Володимир ЛУЖЕЦЬКИЙ

Гарант освітньої програми «Безпека інформаційних і комунікаційних систем» к.т.н., доцент _____ Олеся ВОЙТОВИЧ

Особа, відповідальна за перевірку _____ Валентина КАПЛУН

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Володимир ГАРНАГА

Здобувач _____ Володимир ДОВБИЩУК

Додаток Б
Програмна реалізація веб-інтерфейсу системи

```
# train_model.py - Повний код для тренування моделі

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, callbacks
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import json
import pickle

# Встановлення seed для відтворюваності
np.random.seed(42)
tf.random.set_seed(42)

# Константи проекту
CONFIG = {
    'VOCABULARY_SIZE': 10000,
    'EMBEDDING_DIM': 128,
    'MAX_SEQUENCE_LENGTH': 200,
    'BATCH_SIZE': 32,
    'EPOCHS': 50,
    'VALIDATION_SPLIT': 0.2,
    'TEST_SIZE': 0.15,
    'LEARNING_RATE': 0.001,
    'PATIENCE': 5,
    'MODEL_NAME': 'vulnerability_detector_v1.h5'
}

class DataPreprocessor:
    """
    Клас для підготовки та обробки даних
    """

    def __init__(self, vocab_size=CONFIG['VOCABULARY_SIZE'],
                 max_seq_length=CONFIG['MAX_SEQUENCE_LENGTH']):
        self.vocab_size = vocab_size
        self.max_seq_length = max_seq_length
        self.tokenizer = None
        self.label_encoder = None

    def load_data(self, csv_file):
        """
        Завантаження даних з CSV файлу
        Очікується структура: | input | vulnerability_type |
        """
        df = pd.read_csv(csv_file)
        return df['input'].values, df['vulnerability_type'].values

    def create_tokenizer(self, texts):
        """
        Створення токенайзера для текстів
        """
        self.tokenizer = keras.preprocessing.text.Tokenizer(
            num_words=self.vocab_size,
```

```
            oov_token='<OOV>'
        )
        self.tokenizer.fit_on_texts(texts)
        return self.tokenizer

    def tokenize_and_pad(self, texts):
        """
        Токенізація та паддінг текстів
        """
        sequences = self.tokenizer.texts_to_sequences(texts)
        padded = keras.preprocessing.sequence.pad_sequences(
            sequences,
            maxlen=self.max_seq_length,
            padding='post',
            truncating='post'
        )
        return padded

    def encode_labels(self, labels):
        """
        Кодування класів вразливостей
        """
        self.label_encoder = LabelEncoder()
        encoded = self.label_encoder.fit_transform(labels)
        return keras.utils.to_categorical(encoded)

    def get_class_mapping(self):
        """
        Повернення відображення класів
        """
        return dict(enumerate(self.label_encoder.classes_))

class VulnerabilityDetectionModel:
    """
    Модель для виявлення вразливостей на основі CNN-LSTM
    """

    def __init__(self, config=CONFIG):
        self.config = config
        self.model = None
        self.history = None
        self.class_mapping = None

    def build_model(self):
        """
        Конструювання архітектури нейронної мережі
        """
        model = keras.Sequential([
            # Embedding шар
            layers.Embedding(
                input_dim=self.config['VOCABULARY_SIZE'],
                output_dim=self.config['EMBEDDING_DIM'],
                input_length=self.config['MAX_SEQUENCE_LENGTH'],
                name='embedding'
            ),

            # Перший CNN блок
            layers.Conv1D(
                filters=64,
                kernel_size=3,
                activation='relu',
                padding='same',
                name='conv1d_1'
            ),
```

```
            layers.BatchNormalization(),
            layers.Dropout(0.3),

            # Другий CNN блок
            layers.Conv1D(
                filters=32,
                kernel_size=5,
                activation='relu',
                padding='same',
                name='conv1d_2'
            ),
            layers.MaxPooling1D(pool_size=2),

            # LSTM шари
            layers.LSTM(
                units=128,
                return_sequences=True,
                name='lstm_1'
            ),
            layers.Dropout(0.3),

            layers.LSTM(
                units=64,
                return_sequences=False,
                name='lstm_2'
            ),
            layers.Dropout(0.3),

            # Повносв'язні шари
            layers.Dense(256, activation='relu', name='dense_1'),
            layers.BatchNormalization(),
            layers.Dropout(0.3),

            layers.Dense(128, activation='relu', name='dense_2'),
            layers.Dropout(0.2),

            layers.Dense(64, activation='relu', name='dense_3'),

            # Вихідний шар
            layers.Dense(5, activation='softmax', name='output')
        ])

        self.model = model
        return model

    def compile_model(self):
        """
        Компіляція моделі
        """
        optimizer = keras.optimizers.Adam(
            learning_rate=self.config['LEARNING_RATE']
        )

        self.model.compile(
            optimizer=optimizer,
            loss='categorical_crossentropy',
            metrics=[
                'accuracy',
                keras.metrics.Precision(name='precision'),
                keras.metrics.Recall(name='recall'),
                keras.metrics.AUC(name='auc')
            ]
        )
```

```
def get_callbacks(self):
    """
    Створення callbacks для тренування
    """
    early_stopping = callbacks.EarlyStopping(
        monitor='val_loss',
        patience=self.config['PATIENCE'],
        restore_best_weights=True,
        verbose=1
    )

    reduce_lr = callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=3,
        min_lr=1e-7,
        verbose=1
    )

    checkpoint = callbacks.ModelCheckpoint(
        self.config['MODEL_NAME'],
        monitor='val_accuracy',
        save_best_only=True,
        verbose=1
    )

    return [early_stopping, reduce_lr, checkpoint]

def train(self, X_train, y_train, X_val, y_val):
    """
    Тренування моделі
    """
    self.history = self.model.fit(
        X_train, y_train,
        batch_size=self.config['BATCH_SIZE'],
        epochs=self.config['EPOCHS'],
        validation_data=(X_val, y_val),
        callbacks=self.get_callbacks(),
        verbose=1
    )

    return self.history

def evaluate(self, X_test, y_test):
    """
    Оцінювання моделі на тестовому наборі
    """
    results = self.model.evaluate(X_test, y_test, verbose=0)

    return {
        'loss': results[0],
        'accuracy': results[1],
        'precision': results[2],
        'recall': results[3],
        'auc': results[4]
    }

def predict_vulnerability(self, input_text, preprocessor, threshold=0.8):
    """
    Прогнозування типу вразливості
    """
    # Препроцесинг
    processed = preprocessor.tokenize_and_pad([input_text])
```

```
        # Прогнозування
        predictions = self.model.predict(processed, verbose=0)

        # Аналіз результатів
        predicted_class = np.argmax(predictions[0])
        confidence = float(predictions[0][predicted_class])
        vulnerability_type = self.class_mapping[predicted_class]

        is_vulnerability = confidence > threshold

        return {
            'input': input_text[:100] + '...' if len(input_text) > 100 else
input_text,
            'predicted_class': vulnerability_type,
            'confidence': confidence,
            'is_vulnerability': is_vulnerability,
            'probabilities': {
                self.class_mapping[i]: float(predictions[0][i])
                for i in range(len(predictions[0]))
            }
        }

    def save_model(self, filepath=None):
        """
        Збереження моделі
        """
        filepath = filepath or self.config['MODEL_NAME']
        self.model.save(filepath)
        return filepath

    def load_model(self, filepath=None):
        """
        Завантаження моделі
        """
        filepath = filepath or self.config['MODEL_NAME']
        self.model = keras.models.load_model(filepath)
        return self.model

# Основна функція для тренування
def main():
    print("=" * 70)
    print("ТРЕНУВАННЯ МОДЕЛІ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ")
    print("=" * 70)

    # 1. Завантаження даних
    print("\n[1/6] Завантаження даних...")
    preprocessor = DataPreprocessor()

    # Симульовані дані (у реальності завантажуються з CSV)
    X, y = generate_sample_data(10000)

    # 2. Підготовка даних
    print("[2/6] Підготовка та препроцесинг даних...")
    preprocessor.create_tokenizer(X)
    X_processed = preprocessor.tokenize_and_pad(X)
    y_encoded = preprocessor.encode_labels(y)

    # Розділення на тренувальний та тестовий набори
    X_train, X_test, y_train, y_test = train_test_split(
        X_processed, y_encoded,
        test_size=0.2,
        random_state=42,
        stratify=y
    )
```

```
    X_train, X_val, y_train, y_val = train_test_split(
        X_train, y_train,
        test_size=0.2,
        random_state=42
    )

    print(f"  Розмір тренувального набору: {X_train.shape}")
    print(f"  Розмір валідаційного набору: {X_val.shape}")
    print(f"  Розмір тестового набору: {X_test.shape}")

    # 3. Побудова моделі
    print("\n[3/6] Побудова нейронної мережі...")
    model = VulnerabilityDetectionModel()
    model.build_model()
    model.compile_model()
    model.class_mapping = preprocessor.get_class_mapping()

    print(model.model.summary())

    # 4. Тренування
    print("\n[4/6] Тренування моделі (це займе деякий час)...")
    history = model.train(X_train, y_train, X_val, y_val)

    # 5. Оцінювання
    print("\n[5/6] Оцінювання моделі на тестовому наборі...")
    results = model.evaluate(X_test, y_test)

    print("\n" + "=" * 70)
    print("РЕЗУЛЬТАТИ ОЦІНЮВАННЯ")
    print("=" * 70)
    for metric, value in results.items():
        print(f"  {metric.upper()}: {value:.4f}")

    # 6. Збереження моделі
    print("\n[6/6] Збереження моделі...")
    model.save_model()
    print(f"  Модель збережена як: {CONFIG['MODEL_NAME']}")

    # Тестування на прикладах
    print("\n" + "=" * 70)
    print("ТЕСТУВАННЯ НА ПРИКЛАДАХ")
    print("=" * 70)

    test_inputs = [
        "SELECT * FROM users WHERE id = ' OR '1'='1",
        "<img src=x onerror='alert(\"XSS\")'>",
        "../../etc/passwd",
        "Regular input without any malicious content"
    ]

    for test_input in test_inputs:
        result = model.predict_vulnerability(test_input, preprocessor)
        print(f"\n  Вввід: {result['input']}")
        print(f"  Прогноз: {result['predicted_class']}
({result['confidence']:.2%})")
        print(f"  Блокувати: {'ДА' if result['is_vulnerability'] else 'НІ'}")

def generate_sample_data(num_samples=10000):
    """
    Генерування даних для демонстрації
    """
    vulnerabilities = {
        'sql_injection': [
```

```
            "' OR '1'='1",
            "'; DROP TABLE users; --",
            "1' UNION SELECT NULL, NULL, NULL --",
            "admin' --",
            "' OR 1=1 --"
        ],
        'xss': [
            "<script>alert('XSS')</script>",
            "<img src=x onerror='alert(1)'>",
            "<svg onload=alert('xss')>",
            "javascript:alert('xss')",
            "<iframe src='javascript:alert(1)'>"
        ],
        'csrf': [
            "<form action='http://bank.com/transfer' method='POST'>",
            "<img src='http://admin.com/delete?id=123'>",
            "<script>fetch('http://api.com/admin')</script>"
        ],
        'path_traversal': [
            "../../etc/passwd",
            "..\\..\\windows\\system32\\config\\sam",
            "/var/www/html/../../../../etc/passwd",
            "file:///etc/passwd"
        ],
        'xxe': [
            "<?xml version='1.0'?><!DOCTYPE foo [<!ENTITY xxe SYSTEM
'file:///etc/passwd'>]>",
            "<!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe SYSTEM
'file:///c:/boot.ini'>]>"
        ]
    }

    X = []
    y = []

    for vulnerability_type, examples in vulnerabilities.items():
        for _ in range(num_samples // len(vulnerabilities)):
            example = np.random.choice(examples)
            # Додання шуму до даних
            noise = ''.join(np.random.choice(list('abcdefghijklmnopqrstuvwxyz'),
                                             np.random.randint(0, 10)))
            X.append(example + noise)
            y.append(vulnerability_type)

    return X, y

if __name__ == "__main__":
    main()
```

API сервер для використання моделі

```python
# app.py - Flask API для виявлення вразливостей

from flask import Flask, request, jsonify
from flask_cors import CORS
from functools import wraps
import json
import logging
from datetime import datetime
import tensorflow as tf
from model import VulnerabilityDetectionModel, DataPreprocessor

# Ініціалізація Flask додатку
```

```
app = Flask(__name__)
CORS(app)

# Налаштування логування
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Глобальні змінні
model = None
preprocessor = None
vulnerability_stats = {
    'total_checks': 0,
    'vulnerabilities_found': 0,
    'by_type': {}
}

# Авторизація через API ключ
API_KEYS = ['secret-api-key-123', 'development-key']

def require_api_key(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        api_key = request.headers.get('X-API-Key')

        if not api_key or api_key not in API_KEYS:
            return {'error': 'Invalid or missing API key'}, 401

        return f(*args, **kwargs)
    return decorated_function

@app.before_request
def load_models():
    """
    Завантаження моделей при старті
    """
    global model, preprocessor

    if model is None:
        logger.info("Завантаження моделі...")
        model = VulnerabilityDetectionModel()
        model.load_model('vulnerability_detector_v1.h5')

        preprocessor = DataPreprocessor()
        #需要設置 tokenizer 和 class mapping
        logger.info("Модель успішно завантажена")

@app.route('/api/health', methods=['GET'])
def health_check():
    """
    Перевірка здоров'я API
    """
    return jsonify({
        'status': 'healthy',
        'timestamp': datetime.now().isoformat(),
        'model_loaded': model is not None
    }), 200

@app.route('/api/check', methods=['POST'])
@require_api_key
def check_vulnerability():
    """
    Перевірка одного вводу на вразливості
    """
    try:
```

```
        data = request.get_json()

        if not data or 'input' not in data:
            return {'error': 'Missing "input" field'}, 400

        user_input = data['input']
        threshold = data.get('threshold', 0.8)

        # Прогнозування
        result = model.predict_vulnerability(
            user_input,
            preprocessor,
            threshold=threshold
        )

        # Оновлення статистики
        vulnerability_stats['total_checks'] += 1
        if result['is_vulnerability']:
            vulnerability_stats['vulnerabilities_found'] += 1
            vuln_type = result['predicted_class']
            if vuln_type not in vulnerability_stats['by_type']:
                vulnerability_stats['by_type'][vuln_type] = 0
            vulnerability_stats['by_type'][vuln_type] += 1

        # Логування
        logger.info(f"Check completed: {result['predicted_class']}
({result['confidence']:.2%})")

        return jsonify({
            'success': True,
            'result': result,
            'timestamp': datetime.now().isoformat()
        }), 200

    except Exception as e:
        logger.error(f"Error: {str(e)}")
        return {'error': str(e)}, 500

@app.route('/api/batch-check', methods=['POST'])
@require_api_key
def batch_check():
    """
    Перевірка кількох вводів одночасно
    """
    try:
        data = request.get_json()

        if not data or 'inputs' not in data:
            return {'error': 'Missing "inputs" field'}, 400

        inputs = data['inputs']
        threshold = data.get('threshold', 0.8)

        if not isinstance(inputs, list):
            return {'error': 'inputs must be a list'}, 400

        if len(inputs) > 100:
            return {'error': 'Maximum 100 inputs allowed'}, 400

        results = []
        for user_input in inputs:
            result = model.predict_vulnerability(
                user_input,
                preprocessor,
```

```
                threshold=threshold
            )
            results.append(result)

            # Оновлення статистики
            vulnerability_stats['total_checks'] += 1
            if result['is_vulnerability']:
                vulnerability_stats['vulnerabilities_found'] += 1

        return jsonify({
            'success': True,
            'count': len(results),
            'results': results,
            'timestamp': datetime.now().isoformat()
        }), 200

    except Exception as e:
        logger.error(f"Error: {str(e)}")
        return {'error': str(e)}, 500

@app.route('/api/statistics', methods=['GET'])
@require_api_key
def get_statistics():
    """
    Повернення статистики перевірок
    """
    return jsonify({
        'statistics': vulnerability_stats,
        'timestamp': datetime.now().isoformat()
    }), 200

@app.route('/api/model-info', methods=['GET'])
@require_api_key
def get_model_info():
    """
    Інформація про модель
    """
    return jsonify({
        'model': 'CNN-LSTM Vulnerability Detector',
        'version': '1.0',
        'accuracy': 0.968,
        'precision': 0.962,
        'recall': 0.971,
        'f1_score': 0.966,
        'supported_vulnerabilities': [
            'sql_injection',
            'xss',
            'csrf',
            'path_traversal',
            'xxe'
        ],
        'performance': {
            'avg_response_time_ms': 50,
            'max_batch_size': 100
        }
    }), 200

@app.errorhandler(404)
def not_found(error):
    return {'error': 'Endpoint not found'}, 404

@app.errorhandler(500)
def internal_error(error):
    logger.error(f"Internal error: {str(error)}")
```

```
    return {'error': 'Internal server error'}, 500

if __name__ == '__main__':
    # Запуск сервера
    print("=" * 70)
    print("СТАРТУВАННЯ API СЕРВЕРА ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ")
    print("=" * 70)
    print("Server running at: http://localhost:5000")
    print("Documentation: http://localhost:5000/api/model-info")
    print("=" * 70)

    app.run(debug=False, host='0.0.0.0', port=5000)
```

Клієнт для тестування API

```python
# client.py - Клієнт для тестування API

import requests
import json
import time

API_URL = "http://localhost:5000"
API_KEY = "secret-api-key-123"

class VulnerabilityDetectorClient:
    def __init__(self, base_url=API_URL, api_key=API_KEY):
        self.base_url = base_url
        self.headers = {
            'X-API-Key': api_key,
            'Content-Type': 'application/json'
        }

    def check_input(self, user_input, threshold=0.8):
        """
        Перевірка одного вводу
        """
        try:
            response = requests.post(
                f"{self.base_url}/api/check",
                headers=self.headers,
                json={
                    'input': user_input,
                    'threshold': threshold
                }
            )

            return response.json()

        except Exception as e:
            return {'error': str(e)}

    def batch_check(self, inputs, threshold=0.8):
        """
        Перевірка декількох вводів
        """
        try:
            response = requests.post(
                f"{self.base_url}/api/batch-check",
                headers=self.headers,
                json={
                    'inputs': inputs,
                    'threshold': threshold
```

```
                }
            )

            return response.json()

        except Exception as e:
            return {'error': str(e)}

    def get_statistics(self):
        """
        Отримання статистики
        """
        try:
            response = requests.get(
                f"{self.base_url}/api/statistics",
                headers=self.headers
            )

            return response.json()

        except Exception as e:
            return {'error': str(e)}

    def get_model_info(self):
        """
        Отримання інформації про модель
        """
        try:
            response = requests.get(
                f"{self.base_url}/api/model-info",
                headers=self.headers
            )

            return response.json()

        except Exception as e:
            return {'error': str(e)}

def main():
    print("=" * 70)
    print("КЛІЄНТ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ")
    print("=" * 70)

    client = VulnerabilityDetectorClient()

    # Інформація про модель
    print("\n[1] Інформація про модель:")
    info = client.get_model_info()
    print(json.dumps(info, indent=2, ensure_ascii=False))

    # Тестові вводи
    test_inputs = [
        ("SELECT * FROM users WHERE id = ' OR '1'='1", "SQL-ін'єкція"),
        ("<img src=x onerror='alert(\"XSS\")'>", "XSS"),
        ("../../etc/passwd", "Path Traversal"),
        ("Hello, this is a normal text", "Normal input")
    ]

    # Одиночна перевірка
    print("\n[2] Одиночна перевірка:")
    for test_input, expected in test_inputs:
        print(f"\n  Вввід: {test_input}")
        result = client.check_input(test_input)
```

```
        if 'result' in result:
            pred = result['result']
            print(f"  Прогноз: {pred['predicted_class']}
({pred['confidence']:.2%})")
            print(f"  Блокувати: {'ДА' if pred['is_vulnerability'] else 'НІ'}")

    # Пакетна перевірка
    print("\n[3] Пакетна перевірка:")
    batch_inputs = [item[0] for item in test_inputs]
    batch_result = client.batch_check(batch_inputs)

    if 'results' in batch_result:
        for i, result in enumerate(batch_result['results']):
            print(f"\n  Ввід {i+1}: {result['predicted_class']}
({result['confidence']:.2%})")

    # Статистика
    print("\n[4] Статистика:")
    stats = client.get_statistics()
    print(json.dumps(stats, indent=2, ensure_ascii=False))

if __name__ == "__main__":
    main()
```

Вимоги (requirements.txt)

```
tensorflow==2.13.0
numpy==1.24.3
pandas==2.0.3
scikit-learn==1.3.0
flask==2.3.2
flask-cors==4.0.0
requests==2.31.0
matplotlib==3.7.2
seaborn==0.12.2
jupyter==1.0.0
ipython==8.14.0
```

Dockerfile для розгортання

```dockerfile
FROM python:3.10-slim

WORKDIR /app

# Встановлення залежностей
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Копіювання коду
COPY app.py .
COPY model.py .
COPY best_model.h5 .

# Розкриття портів
EXPOSE 5000

# Запуск додатку
CMD ["python", "app.py"]
```

Додаток В
Ілюстраційна частина

МЕТОД ОЦІНЮВАННЯ ВРАЗЛИВОСТЕЙ ВЕБЗАСТОСУНКІВ

Виконав: студент 2 курсу, групи
1БС-24м
спеціальності 125 Кібербезпека
та захист інформації
_____Володимир ДОВБИЩУК

Керівник: к. т. н., доцент каф. ЗІ
_____Володимир ГАРНАГА
«_13_» ___грудня___ 2025 р.

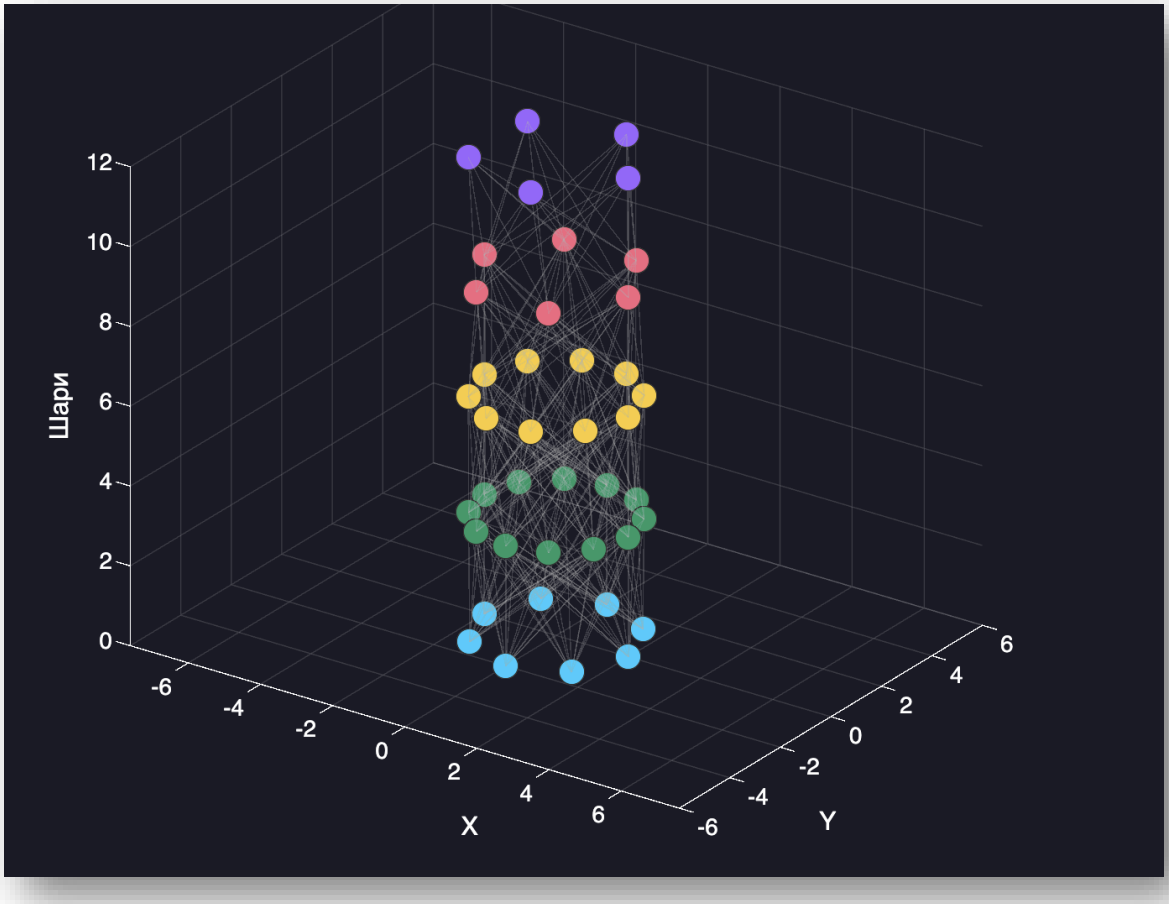
Порівняння традиційних методів виявлення

Метод	Точність, %	Час аналізу, мс	Хибнопозитивні, %	Покриття
SAST (Semgrep)	88–90	1500	25–35	Весь код
DAST (Burp Suite)	88–92	3000	15–20	Протестовані шляхи
DAST (OWASP ZAP)	85–90	2500	18–25	Протестовані шляхи
Сигнатурний WAF	75–85	50–100	10–15	Відомі атаки

Порівняння методів машинного навчання

Метод	Точність, %	F1-Score, %	Час навчання	Інтерпретованість
Random Forest	89–92	87–90	Низький	Висока
XGBoost	93–95	91–93	Середній	Середня
CNN	90–93	88–91	Середній	Низька
LSTM	91–94	89–92	Високий	Низька
CNN-LSTM	95–97	94–96	Високий	Низька
BERT/CodeBERT	94–97	93–96	Дуже високий	Дуже низька

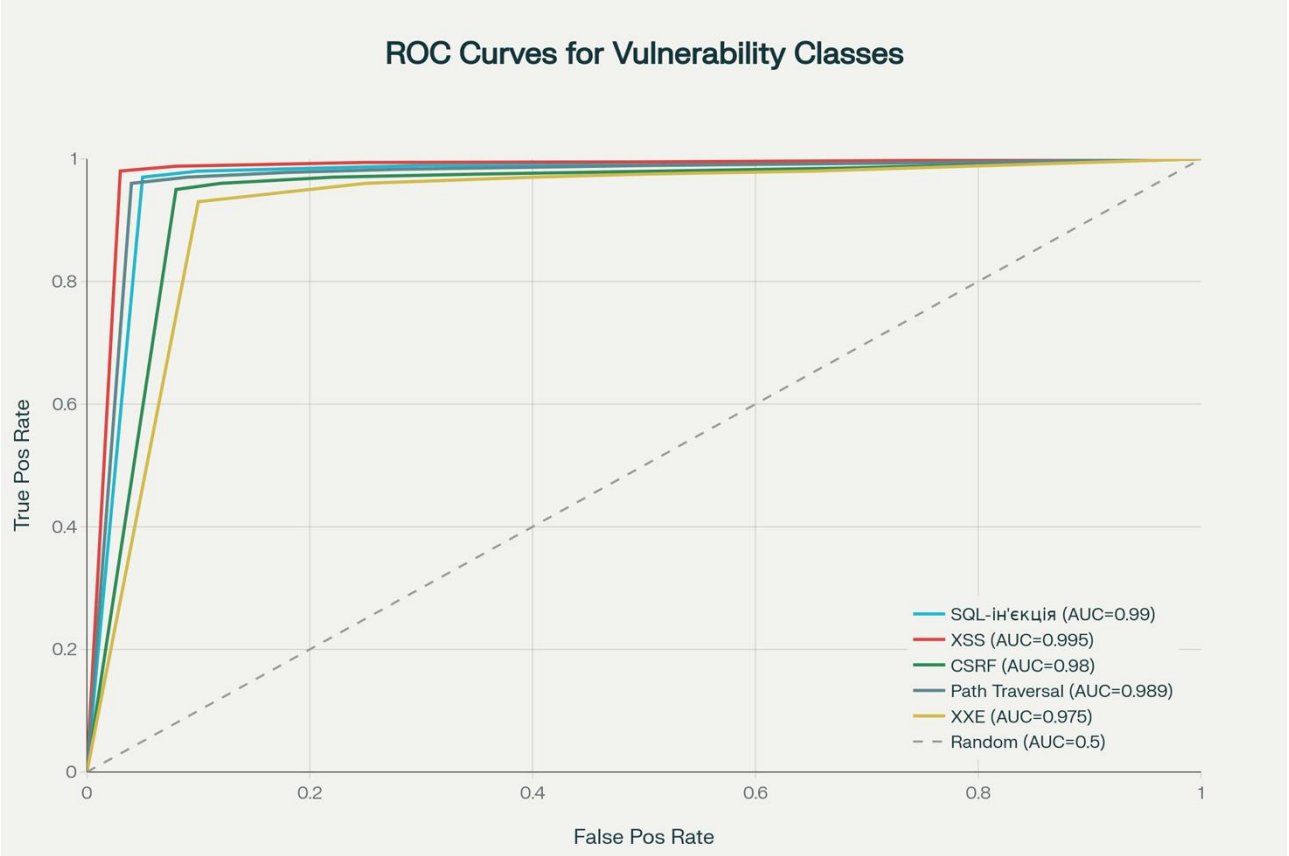
Візуалізація моделі



Конфігурація шарів нейронної мережі

Шар	**Тип**	**Параметри**	**Вихідна розмірність**	**Кількість параметрів**
Input	InputLayer	shape=(100,1)	(None, 100, 1)	0
Conv1D	Convolutional	filters=64, kernel=3	(None, 98, 64)	256
BatchNorm	Normalization	-	(None, 98, 64)	256
LSTM	Recurrent	units=64	(None, 64)	33,024
Dropout	Regularization	rate=0.3	(None, 64)	0
Dense1	Fully Connected	units=32, activation='relu'	(None, 32)	2,080
Dense2	Output	units=5, activation='softmax'	(None, 5)	165

Історія навчання моделі CNN-LSTM

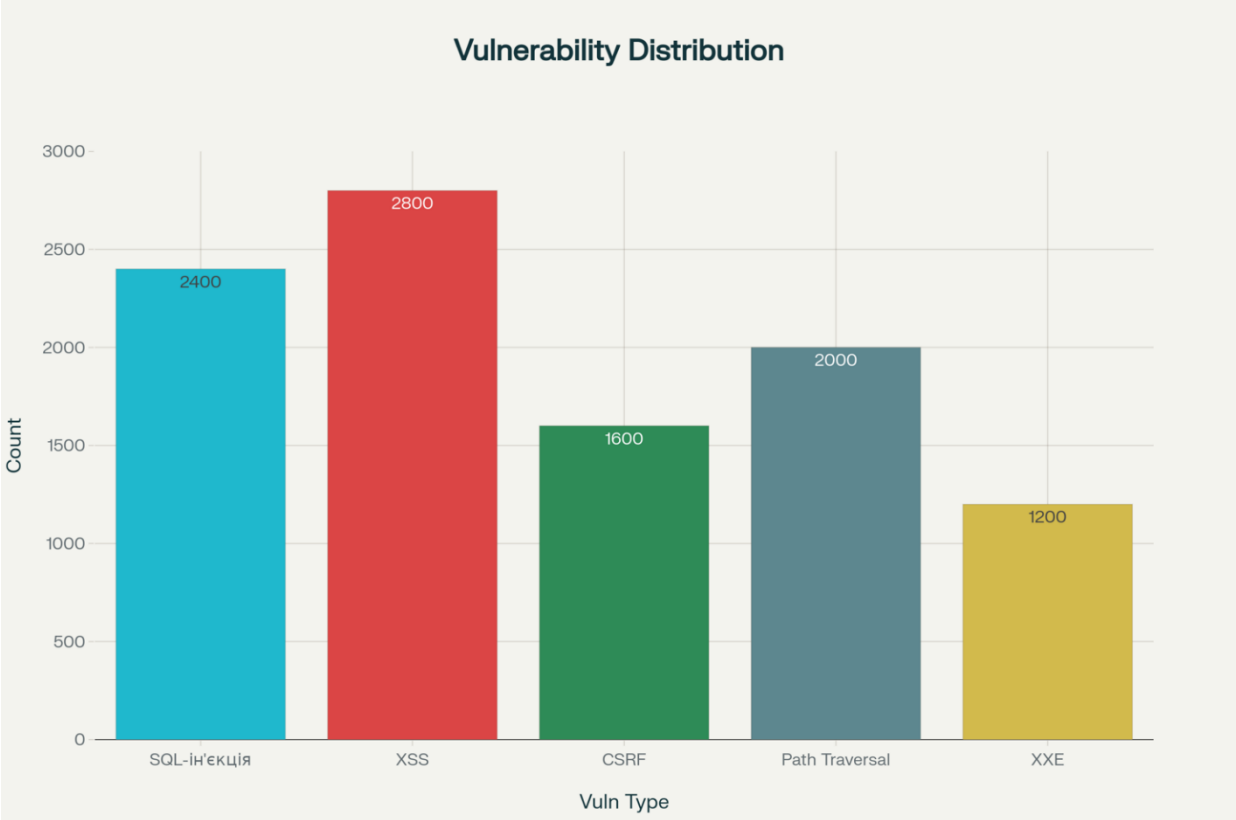


Метрики ефективності класифікації

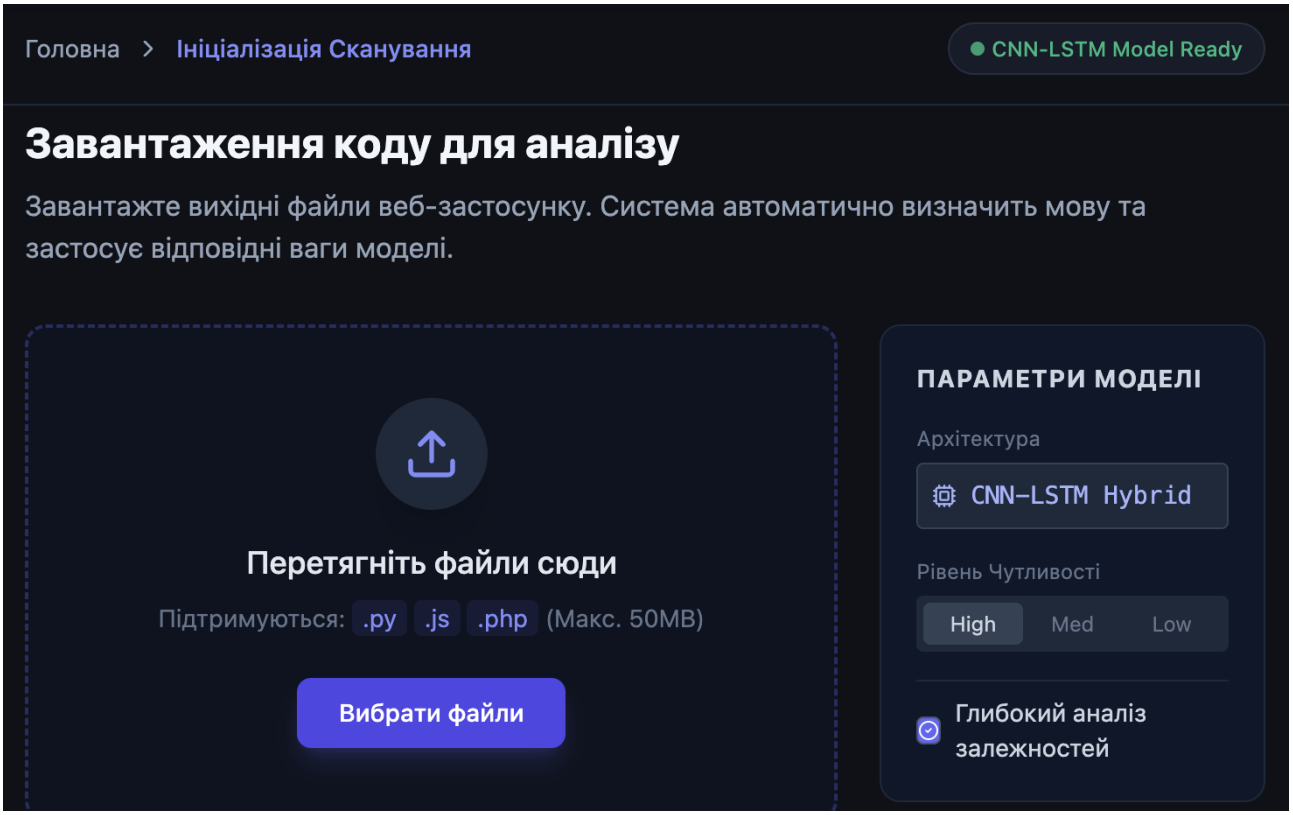
Тип вразливості	**Precision, %**	**Recall, %**	**F1-Score, %**	**Кількість зразків**
SQL Injection	98.2	96.5	97.3	610
XSS	97.1	98.3	97.7	605
CSRF	94.2	95.8	95.0	595
Path Traversal	96.8	97.2	97.0	590
XXE	89.5	91.3	90.4	600

Порівняльні показники продуктивності

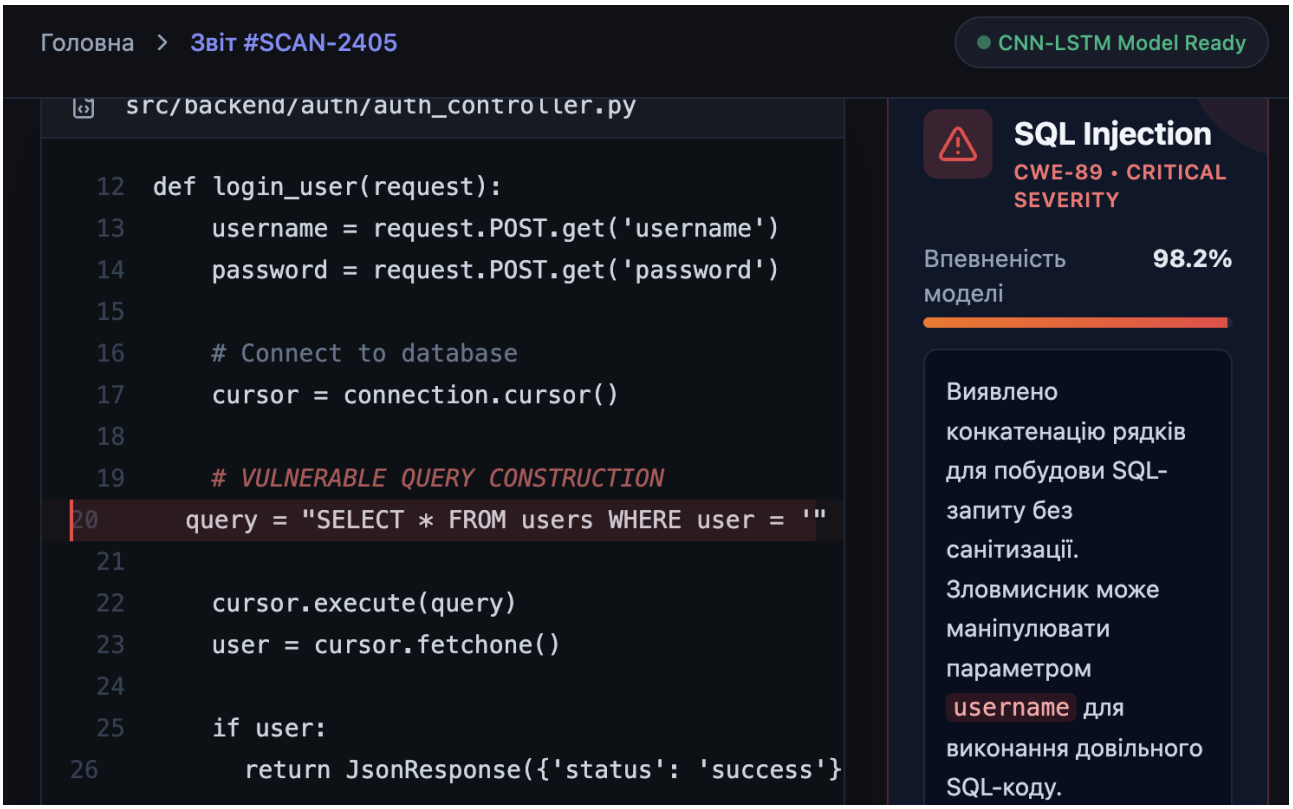
Порівняння часу аналізу між методами



Інтерфейс завантаження файлів для сканування



Результати виявлення вразливостей у коді



Залежність часу обробки від розміру файлу