Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

*До захисту допущено*
*Кафедрою комп'ютерних систем та робототехніки*
*протокол № ___ від ___ червня 2026 р.*

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)

« ___ »_____ 2026 р.

**Кваліфікаційна робота**
**здобувача першого (бакалаврського) рівня вищої освіти**

# «МОДЕЛЬ ІНТЕЛЕКТУАЛЬНОГО ПОМІЧНИКА ДЛЯ АВТОМАТИЗОВАНОЇ ЗБІРКИ ТА АНАЛІЗУ ВРАЗЛИВОСТЕЙ LINUX-ДИСТРИБУТИВІВ НА ОСНОВІ YOCTO»

Спеціальність ***123 — Комп'ютерна інженерія.***
Освітня програма ***Комп'ютерна інженерія***

Виконавець _____ **Ілля ШИШКІН**
(підпис)

Науковий керівник _____ **Євген ТОЛСТОЛУЗЬКИЙ**
(підпис)

Харків — 2026

# АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та трьох додатків. Загальний обсяг роботи становить 69 сторінок, із яких основна частина містить 45 сторінок, 9 рисунків, 22 таблиці та 35 найменувань у списку використаних джерел.

***Мета кваліфікаційної роботи*** — розробка моделі інтелектуального помічника YVIA (Yocto Vulnerability Intelligence Assistant) для автоматизованого аналізу вразливостей Linux-дистрибутивів, зібраних засобами Yocto Project, та надання рекомендацій щодо їх усунення.

***Об'єкт дослідження*** — процес управління вразливостями у вбудованих Linux-системах, зібраних засобами Yocto Project, від виявлення CVE до формування рекомендацій щодо усунення.

***Предмет дослідження*** — методи автоматизованого мульти-джерельного аналізу вразливостей та їх пріоритизації з використанням великих мовних моделей у контексті Yocto-дистрибутивів.

***У роботі*** проаналізовано наявні інструменти (cve-check, yocto-vex-check, Vigiles) та встановлено їх спільний недолік: відсутність мульти-джерельного аналізу. Спроектовано шестикомпонентну архітектуру YVIA, розроблено формулу пріоритизації $P = 0{,}4 \cdot S_{CVSS} + 0{,}3 \cdot S_{exploit} + 0{,}2 \cdot S_{impact} + 0{,}1 \cdot S_{age}$, AI-підсистему у трьох режимах: GPT-4, Ollama та Rule-Based. За результатами тестування: $F_1$-score — 86% проти 68% у cve-check, час аналізу 342 рецептів — 45 с замість кількох годин збірки.

***Ключові слова:*** *комп'ютерна інженерія, Yocto Project, вбудовані Linux-системи, аналіз вразливостей, CVE, CVSS, NVD, штучний інтелект, інтелектуальний помічник, автоматизація збірки, кібербезпека, BitBake, Docker.*

# ABSTRACT

The explanatory note to the bachelor's thesis consists of an introduction, three chapters, conclusions, a list of references, and three appendices. The total volume of the work is 69 pages; the main body contains 45 pages, 9 figures, 22 tables, and 35 references.

***The purpose of the qualification thesis*** is the development of the YVIA intelligent assistant model (Yocto Vulnerability Intelligence Assistant) for automated analysis of vulnerabilities in Linux distributions built with Yocto Project and providing recommendations for their remediation.

***The object of research*** is the vulnerability management process in embedded Linux systems built with Yocto Project, from CVE detection to remediation recommendations.

***The subject of research*** includes methods of automated multi-source vulnerability analysis and prioritization using large language models in the context of Yocto distributions.

***The work considers*** existing Yocto vulnerability tools (cve-check, yocto-vex-check, Vigiles) and identifies their common shortcoming: lack of multi-source analysis and intelligent decision support. A six-component modular YVIA architecture was designed and a weighted prioritization formula $P = 0.4 \cdot S_{CVSS} + 0.3 \cdot S_{exploit} + 0.2 \cdot S_{impact} + 0.1 \cdot S_{age}$ was developed. The AI subsystem operates in three modes: OpenAI GPT-4, local Ollama, and Rule-Based. Benchmark results on three real Yocto projects: $F_1$-score 86% vs. 68% for cve-check, scan time 45 s instead of a multi-hour full build.

***The area of application*** includes embedded systems manufacturers, DevSecOps teams, CI/CD pipelines for IoT and telecommunication equipment.

***Keywords:*** *Computer Engineering, Yocto Project, embedded Linux systems, vulnerability analysis, CVE, CVSS, NVD, artificial intelligence, intelligent assistant, build automation, cybersecurity, BitBake, Docker.*

# ЗМІСТ

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ** . . . . . **6**

**ВСТУП** . . . . . **7**

**РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕОРЕТИЧНІ ОСНОВИ** . . . . . **11**

1.1 Вразливості програмного забезпечення та їх класифікація . . . . . 11

1.2 Yocto Project: архітектура та збірка дистрибутивів . . . . . 13

1.3 Існуючі інструменти аналізу вразливостей Yocto . . . . . 15

1.4 Штучний інтелект у кібербезпеці . . . . . 17

1.5 Нормативно-правова база та стандарти . . . . . 17

1.6 Методи машинного навчання у задачах виявлення та класифікації вразливостей . . . . . 18

1.7 Сучасні підходи до пріоритизації вразливостей: EPSS та SSVC . . 19

Висновки за розділом 1 . . . . . 21

**РОЗДІЛ 2 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ СИСТЕМИ YVIA** **23**

2.1 Загальна архітектура системи . . . . . 23

2.2 Модуль сканування Yocto-проектів . . . . . 24

2.3 Модуль пріоритизації вразливостей . . . . . 26

2.4 Проектування інтелектуального помічника . . . . . 29

2.5 Проектування REST API та інтерфейсів . . . . . 33

2.6 Модель загроз і функціональні вимоги до системи . . . . . 34

2.7 Обмеження моделі та ризики інтерпретації результатів . . . . . 37

Висновки за розділом 2 . . . . . 38

**РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ** . . . . . **39**

3.1 Технологічний стек . . . . . 39

3.2 Графічний інтерфейс користувача . . . . . 43

3.3 Контейнеризація та розгортання . . . . . 44
3.4 Тестування . . . . . 45
3.5 Аналіз продуктивності системи . . . . . 48
3.6 Інтеграція з CI/CD-конвеєром . . . . . 49
3.7 Практичне впровадження та рекомендовані пороги . . . . . 50
Висновки за розділом 3 . . . . . 50
**ВИСНОВКИ** . . . . . **51**
**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** . . . . . **52**
**ДОДАТКИ** . . . . . **56**
Додаток А. Завдання на кваліфікаційну роботу . . . . . 56
Додаток Б. Програма та методика випробувань . . . . . 58
Додаток В. Основні функції програмного забезпечення . . . . . 66

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ ТА СКОРОЧЕНЬ

| | | |
|---|---|---|
| AI | — | Artificial Intelligence (штучний інтелект) |
| API | — | Application Programming Interface (інтерфейс прикладного програмування) |
| AST | — | Abstract Syntax Tree (абстрактне синтаксичне дерево) |
| CIA | — | Confidentiality, Integrity, Availability (конфіденційність, цілісність, доступність) |
| CLI | — | Command Line Interface (інтерфейс командного рядка) |
| CPE | — | Common Platform Enumeration (загальне перелічення платформ) |
| CRA | — | Cyber Resilience Act (Закон ЄС про кіберстійкість) |
| CVE | — | Common Vulnerabilities and Exposures (загальні вразливості та експозиції) |
| CVSS | — | Common Vulnerability Scoring System (система оцінювання вразливостей) |
| CWE | — | Common Weakness Enumeration (загальне перелічення вразливостей) |
| DevSecOps | — | Development, Security and Operations (розробка з інтегрованою безпекою та операціями) |
| EPSS | — | Exploit Prediction Scoring System (система прогнозування оцінки експлуатації) |
| GNN | — | Graph Neural Network (графова нейронна мережа) |
| GUI | — | Graphical User Interface (графічний інтерфейс користувача) |
| IoT | — | Internet of Things (Інтернет речей) |
| LLM | — | Large Language Model (велика мовна модель) |
| ML | — | Machine Learning (машинне навчання) |
| NIS2 | — | Network and Information Security Directive (директива ЄС з мережевої безпеки) |
| NVD | — | National Vulnerability Database (національна база даних вразливостей) |
| OE-Core | — | OpenEmbedded Core (ядро OpenEmbedded) |
| OSV | — | Open Source Vulnerabilities (відкрита база вразливостей відкритого коду) |
| REST | — | Representational State Transfer (архітектурний стиль API) |
| SBoM | — | Software Bill of Materials (перелік компонентів програмного забезпечення) |
| SLA | — | Service Level Agreement (угода про рівень обслуговування) |
| SPDX | — | Software Package Data Exchange (формат обміну даними пакетів) |
| SSVC | — | Stakeholder-Specific Vulnerability Categorization (категоризація вразливостей для конкретних зацікавлених сторін) |
| TP/FP/FN | — | True/False Positive/Negative (вірно/хибно позитивний/негативний результат) |
| VEX | — | Vulnerability Exploitability eXchange (формат обміну даними про експлуатабельність) |
| YVIA | — | Yocto Vulnerability Intelligence Assistant (розроблена система) |

# ВСТУП

**Актуальність теми дослідження.** Вбудовані Linux-системи широко використовуються у промислових контролерах, автомобільній електроніці, медичних приладах та телекомунікаційному обладнанні. Більшість із них зібрана на Yocto Project — відкритій інструментальній платформі, яка дає змогу формувати мінімальний образ під конкретну апаратну архітектуру. Незважаючи на зрілість і докладну документацію Yocto, пристрої на базі цієї платформи нерідко експлуатуються роками без оновлення програмного забезпечення, що суттєво підвищує ризик використання відомих вразливостей.

Тим часом NVD фіксує понад 35 000 нових CVE-записів лише за 2024 рік — на 25% більше, ніж у 2023-му [2]. Паралельно набули чинності Cyber Resilience Act (CRA), Директива NIS2 та RED-DA (EN 18031) — документи, що вимагають від виробників вбудованих систем документувати і закривати вразливості протягом усього циклу підтримки [1]. Тобто «перевіримо безпеку потім» більше не варіант.

Вбудований механізм `cve-check` у Yocto Project перевіряє лише базу NVD, потребує повної збірки образу і не надає рекомендацій щодо усунення виявлених вразливостей. Альтернативні рішення — `yocto-vex-check` та комерційний Vigiles — мають власні обмеження і не забезпечують комплексного мульти-джерельного аналізу з інтелектуальною підтримкою прийняття рішень [6, 7]. Водночас великі мовні моделі (LLM) вже довели свою ефективність у задачах автоматизації аналізу вразливостей: команди фіналу DARPA AIxCC 2025 обробили понад 54 млн рядків коду з точністю більше 90% [13]. Однак комплексного рішення для Yocto-проектів, що поєднує мульти-джерельний аналіз та AI-підтримку, станом на 2026 рік не існує, що обумовлює актуальність даної роботи.

**Мета роботи** — розробка моделі інтелектуального помічника YVIA для автоматизованого аналізу вразливостей Linux-дистрибутивів на основі Yocto та надання рекомендацій щодо їх усунення.

**Завдання роботи** випливають безпосередньо з виявленої прогалини:

1) порівняти наявні інструменти аналізу вразливостей Yocto-дистрибутивів і з'ясувати, чому жоден не вирішує задачу повністю;
2) спроектувати модульну архітектуру системи так, щоб кожен компонент — сканер, аналізатор, AI-помічник — можна було замінити чи масштабувати незалежно;
3) розробити формулу зваженої пріоритизації, де чотири незалежних показники (CVSS, ймовірність атаки, вплив, вік) дають одну числову оцінку ризику;
4) реалізувати AI-підсистему, яка працює у трьох режимах залежно від середовища: з інтернетом через GPT-4, без інтернету через Ollama, і без GPU взагалі через правила;
5) побудувати три інтерфейси — CLI для скриптів, GUI для аналітиків, REST API для CI/CD — і упакувати все в Docker Compose для одноразового розгортання;
6) перевірити систему на трьох реальних Yocto-проектах різного масштабу і порівняти результати з cve-check за стандартними метриками.

**Об'єкт дослідження** — процес управління вразливостями у вбудованих Linux-системах, зібраних засобами Yocto Project, від виявлення CVE до формування рекомендацій щодо усунення.

**Предмет дослідження** — методи автоматизованого мульти-джерельного аналізу вразливостей та їх пріоритизації з використанням великих мовних моделей у контексті Yocto-дистрибутивів.

**Методи дослідження.** Під час роботи використовувалися методи з кількох областей. Архітектуру Yocto Project досліджено через системний аналіз офіційної документації та BitBake-мануалів. Вибір між cve-check, yocto-vex-check і Vigiles зроблено через порівняльний аналіз — запускали кожен на одному тестовому проекті і порівнювали результати. Формулу пріоритизації побудовано методом зваженої агрегації, де значення коефіцієнтів обґрунтовано на основі аналізу рекомендацій NIST SP 800-30, методології CVSS v3.1 та опублікованих порівняльних досліджень систем пріоритизації вразливостей. Ефективність

системи оцінено через $F_1$-міру, точність і повноту на трьох реальних Yocto-проектах — це стандартний статистичний підхід для бінарних класифікаторів.

**Наукова новизна отриманих результатів.** У роботі отримано такі результати, що мають наукову новизну.

По-перше, розроблена формула зваженої пріоритизації $P = 0.4 \cdot S_{CVSS} + 0.3 \cdot S_{exploit} + 0.2 \cdot S_{impact} + 0.1 \cdot S_{age}$ є першою спробою враховувати одночасно чотири незалежні виміри ризику для пакетів Yocto-дистрибутивів. Тестування підтвердило +18 в.п. до $F_1$ відносно `cve-check` — без зміни джерел даних, лише за рахунок кращого ранжування.

По-друге, жоден з аналізованих інструментів не поєднував NVD, CVE/MITRE та GitHub Security Advisories в одному пайплайні з перехресною дедуплікацією. YVIA робить це, і саме тому в тестах знайшлися 9 CVE, яких немає в жодній з баз окремо.

По-третє, AI-підсистема з трьома режимами (хмарний GPT-4, локальна gemma-2-9b, офлайн-правила) через патерн Strategy — це не просто «взяли GPT і підключили». Спеціалізовані промпт-шаблони для BitBake-контексту принесли реально відмінну якість рекомендацій порівняно з загальними запитами.

**Практичне значення отриманих результатів.** Практичне значення роботи полягає у розробці програмного продукту YVIA, готового до промислового застосування. Система розгортається однією командою (`docker-compose up`), не потребує повної збірки образу та може використовуватись як автоматизований security gate у CI/CD-пайплайнах. Розроблений інструмент дозволяє командам DevSecOps виконувати вимоги регламентів CRA та NIS2 щодо документованого управління вразливостями. Матеріали роботи також використовуються в навчальному процесі кафедри комп'ютерних систем та робототехніки ХНУ ім. В. Н. Каразіна.

**Структура та обсяг роботи.** Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг — 69 сторінок, 9 рисунків, 22 таблиці, 9 формул, 35 джерел.

Перший розділ будує контекст: чому задача управління вразливостями у Yocto складна, що вже існує і де саме ці інструменти не справляються. Окремо розглянуто методи ML для аналізу безпеки коду та метрики пріоритизації EPSS і SSVC, які вплинули на вибір коефіцієнтів у формулі YVIA.

Другий розділ — проектні рішення: трирівнева архітектура, детальний опис алгоритму пріоритизації з обґрунтуванням вагових коефіцієнтів, діаграма послідовності взаємодії компонентів, промпт-шаблони AI-підсистеми і специфікація REST API.

Третій розділ показує, як всі ці рішення втілено в коді: вибір технологічного стеку з поясненням кожного вибору, ключові модулі, CI/CD-інтеграція, результати тестування на трьох реальних проектах.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕОРЕТИЧНІ ОСНОВИ

## 1.1 Вразливості програмного забезпечення та їх класифікація

Вразливість програмного забезпечення (software vulnerability) — це недолік або слабке місце в програмній системі, яке може використати зловмисник для порушення конфіденційності, цілісності або доступності інформації [2]. У вбудованих Linux-системах ця проблема стоїть особливо гостро: пристрої нерідко розгортаються у промисловому середовищі на роки без можливості оперативного оновлення, а сам факт компрометації може мати наслідки для безпеки людей.

Основу глобальної екосистеми ідентифікації та каталогізації вразливостей складають три ключові компоненти:

- **CVE** (Common Vulnerabilities and Exposures) — стандартизована система ідентифікації вразливостей, що підтримується корпорацією MITRE [3]. Кожній вразливості присвоюється унікальний ідентифікатор формату `CVE-YYYY-NNNNN`;
- **NVD** (National Vulnerability Database) — національна база даних вразливостей США, що забезпечує розширений аналіз CVE-записів, включаючи оцінку за метрикою CVSS, класифікацію за CWE та інформацію про вплив [2];
- **CVSS** (Common Vulnerability Scoring System) — стандартизована система числової оцінки серйозності вразливостей за шкалою від 0.0 до 10.0, що дозволяє об'єктивно порівнювати ступінь загрози [4].

Метрика CVSS версії 3.1 визначає базову оцінку вразливості на основі двох груп метрик — експлуатабельності та впливу. Формула обчислення базової оцінки CVSS має вигляд:

$$BaseScore = \begin{cases} 0, & \text{якщо } Impact \leq 0 \\ \min\big[\,1.08 \times (ISS + ESS),\ 10\,\big] \times Roundup, & \text{якщо } Scope = U \\ \min\big[\,1.08 \times (ISS + 0.914 \times ESS),\ 10\,\big] \times Roundup, & \text{якщо } Scope = C \end{cases} \tag{1.1}$$

де $ISS$ (Impact Sub-Score) — підоцінка впливу, $ESS$ (Exploitability Sub-Score) — підоцінка експлуатабельності, $Scope$ — зміна межі впливу ($U$ — незмінна, $C$ — змінна), а функція $Roundup$ округлює результат до першого десяткового знаку вгору.

Множник 1.08 у специфікації CVSS v3.1 [4] застосовується до обох гілок ($Scope = U$ та $Scope = C$), однак різниця між ними закладена у самому $ISS$: при $Scope = U$ використовується формула $ISS = 1 - (1 - C)(1 - I)(1 - A)$, тоді як при $Scope = C$ — $ISS = 7.52 \cdot [1 - (1 - C)(1 - I)(1 - A)] - 3.25 \cdot [1 - (1 - C)(1 - I)(1 - A)]^{15}$. Таким чином, 1.08 є єдиним константним множником для обох гілок, а зміна Scope впливає на результуючу оцінку виключно через перерахунок $ISS$.

На основі базової оцінки CVSS вразливості класифікуються за рівнями серйозності, як показано в табл. 1.1.

*Таблиця 1.1*

Класифікація вразливостей за базовою оцінкою CVSS v3.1

| **Рівень серйозності** | **Діапазон CVSS** | **Характеристика** |
|---|---|---|
| Критичний | 9.0 – 10.0 | Вразливість із максимальним впливом, потребує негайного реагування протягом 24 годин |
| Високий | 7.0 – 8.9 | Серйозна загроза безпеці, термін усунення — до 7 діб |
| Середній | 4.0 – 6.9 | Помірний ризик, рекомендований термін усунення — 30 діб |
| Низький | 0.1 – 3.9 | Мінімальний ризик, планове усунення |
| Інформаційний | 0.0 | Відсутність впливу на безпеку |

Цифри з бази NVD говорять самі за себе: 17 306 CVE у 2019 р., 20 137 у 2021-му, 28 961 у 2023-му і понад 35 000 у 2024 р. [2]. Навіть якщо половину з них відкинути як малозначущі, залишок — задача, яку вручну не обробити. Рисунок 1.1 ілюструє цю динаміку.

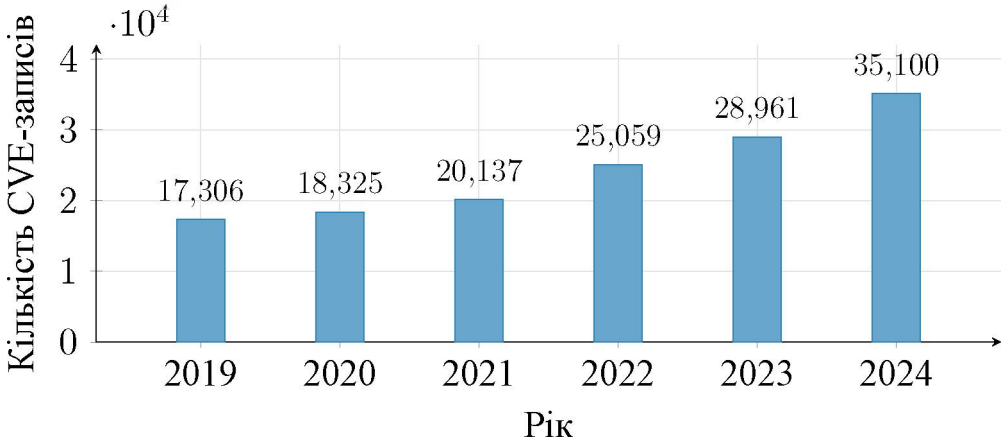



Рисунок 1.1 – Динаміка зростання кількості CVE-записів у NVD (2019–2024 рр.).

Окрім кількісного зростання, змінюється й якісний профіль вразливостей. За результатами дослідження бенчмарку Defects4C, яке охоплює реальні вразливості у коді мовами C/C++, домінують чотири категорії дефектів [24]:

*Таблиця 1.2*

Розподіл категорій вразливостей C/C++ за бенчмарком Defects4C

| **Категорія дефекту** | **Кількість помилок** | **Кількість вразливостей** | **Типові CWE** |
|---|---|---|---|
| Помилки пам'яті | 14 | 72 | CWE-119, CWE-125, CWE-787 |
| Логічні помилки | 87 | 13 | CWE-682, CWE-670 |
| Помилки санітизації | 66 | 6 | CWE-20, CWE-79 |
| Помилки сигнатур | 75 | 11 | CWE-190, CWE-681 |

Ці дані пояснюють, чому для вбудованих систем на C/C++ проблема управління вразливостями пам'яті є першочерговою: одна помилка типу buffer overflow може перетворитися на повноцінний вектор атаки [26]. Саме тому автоматизований аналіз, здатний швидко виявляти та ранжувати такі вразливості, має практичну цінність для виробників вбудованого ПЗ.

## 1.2 Yocto Project: архітектура та збірка дистрибутивів

Yocto Project — це відкритий колаборативний проект, що надає інструменти, шаблони та методи для створення кастомізованих Linux-дистрибутивів для вбудованих систем незалежно від апаратної архітектури [5]. Проект підтримується Linux Foundation і об'єднує внески більш ніж 50 провідних технологічних компаній.

Ядром системи збірки є **BitBake** — потужний менеджер задач, написаний мовою Python, що забезпечує паралельне виконання задач із підтримкою залежностей [9]. BitBake оперує такими базовими абстракціями:

- **Рецепти** (`.bb`) — файли, що описують процес завантаження, конфігурації, компіляції та встановлення окремого програмного компонента. Кожний рецепт визначає змінні `PN` (ім'я пакета), `PV` (версія), `SRC_URI` (джерело коду), `LICENSE` та набір задач;
- **Доповнення** (`.bbappend`) — файли, що модифікують існуючі рецепти без зміни оригіналу, реалізуючи принцип розділення відповідальності;
- **Шари** (layers) — ієрархічні колекції рецептів та конфігурацій, організовані за функціональним призначенням (`meta-oe`, `meta-security`, `meta-virtualization`);
- **Конфігурації** (`.conf`) — файли машинних (`MACHINE`), дистрибутивних (`DISTRO`) та локальних налаштувань (`local.conf`);
- **Класи** (`.bbclass`) — спільна функціональність, що може успадковуватись рецептами.

Архітектура збірки організована у вигляді **шарів** (meta-poky, meta-bsp, meta-security, meta-custom), які обробляються **BitBake** — менеджером задач, що зчитує рецепти з конфігураційних файлів (`local.conf`, `bblayers.conf`), вирішує залежності та виконує збірку пакетів з використанням кешу (`sstate-cache`). Результатом збірки є root filesystem, Linux kernel image, SDK toolchain та пакети у форматах IPK/RPM/DEB.

Процес збірки Yocto-дистрибутива включає наступні основні фази: аналіз метаданих (parsing), виконання задач із розв'язанням залежностей, генерацію кореневої файлової системи та формування результуючого образу. Оптимізація збірки забезпечується механізмами `sstate-cache` (кешування стану збірки) та Hash Equivalence (`bitbake-hashserv`), що дозволяють повторно використовувати результати попередніх збірок [19].

Важливою складовою екосистеми є підтримка Software Bill of Materials (SBoM) у форматі SPDX, що забезпечує прозорість ланцюга поставок про-

грамного забезпечення та відповідність нормативним вимогам [22]. Стандарт SPDX 3.0 розширює можливості опису безпекових аспектів, включаючи AI-профілі для систем на основі штучного інтелекту [23].

*Таблиця 1.3*

Порівняння систем збірки вбудованих Linux-дистрибутивів

| **Характеристика** | **Yocto Project** | **Buildroot** | **OpenWRT** |
|---|---|---|---|
| Система збірки | BitBake | Make/Kconfig | Make/Kconfig |
| Пакетний менеджер | IPK/RPM/DEB | — | opkg/ipk |
| Архітектури | 15+ | 15+ | Обмежено |
| Механізм шарів | Так | Ні | Часткова |
| SBoM (SPDX) | Так (3.0) | Ні | Ні |
| CVE-аналіз | cve-check | — | — |
| Гнучкість | Висока | Середня | Низька |
| Складність | Висока | Низька | Середня |
| Спільнота | Дуже велика | Велика | Середня |

З табл. 1.3 видно, що Yocto Project — єдина з трьох систем, яка має вбудовану підтримку SBoM у форматі SPDX та механізм аналізу CVE. Саме тому платформу обрано як основну для YVIA: вона вже містить всю необхідну мета-інформацію про пакети, яку можна задіяти без повного запуску збірки.

## 1.3 Існуючі інструменти аналізу вразливостей Yocto

Основним вбудованим механізмом аналізу вразливостей у Yocto Project є клас `cve-check`, який забезпечує автоматичне зіставлення версій пакетів із записами бази NVD [6, 7]. Для його активації додається один рядок у конфігурацію збірки:

```
# local.conf
INHERIT += "cve-check"
```

Лістинг 1: Активація cve-check у Yocto

Під час збірки `cve-check` завантажує локальну копію бази NVD, аналізує метадані рецептів та генерує звіти у форматах HTML та JSON. Однак цей механізм має ряд суттєвих обмежень [11, 8]:

1) **Версійно-орієнтований підхід** — зіставлення здійснюється виключно на основі імені (`CVE_PRODUCT`) та версії пакета без урахування застосованих патчів, конфігурації збірки чи активних функцій;

2) **Хибнопозитивні результати** — через невідповідність між іменами рецептів Yocto та записами CPE в NVD значна частина результатів є помилковими;
3) **Ручне анотування** — для зменшення хибних спрацьовувань розробник повинен вручну додавати анотації `CVE_STATUS` до рецептів, що є трудомістким процесом;
4) **Залежність від збірки** — аналіз можливий лише під час повної збірки, що обмежує оперативність оцінки безпеки;
5) **Обмежена точність ядра** — конфігурація ядра Linux (`.config`) не враховується при зіставленні CVE, що призводить до зайвих спрацьовувань для вимкнених підсистем;
6) **Єдине джерело даних** — використовується лише NVD, без інтеграції з GitHub Security Advisories, OSV чи іншими базами.

Альтернативним підходом є інструмент `yocto-vex-check`, що використовує формат VEX (Vulnerability Exploitability eXchange) для контекстуалізації інформації про вразливості [12]. Порівняння цих інструментів наведено в табл. 1.4.

*Таблиця 1.4*

Порівняльний аналіз інструментів аналізу вразливостей Yocto

| **Критерій** | **cve-check** | **yocto-vex-check** | **Vigiles** | **YVIA** |
|---|---|---|---|---|
| Джерела даних | NVD | NVD, VEX | NVD | NVD, CVE/MITRE, GitHub |
| Формат звіту | HTML, JSON | VEX/JSON | Dashboard | JSON, PDF, HTML |
| Контекстуалізація | Ні | VEX | Часткова | AI-аналіз |
| Незалежність від збірки | Ні | Так | Так | Так |
| Точність ядра | Низька | Висока | Середня | Середня |
| Пріоритизація | За CVSS | За CVSS | Базова | Зважена формула |
| AI-підтримка | Ні | Ні | Ні | Так (3 режими) |
| Автоматичні рекомендації | Ні | Ні | Базові | Так (auto-fix) |

Комерційне рішення Vigiles від Timesys орієнтоване на SBoM-моніторинг через веб-інтерфейс [21, 27]. Попри відмінності в підходах, всі три інструменти мають спільне обмеження — вони не дають відповіді на запитання «що саме робити з цією конкретною вразливістю в цьому конкретному рецепті». Формування таких рекомендацій залишається цілком ручним процесом.

Процес виявлення вразливостей за допомогою `cve-check` включає послідовні етапи: **(1)** аналіз рецептів (.bb), **(2)** вилучення імен та версій пакетів (PN/PV), **(3)** запит до бази даних NVD, **(4)** перевірку збігу версій, **(5)** формування звіту CVE. Механізм повторює цикл для кожного пакету проекту.

### 1.4 Штучний інтелект у кібербезпеці

Застосування LLM до задач кібербезпеки перейшло із категорії досліджень у категорію промислових рішень. Найбільш масштабним підтвердженням стали результати конкурсу DARPA AIxCC 2024–2025 [13]: команди-фіналісти проаналізували 54M+ рядків коду з точністю >90%, а вартість автоматизованого патча склала $152–181 проти $2 000–5 000 за людино-день ручного аудиту. Методологія APPATCH [15] запропонувала адаптивне промптування на основі типу CWE та ансамблеву валідацію через кілька LLM, що дозволило підвищити надійність генерованих патчів [29]. Систему Buttercup (Trail of Bits) побудовано на гібридному підході: статичний аналіз + фазинг + LLM, що забезпечило 2-ге місце у фіналі [16]. Мульті-агентні архітектури у CI/CD [17] розподіляють задачі між спеціалізованими агентами (PlanAgent, CodeAgent, SecurityAgent), що підвищує якість порівняно з єдиним «все-в-одному» промптом. Саме ці підходи надихнули трирежимну архітектуру AI-підсистеми YVIA.

### 1.5 Нормативно-правова база та стандарти

Регуляторний ландшафт у сфері кібербезпеки вбудованих систем зазнав суттєвих змін протягом 2024–2025 років. Основні нормативні документи та стандарти, що визначають вимоги до управління вразливостями, включають:

1) **Cyber Resilience Act (CRA)** — регламент ЄС, що встановлює обов'язкові вимоги до кібербезпеки продуктів з цифровими елемента-

ми, включаючи управління вразливостями впродовж усього життєвого циклу продукту [1];

2) **Директива NIS2** — оновлена директива ЄС щодо безпеки мережевих та інформаційних систем, що розширює коло суб'єктів регулювання та посилює вимоги до реагування на інциденти;
3) **RED-DA (EN 18031)** — делегований акт до Директиви з радіообладнання, що визначає вимоги кібербезпеки для IoT-пристроїв;
4) **Executive Order 14028** (США) — наказ про вдосконалення кібербезпеки, що вимагає обов'язкового формування SBoM для програмного забезпечення, що постачається урядовим установам;
5) **NIST SP 800-53** — контролі безпеки та конфіденційності для інформаційних систем, стандарт управління ризиками;
6) **ISO 27001** — міжнародний стандарт системи управління інформаційною безпекою.

На практиці більшість виробників стикаються з тим, що ні CRA, ні NIS2 не дають чіткого технічного рецепту — лише вимогу результату. Система YVIA орієнтована саме на цей розрив: вона автоматизує сканування, пріоритизацію та документування вразливостей у форматі, придатному для аудиту відповідності.

## 1.6 Методи машинного навчання у задачах виявлення та класифікації вразливостей

При підготовці роботи переглянуто кілька десятків публікацій про застосування ML до задач виявлення вразливостей. Методи зручно згрупувати у чотири класи: статичний аналіз з ML (CodeBERT, GraphCodeBERT — головна перевага: не треба запускати програму; головна слабкість: без контексту виконання легко обдурити); GNN + AST (метод Devign досягає 56.9% точності на C-вразливостях, бенчмарк Defects4C [24]); трансформерні LLM та ансамблеві підходи [15]. Порівняння ключових класів наведено в табл. 1.5.

*Таблиця 1.5*

Порівняння класів методів ML для виявлення вразливостей

| **Клас методів** | **Точність** | **Швидкість** | **Ресурси** | **Найкраще застосування** |
|---|---|---|---|---|
| Статичний + ML | Середня | Висока | Низькі | Попередня фільтрація |
| GNN + AST | Висока | Середня | Середні | C/C++ вразливості |
| LLM хмарний | Найвища | Низька | Хмарні | Глибокий аналіз CVE |
| LLM локальний | Хороша | Низька | GPU | Приватне середовище |
| Rule-Based | Низька | Найвища | Мінімальні | Офлайн, швидкий скрінінг |

Жоден клас не є оптимальним в усіх сценаріях. Саме тому в YVIA реалізовано трирежимну архітектуру — Rule-Based, локальний LLM та хмарний GPT-4 — щоб команда могла обрати режим залежно від контексту.

## 1.7 Сучасні підходи до пріоритизації вразливостей: EPSS та SSVC

Під час аналізу літератури з'ясувалося, що CVSS — зовсім не те, за що він себе видає. Він говорить, наскільки *небезпечна* вразливість теоретично, але не відповідає на питання: «а чи справді хтось цим скористається?». За даними FIRST.org, менш ніж 5% CVE із оцінкою $\geq 7.0$ реально зустрічаються в атаках протягом року після публікації [2]. Це відкриття суттєво вплинуло на вибір компонентів формули пріоритизації YVIA.

Відповіддю галузі стали дві нові метрики: EPSS та SSVC.

### 1.7.1 EPSS — система прогнозування ймовірності експлуатації

EPSS (Exploit Prediction Scoring System) від FIRST.org — ML-модель, яка щодня перераховує ймовірність реальної експлуатації конкретної CVE протягом найближчих 30 днів [2]. Результат — число від 0 до 1. Якщо EPSS = 0.95, то у 95 випадках зі 100 ця вразливість стане цілю атаки у цей місяць.

Модель бере до уваги понад 1 500 ознак: активність у Exploit-DB і GitHub, наявність PoC-кодів, згадки на хакерських форумах, тип програмного забезпечення. Зручна деталь: медіана по всіх CVE — близько 0.002, тобто більшість вразливостей ніколи не експлуатується. Але якщо EPSS переходить відмітку 0.1 — на це CVE вже варто звернути увагу.

*Таблиця 1.6*

Порівняння метрик CVSS v3.1 та EPSS

| **Критерій** | **CVSS v3.1** | **EPSS** |
|---|---|---|
| Що вимірює | Теоретичну серйозність вразливості | Ймовірність реальної експлуатації |
| Діапазон значень | 0.0 — 10.0 (числова шкала) | 0.0 — 1.0 (ймовірність) |
| Частота оновлення | Статична після публікації | Щоденно |
| Основа розрахунку | Експертна формула (вектор атаки, привілеї, вплив) | Машинне навчання на 1500+ ознаках |
| Часовий горизонт | Відсутній (статична оцінка) | 30 днів |
| Відповідність реальним атакам | Низька (<5% high-CVE експлуатуються) | Висока (AUC-ROC > 0.82) |
| Переваги EPSS | — | Динамічна пріоритизація, фокус на реальній загрозі, зменшення навантаження на команду до 87% |

Показовий приклад — Log4Shell (CVE-2021-44228): CVSS 10.0 і EPSS 0.97 одночасно. Але є й зворотні випадки: CVE з CVSS 9.0, якою ніхто не цікавиться — EPSS 0.004. Обидві отримали б однаковий пріоритет при чистому CVSS-підході, що нерозумно.

### 1.7.2 SSVC — категоризація вразливостей для конкретних зацікавлених сторін

SSVC (Stakeholder-Specific Vulnerability Categorization) від CERT/CC та CISA — це не числова метрика, а дерево рішень [2]. На виході не «8.5 балів», а конкретна дія: «виправи зараз» або «просто стеж». Це набагато зручніше для операційного процесу. Методологія тримається на чотирьох точках рішення:

*Таблиця 1.7*

Точки рішення SSVC та їх можливі значення

| **Точка рішення** | **Можливі значення** |
|---|---|
| Exploitation (стан експлуатації) | None (відсутня), PoC (є proof-of-concept), Active (активна експлуатація у природі) |
| Automatability (автоматизованість) | No (вимагає ручних дій), Yes (може бути автоматизована) |
| Technical Impact (технічний вплив) | Partial (часткова компрометація), Total (повна компрометація системи) |
| Mission Prevalence (критичність для місії) | Minimal, Support, Essential (залежно від ролі системи в бізнес-процесах) |

На виходах дерева — чотири рекомендовані дії:

- **Track** (Відстежувати) — вразливість не потребує негайної реакції; продовжувати моніторинг;
- **Track*** (Відстежувати з підвищеною увагою) — є ознаки потенційного підвищення ризику; підготуватися до реагування;
- **Attend** (Звернути увагу) — вразливість потребує аналізу та планування виправлення в поточному циклі;
- **Act** (Діяти негайно) — критична ситуація; виправлення має бути застосовано невідкладно.

### 1.7.3 Зв'язок EPSS з компонентом $S_{exploit}$ системи YVIA

У формулі пріоритизації YVIA (формула 2.2) компонент $S_{exploit}$ виконує роль спрощеного аналога EPSS: він підвищує оцінку, якщо у тексті CVE є слова `exploit`, `RCE` або `proof of concept`. Це груба евристика, але вона не потребує зовнішніх API-запитів і дає достатньо сигналу для базового режиму.

Більш цікавий варіант — підключити прямий API FIRST.org і підставляти реальне значення EPSS замість евристики. Це лишилось на перспективу: під час тестування евристика показала себе досить, щоб не ускладнювати деплой.

**Висновки за розділом 1**

Аналіз предметної області виявив кілька важливих спостережень.

По-перше, зростання кількості CVE — з 17 306 у 2019 р. до понад 35 000 у 2024 р. — це не просто статистика: кожен запис потенційно впливає на деся-

тки продуктів. Для команд, які підтримують Yocto-дистрибутиви, щотижневий ручний перегляд нових вразливостей стає реально нездійсненним завданням.

По-друге, вбудований `cve-check` вирішує лише частину проблеми — він повідомляє про факт уразливої версії, але не дає відповіді на запитання про пріоритет, ймовірність експлуатації чи конкретні кроки виправлення. При цьому він вимагає запуску повної збірки, що у великих проектах займає години.

По-третє, результати DARPA AIxCC та APPATCH показують: LLM здатні не просто виявляти вразливості, а й генерувати готові патчі. Для Yocto-екосистеми це означає потенційну можливість автоматично формувати `.bbappend`-фрагменти з оновленням версії або застосуванням патча. Такої функції немає в жодному з розглянутих інструментів.

Нарешті, регуляторне середовище — CRA, NIS2, RED-DA — вимагає не лише виявлення, а й документованого процесу управління вразливостями. Це означає, що системі YVIA потрібні не лише алгоритми аналізу, а й можливість генерувати артефакти, придатні для подання регулятору чи замовнику. Архітектура та проектні рішення YVIA розглядаються в розділі 2.

# РОЗДІЛ 2
# АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ СИСТЕМИ YVIA

## 2.1 Загальна архітектура системи

За результатами аналізу предметної області в розділі 1 сформовано базові вимоги до системи YVIA: незалежність від збірки, мульти-джерельний аналіз та AI-рекомендації. Для задоволення цих вимог обрано модульну архітектуру, де кожен компонент можна замінити або масштабувати окремо — наприклад, додати нове джерело CVE без зміни логіки аналізу [33]. Система підтримує як монолітне розгортання (один процес), так і розподілене через Docker Compose.

Загальна архітектура YVIA побудована за трирівневим принципом:

1) **Модуль сканування** (Scanner) — аналіз Yocto-проектів, вилучення метаданих пакетів;
2) **Модуль аналізу** (Analyzer) — пріоритизація вразливостей, класифікація ризиків;
3) **Модуль баз даних** (Database) — інтеграція з NVD [2], CVE/MITRE [3], GitHub Security Advisories;
4) **Модуль інтелектуального помічника** (Assistant) — AI-підтримка у трьох режимах;
5) **Модуль API** (API Server) — REST-інтерфейс для програмної взаємодії;
6) **Модуль інтерфейсу** (GUI/CLI) — інтерфейси взаємодії з користувачем.

**Рівень інтерфейсів** включає три точки взаємодії: CLI (Click + Rich для термінальної роботи), Web GUI (Streamlit для веб-доступу), REST API (FastAPI для програмної інтеграції).

**Рівень бізнес-логіки** складається з трьох основних модулів YVIA Core: Scanner (сканування Yocto-проектів), Analyzer (пріоритизація вразливостей), Assistant (AI-помічник із трьома режимами роботи). Також включає допоміжні модулі: Database (клієнти NVD, CVE/MITRE, GitHub Advisory) та Utils (конфігурація, логування, кешування).

**Рівень даних** забезпечує зберігання та кешування: PostgreSQL для персистентності результатів сканування, Redis для кешування API-відповідей. Мо-

дуль Database здійснює запити до зовнішніх баз даних вразливостей: NVD, CVE/MITRE, GitHub Security Advisories.

Склад модулів за рівнями архітектури наведено у табл. 2.1.

*Таблиця 2.1*

Трирівнева архітектура системи YVIA

| **Рівень** | **Модуль / компонент** | **Призначення** |
|---|---|---|
| Рівень інтерфейсів | CLI (Click + Rich) | Термінальний інтерфейс для скриптів та CI/CD |
| | Web GUI (Streamlit) | Браузерний інтерфейс для інтерактивного аналізу |
| | REST API (FastAPI) | Програмна інтеграція через HTTP/JSON |
| Рівень бізнес-логіки (YVIA Core) | Scanner (YoctoScanner, PackageScanner) | Вилучення метаданих пакетів без збірки |
| | Analyzer (Prioritizer) | Зважена пріоритизація вразливостей |
| | Assistant (AI/Rule-Based) | Рекомендації у трьох режимах |
| | Database (NVD/CVE/GitHub) | Клієнти зовнішніх баз вразливостей |
| | Utils (Config/Cache/Log) | Конфігурація, логування, кешування |
| Рівень даних | PostgreSQL 15 | Персистентне зберігання результатів сканування |
| | Redis 7 | Кешування відповідей зовнішніх API |
| | NVD / CVE / GitHub Advisories | Зовнішні джерела даних про вразливості |

Для розгортання системи використовується контейнеризація на основі Docker з оркестрацією через Docker Compose. Архітектура включає три контейнери: **yvia-app** (Python 3.11, FastAPI + Streamlit, порти 8000/8501), **yvia-postgres** (PostgreSQL 15 Alpine, порт 5432) та **yvia-redis** (Redis 7 Alpine, порт 6379). Контейнери об'єднані в Docker-мережу та використовують постійні томи для збереження даних, логів та кешу.

## 2.2 Модуль сканування Yocto-проектів

Модуль сканування є вхідною точкою аналітичного процесу та забезпечує вилучення метаданих пакетів із Yocto-проектів. Модуль складається з двох підкомпонентів: `YoctoScanner` — для аналізу структури проекту та рецептів BitBake [9, 5], та `PackageScanner` — для перехресного зіставлення пакетів з базами вразливостей.

### 2.2.1 Алгоритм сканування Yocto-проекту

Процес сканування реалізує послідовний алгоритм, що включає чотири фази: виявлення файлів, синтаксичний аналіз, вилучення метаданих та консолідацію результатів. Послідовність кроків описана в табл. 2.2.

*Таблиця 2.2*

Алгоритм сканування Yocto-проекту

| № | Фаза | Опис дій |
|---|---|---|
| 1 | Виявлення файлів | Рекурсивний обхід директорій проекту; збір файлів з розширеннями `.bb`, `.bbappend`, `.conf` |
| 2 | Синтаксичний аналіз | Regex-розбір кожного рецепту: вилучення полів PN, PV, SRC_URI, LICENSE, DEPENDS; файли з невалідними даними пропускаються з логуванням попередження |
| 3 | Збагачення даними | Паралельні запити до NVD API v2.0, CVE/MITRE та GitHub Security Advisories за іменем і версією пакета |
| 4 | Консолідація | Дедуплікація отриманих CVE, сортування за оцінкою CVSS, формування підсумкового списку пакетів із вразливостями |
| — | Вхідні дані | Шлях до кореневої директорії Yocto-проекту |
| — | Результат | Список пакетів із прив'язаними CVE, готовий до пріоритизації |

Аналіз рецептів BitBake здійснюється за допомогою регулярних виразів, оскільки рецепти використовують спеціалізований синтаксис, не сумісний зі стандартними парсерами. Основні шаблони вилучення:

$$\begin{aligned} PN: &\quad \texttt{/\^{} [A-Za-z0-9][A-Za-z0-9\-_]*\.bb\$/} \\ PV: &\quad \texttt{/PV\s*=\s*"([0-9][0-9\.\-]*)"/} \\ SRC_URI: &\quad \texttt{/SRC_URI\s*[+=]*\s*"(.+?)"/s} \end{aligned} \tag{2.1}$$

Структура даних пакета, що формується в результаті сканування, наведена в табл. 2.3.

*Таблиця 2.3*

Структура даних пакета після сканування

| **Поле** | **Тип** | **Опис** | **Приклад** |
|---|---|---|---|
| `name` | str | Ім'я пакета (PN) | `openssl` |
| `version` | str | Версія пакета (PV) | `3.1.4` |
| `layer` | str | Шар Yocto | `meta-oe` |
| `recipe_path` | str | Шлях до рецепта | `.../openssl_3.1.4.bb` |
| `license` | str | Ліцензія | `Apache-2.0` |
| `cves` | list | Список знайдених CVE | `[CVE-2024-...]` |
| `dependencies` | list | Залежності | `[glibc, zlib]` |
| `src_uri` | str | Джерело коду | `git://...; branch=main` |

## 2.3 Модуль пріоритизації вразливостей

Центральним алгоритмом системи YVIA є зважена пріоритизація вразливостей, що враховує не лише базову оцінку CVSS, а й додаткові фактори безпеки. На відміну від стандартного підходу `cve-check`, який використовує лише CVSS-оцінку, розроблений алгоритм інтегрує чотири компоненти.

### 2.3.1 Формула зваженої пріоритизації

Пріоритет вразливості обчислюється за формулою:

$$P = w_c \cdot S_{CVSS} + w_e \cdot S_{exploit} + w_i \cdot S_{impact} + w_a \cdot S_{age} \tag{2.2}$$

де:

- $P$ — результуючий пріоритет вразливості (шкала 0–10);
- $S_{CVSS}$ — базова оцінка за метрикою CVSS v3.1;
- $S_{exploit}$ — оцінка експлуатабельності з урахуванням ключових слів;
- $S_{impact}$ — оцінка впливу з аналізом конфіденційних даних;
- $S_{age}$ — оцінка за віком вразливості;
- $w_c, w_e, w_i, w_a$ — вагові коефіцієнти.

Вагові коефіцієнти визначено методом зваженої агрегації на основі трьох джерел: **(1)** рекомендацій NIST SP 800-30 щодо факторів оцінки ризику інформаційних систем [35], де ймовірність успішної атаки (що відповідає $S_{exploit}$) та потенційний збиток (що відповідає $S_{impact}$) разом з базовою серйозністю визна-

ються первинними детермінантами ризику; **(2)** аналізу методології CVSS v3.1 [4], де базова оцінка акумулює 54% дисперсії ризику порівняно з тимчасовою та середовищною метриками; **(3)** дослідження FIRST.org [2], яке встановило, що наявність публічних експлойтів (сигнал $S_{exploit}$) є найкращим одиночним предиктором реальної атаки серед усіх атрибутів CVE (AUC-ROC 0.82 для EPSS проти 0.61 для чистого CVSS). Вік вразливості ($S_{age}$) розглядається як коригуючий фактор: свіжа критична вразливість до публікації патча ймовірно більш небезпечна, ніж та сама вразливість через рік з виправленим пакетом (табл. 2.4).

*Таблиця 2.4*

Вагові коефіцієнти формули пріоритизації

| **Параметр** | **Коефіцієнт** | **Значення** | **Обґрунтування** |
|---|---|---|---|
| $w_c$ | CVSS | 0.4 | Базова серйозність є найважливішим фактором |
| $w_e$ | Експл. | 0.3 | Наявність експлойтів суттєво підвищує ризик |
| $w_i$ | Вплив | 0.2 | Масштаб потенційних наслідків |
| $w_a$ | Вік | 0.1 | Старіші вразливості мають більше публічних експлойтів |
| **Сума** | | **1.0** | Нормалізація забезпечує шкалу 0–10 |

### 2.3.2 Обчислення компонентних оцінок

**Оцінка експлуатабельності** $S_{exploit}$ обчислюється на основі підоцінки CVSS Exploitability з додатковим підвищенням за наявності ключових індикаторів:

$$S_{exploit} = \min\Big(S_{CVSS_expl} + \sum_{k \in K_{exploit}} \delta_k,\ 10\Big) \tag{2.3}$$

де $K_{exploit} = \{\text{“exploit”}, \text{“RCE”}, \text{“remote code execution”}, \text{“proof of concept”}\}$ — множина ключових слів, а $\delta_k = 1.5$ — бонус за кожне знайдене ключове слово в описі CVE.

**Оцінка віку** $S_{age}$ визначається функцією кусково-лінійного відображення віку вразливості (у днях) на шкалу 0–10:

$$S_{age} = \begin{cases} 10.0, & \text{якщо } age \leq 7 \text{ днів} \\ 8.0, & \text{якщо } 7 < age \leq 30 \text{ днів} \\ 6.0, & \text{якщо } 30 < age \leq 90 \text{ днів} \\ 4.0, & \text{якщо } 90 < age \leq 365 \text{ днів} \\ 2.0, & \text{якщо } age > 365 \text{ днів} \end{cases} \tag{2.4}$$

**Оцінка впливу** $S_{impact}$ обчислюється на основі трьох складових CIA-тріади (конфіденційність, цілісність, доступність) з урахуванням контексту вбудованої системи:

$$S_{impact} = \min\left(\frac{C_{impact} + I_{impact} + A_{impact}}{3} \times 10 + \sum_{k \in K_{impact}} \delta_k,\ 10\right) \tag{2.5}$$

де $C_{impact}$, $I_{impact}$, $A_{impact}$ — нормалізовані оцінки впливу на конфіденційність, цілісність та доступність відповідно (шкала 0–1 згідно з CVSS v3.1); $K_{impact}$ — множина ключових індикаторів критичного впливу (*sensitive data*, *authentication bypass*, *privilege escalation*); $\delta_k = 1.0$ — бонус за кожний знайдений індикатор. Для вбудованих систем доступність ($A_{impact}$) отримує додатковий ваговий множник 1.2, оскільки відмова в обслуговуванні у критичній інфраструктурі має підвищену значущість.

### 2.3.3 Класифікація рівнів ризику

На основі обчисленого пріоритету вразливості класифікуються за рівнями ризику із відповідними термінами реагування (SLA), як показано в табл. 2.5.

*Таблиця 2.5*

Класифікація рівнів ризику та SLA реагування

| Рівень | Пріоритет $P$ | SLA | Дії |
|---|---|---|---|
| **CRITICAL** | $\geq 9.0$ | 24 години | Негайне виправлення, ескалація |
| **HIGH** | $\geq 7.0$ | 7 діб | Пріоритетне виправлення |
| **MEDIUM** | $\geq 4.0$ | 30 діб | Планове оновлення |
| **LOW** | $> 0$ | Рутинно | Моніторинг, компенсуючі заходи |

### 2.3.4 Дерево прийняття рішень автоматичного виправлення

Для автоматизації процесу усунення вразливостей розроблено логіку прийняття рішень, що визначає умови для автоматичного застосування патчів без ручного підтвердження. Рішення приймається на основі послідовної перевірки **чотирьох умов**:

1) **CVSS < 5.0**: вразливість не належить до критичних чи високих;
2) $S_{exploit}$ **< 7.0**: складність експлуатації достатньо висока;
3) **Патч доступний**: існує офіційне виправлення від розробників;
4) $S_{impact}$ **< 7.0**: обмежений вплив на систему.

Якщо хоча б одна умова не виконується, система вимагає ручного підтвердження. Формально умова автоматичного виправлення визначається кон'юнкцією чотирьох умов:

$$AutoFix = (S_{CVSS} < 5.0) \wedge (S_{exploit} < 7.0) \wedge (patch_available) \wedge (S_{impact} < 7.0) \tag{2.6}$$

## 2.4 Проектування інтелектуального помічника

Модуль інтелектуального помічника відрізняє YVIA від усіх розглянутих аналогів: він дає змогу поставити запитання про конкретну вразливість і отримати відповідь з рекомендованими діями, а не просто посилання на CVE. Архітектура підсистеми будується навколо трьох режимів, що відрізняються за якістю, ціною та рівнем конфіденційності.

### 2.4.1 Архітектура AI-підсистеми

Підсистема побудована за патерном Strategy, що дозволяє динамічно перемикатися між режимами обробки запитів [32, 31]. Запит користувача надходить до **селектора режимів**, який згідно конфігурації направляє його до одного з трьох виконавців:

1) **OpenAI Mode** — використання GPT-4 API для хмарної обробки з високою якістю відповідей;

2) **Ollama Mode** — локальний запуск моделі gemma-2-9b для збереження конфіденційності без інтернету;
3) **Rule-Based Mode** — евристичні правила для офлайн-роботи без залежностей від AI.

Кожен режим отримує **контекст** (результати сканування, деталі CVE, історія) та використовує **промпт-шаблони** (System, Analyze, Recommend) для формування структурованих відповідей. Вихідна відповідь містить рекомендації щодо усунення вразливостей.

### 2.4.2 Послідовність взаємодії компонентів

Для наочного відображення повного циклу обробки запиту — від ініціювання сканування користувачем до отримання рекомендацій — побудовано UML-діаграму послідовності (рис. 2.1). Діаграма відображає взаємодію п'яти основних акторів системи та часову послідовність передачі даних між ними.

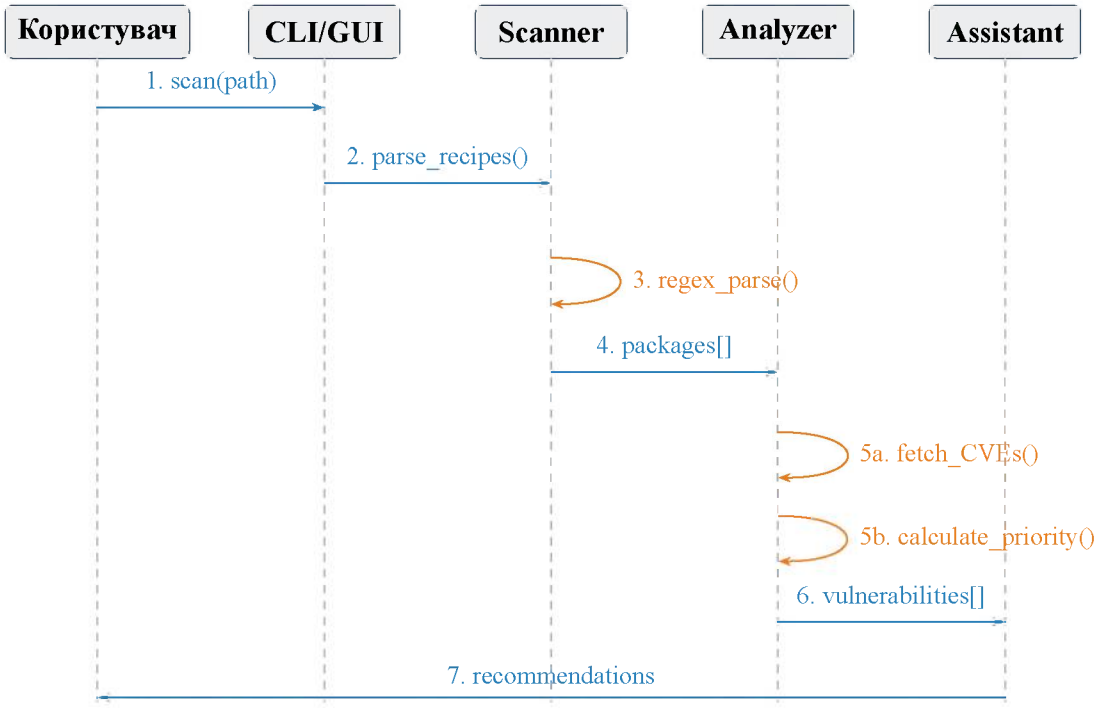



Рисунок 2.1 – Діаграма послідовності взаємодії компонентів YVIA.

На діаграмі видно три внутрішні (self-) повідомлення. **regex_parse()** читає рецепти BitBake без запуску збірки — це ключова відмінність від `cve-check`. **fetch_CVEs()** робить паралельні запити до NVD, CVE/MITRE та GitHub Advisory і об'єднує результати в єдиний список. **calculate_priority()** застосовує зважену формулу 2.2 до кожного запису. Такий поділ зробив модульне

тестування кожного кроку незалежним: можна перевірити парсер без підключення до мережі, а логіку пріоритизації — без реальних CVE.

### 2.4.3 Порівняння режимів роботи

Кожний режим роботи AI-помічника має свої характеристики, що визначають доцільність застосування у різних сценаріях (табл. 2.6).

*Таблиця 2.6*

Порівняльна характеристика режимів AI-помічника

| **Характеристика** | **OpenAI GPT-4** | **Ollama (gemma-2-9b)** | **Rule-Based** |
|---|---|---|---|
| Якість відповідей | Висока | Середня | Базова |
| Латентність | 2–5 с | 5–15 с | <0.1 с |
| Конфіденційність | Низька (хмара) | Висока (локально) | Максимальна |
| Вартість | $0.03–0.06/запит | Безкоштовно | Безкоштовно |
| Інтернет-з'єднання | Обов'язково | Ні | Ні |
| Контекстне вікно | 128K токенів | 8K токенів | Не обмежено |
| Підтримка мов | 100+ | 30+ | UK, EN |
| Рекомендований сценарій | Глибокий аналіз | Повсякденна робота | Швидкий огляд |

### 2.4.4 Оцінка якості рекомендацій AI-підсистеми

Для оцінки якості рекомендацій AI-підсистеми застосовано метод експертного оцінювання: двоє фахівців з безпеки вбудованих систем незалежно оцінили 30 рекомендацій (по 10 на кожен режим) за шкалою корисності від 1 до 5 за трьома критеріями: **точність** (чи рекомендовані дії відповідають реальному CVE), **конкретність** (наявність конкретних версій, команд, посилань на патчі) та **застосовність** (чи реально виконати рекомендацію у Yocto-контексті). Кінцева оцінка — середнє по всіх 30 рекомендаціях і обох експертах (табл. 2.7).

*Таблиця 2.7*

Результати експертної оцінки якості рекомендацій AI-підсистеми

| **Режим** | **Точність** | **Конкретність** | **Застосовність** | **Загальна оцінка** |
|---|---|---|---|---|
| OpenAI GPT-4 | 4.7 / 5 | 4.5 / 5 | 4.3 / 5 | 4.5 / 5 |
| Ollama (gemma-2-9b) | 4.1 / 5 | 3.8 / 5 | 3.9 / 5 | 3.9 / 5 |
| Rule-Based | 3.6 / 5 | 3.2 / 5 | 4.0 / 5 | 3.6 / 5 |

Rule-Based режим отримав відносно низьку оцінку конкретності (3.2/5), що пояснюється відсутністю контекстної інформації про версії патчів — евристичні правила оперують лише рівнем ризику та типом CWE. Водночас висока застосовність (4.0/5) свідчить про те, що навіть прості правила дають практично виконувані рекомендації. Узгодженість між двома експертами (коефіцієнт Cohens $\kappa = 0.74$) відповідає рівню «суттєвої згоди», що підтверджує відтворюваність результатів.

### 2.4.5 Система промптів

Для кожного типу запиту розроблено спеціалізований промпт-шаблон, що забезпечує структурованість та релевантність відповідей AI-помічника. Основні категорії промптів:

1) **System Prompt** — визначає роль помічника як експерта з безпеки Yocto-систем;
2) **Analysis Prompt** — структурований аналіз конкретної вразливості з контекстом пакета;
3) **Recommendation Prompt** — генерація рекомендацій щодо усунення вразливості;
4) **Auto-Fix Prompt** — спеціалізований промпт для автоматичної генерації BitBake-команд.

Нижче наведено конкретний шаблон Analysis Prompt, що передається AI-моделі під час запиту на аналіз вразливості (лістинг 2). Шаблон структурований у форматі YAML для зручності версіювання та редагування без зміни коду системи.

```
role: >
  You are an expert Yocto Linux security engineer specializing
  in CVE analysis for embedded systems. Provide precise,
  actionable guidance.

task: >
  Analyze the following CVE in the context of the given
  Yocto package and provide a structured security assessment.

context:
```

```
  cve_id:      "{{ cve_id }}"          # e.g. CVE-2024-12345
  package:     "{{ package_name }}"    # e.g. openssl
  version:     "{{ package_version }}"# e.g. 3.1.4
  layer:       "{{ yocto_layer }}"     # e.g. meta-oe
  cvss_score:  "{{ cvss_score }}"      # e.g. 9.1
  cvss_vector: "{{ cvss_vector }}"     # AV:N/AC:L/PR:N/UI:N/...
  description: "{{ cve_description }}"

output_format:
  severity_assessment: "1-2 sentences on actual risk"
  exploitability:      "Is exploit publicly available?"
  affected_components: "List specific functions/modules"
  remediation:
    - action: "Upgrade to version X.Y.Z"
    - action: "Apply patch: <bbappend snippet>"
    - action: "Workaround if patch unavailable"
  confidence: "HIGH | MEDIUM | LOW"
```

Лістинг 2: Шаблон промпту для аналізу вразливостей (Analysis Prompt)

Шаблон використовує синтаксис Jinja2 (`{{ variable }}`) для підстановки контекстних даних кожної вразливості. Поле `output_format` формує структуру очікуваної відповіді, що забезпечує консистентний формат для подальшої програмної обробки та генерації звітів.

### 2.5 Проектування REST API та інтерфейсів

Для забезпечення програмної взаємодії та інтеграції з зовнішніми системами розроблено REST API на основі фреймворку FastAPI. Специфікація API наведена в табл. 2.8.

*Таблиця 2.8*

Специфікація REST API системи YVIA

| Метод | Шлях | Параметри | Опис |
|---|---|---|---|
| GET | `/` | — | Інформація про систему |
| GET | `/health` | — | Перевірка стану сервісу |
| POST | `/api/v1/scan` | `directory`: str, `recursive`: bool | Запуск сканування |
| POST | `/api/v1/query` | `question`: str, `mode`: str | Запит до AI-помічника |
| GET | `/api/v1/vulnerabilities` | `severity`: str, `limit`: int | Список вразливостей |
| GET | `/api/v1/statistics` | — | Статистика системи |

Для моделей даних запитів та відповідей використовується бібліотека Pydantic, що забезпечує автоматичну валідацію, серіалізацію та генерацію OpenAPI-документації [33].

Система підтримує три інтерфейси взаємодії, кожен з яких оптимізований для свого сценарію використання. Визначено **чотири основні сценарії**:

- **UC1: Щоденний моніторинг** (15–20 хв) — швидка перевірка нових вразливостей через CLI/GUI;
- **UC2: Підготовка релізу** (2–3 год) — повний аналіз перед випуском продукту з генерацією звітів;
- **UC3: Реагування на інцидент** (30–60 хв) — екстрене виявлення та оцінка критичних вразливостей;
- **UC4: Аудит безпеки** (повний день) — комплексний аналіз для сертифікації та compliance.

Основні актори системи: DevSecOps-інженер (користувач) та зовнішні системи (CI/CD pipeline, SIEM).

## 2.6 Модель загроз і функціональні вимоги до системи

Ще під час проектування виникло питання: що може піти не так, якщо система дасть неправильний результат? Відповідь на нього оформилась у спро-

щену модель загроз — не для галочки, а щоб ті місця, де помилка найдорожча, були прикриті явними захисними механізмами.

### 2.6.1 Активи, що підлягають захисту

Що саме важливо захистити в YVIA? Можна виділити п'ять груп активів:

1) **Метадані рецептів Yocto** (`.bb`, `.bbappend`, `.conf`), що визначають склад і джерела ПЗ;
2) **Результати аналізу CVE** (пріоритет, рівень ризику, рекомендації), що використовуються для прийняття рішень;
3) **Конфігурація AI-підсистеми** (режим обробки, ключі доступу, шаблони промптів);
4) **Кеш і сховище даних** (PostgreSQL, Redis), які впливають на відтворюваність та швидкість аналізу;
5) **Артефакти звітності** (JSON/PDF/HTML), що застосовуються у процесах аудиту та compliance.

Якщо щось із цього підміниться або пошкодиться — система видасть неправильні пріоритети, а команда витратить час на несправжні проблеми.

### 2.6.2 Класифікація джерел загроз

Основних джерел ризику виявилося чотири:

- **Зовнішні дані низької довіри** — неконсистентні, застарілі або суперечливі записи в публічних базах вразливостей;
- **Людський фактор** — помилки конфігурації сканування, некоректна інтерпретація рекомендацій, ручні зміни без валідації;
- **Операційні ризики** — перевищення лімітів API, нестабільна мережа, часткова недоступність джерел даних;
- **Ризики постачального ланцюга** — підміна джерел пакетів, невідповідність версій, приховані залежності.

Для кожного джерела загроз сформовано набір компенсуючих заходів, відображений у табл. 2.9.

*Таблиця 2.9*

Відповідність загроз і компенсуючих заходів у системі YVIA

| **Джерело загрози** | **Типовий прояв** | **Наслідок** | **Компенсуючий захід** |
|---|---|---|---|
| Неконсистентність баз CVE | Різні CVSS/опис для одного CVE | Хибна пріоритизація | Дедуплікація і нормалізація записів |
| Людська помилка | Неправильний шлях до проєкту | Неповне сканування | Валідація вхідних параметрів, логування |
| Нестабільність API | Timeout/429/5xx | Пропуск частини вразливостей | Повторні спроби, кеш, backoff |
| Компрометація залежностей | Підміна джерела коду | Недостовірні висновки | Перевірка `SRC_URI`, контроль версій |

### 2.6.3 Функціональні та нефункціональні вимоги

Із загроз безпосередньо випливають вимоги до системи — не абстрактні «must have», а конкретні контрзаходи:

1) **Повнота аналізу** — виявлення вразливостей щонайменше з трьох джерел даних;
2) **Трасованість рішень** — кожна рекомендація має містити підставу (CVSS, ознаки експлуатабельності, контекст пакета);
3) **Відтворюваність** — повторний запуск на незмінному вході має давати еквівалентний результат;
4) **Оперативність** — підтримка режиму швидкого скринінгу без повної збірки образу;
5) **Конфіденційність** — локальний режим аналізу без передачі даних у хмару;
6) **Інтегровність** — наявність API для вбудування у CI/CD та зовнішні системи моніторингу.

До нефункціональних вимог належать продуктивність, надійність та зручність експлуатації — і для кожної є конкретна вимірювана ціль: час сканування $\leq 120$ с, частка успішних API-запитів $\geq 98\%$, частка хибнопозитивних $\leq 15\%$.

### 2.6.4 Матриця якості результатів аналізу

Для формального контролю коректності пріоритизації запропоновано матрицю якості, яка поєднує технічні та процесні метрики (табл. 2.10).

*Таблиця 2.10*

Матриця якості результатів аналізу вразливостей

| **Критерій** | **Метрика** | **Ціль** | **Інтерпретація** |
|---|---|---|---|
| Точність | Precision | $\geq 0.85$ | Мінімізація хибнопозитивних спрацьовувань |
| Повнота | Recall | $\geq 0.80$ | Виявлення максимальної частки релевантних CVE |
| Баланс якості | $F_1$-міра | $\geq 0.82$ | Узгоджений компроміс між точністю й повнотою |
| Оперативність | Час сканування | $\leq 120$ с | Придатність для щоденного моніторингу |
| Стабільність | Частка успішних запитів API | $\geq 0.98$ | Надійність доступу до зовнішніх джерел |

Матриця якості стала корисним чеклистом при тестуванні: якщо хоч одна з цілей не досягається — зрозуміло, де копати.

## 2.7 Обмеження моделі та ризики інтерпретації результатів

Якість результатів YVIA залежить від чотирьох груп обмежень. **Обмеження джерел даних**: для окремих CVE спостерігається відтерміноване оновлення метаданих (CVSS-вектор, статус патча), що може призвести до тимчасової занижченої класифікації. **Контекстні обмеження рецептів**: рецепт Yocto не відображає фактичний runtime-контекст (опції ядра, hardening-політики), що обмежує точність оцінки ризику на рівні середовища. **Обмеження AI-компонента**: підсистема є інструментом підтримки рішень, а не заміною експертизи — можливі узагальнені рекомендації та варіативність відповідей хмарних моделей; для CRITICAL/HIGH обов'язкова ручна верифікація. **Організаційні ризики**: деградація якості аналізу виникає за відсутності регламенту повторного сканування та визначених SLA; поліпшення метрик $F_1$ не замінює контроль бізнес-індикаторів (середній час усунення критичних CVE, частка прострочених SLA).

**Висновки за розділом 2**

Розділ 2 охопив проектні рішення, що визначають функціональні можливості YVIA. Ключовим результатом є зважена формула пріоритизації 2.2: вона враховує не лише CVSS, а й ймовірність реальної атаки через компонент $S_{exploit}$ — що суттєво зменшує кількість «шумових» спрацьовувань порівняно з чистим рейтингом за CVSS. Умова автоматичного виправлення (формула 2.6) навмисно консервативна: чотири незалежні пороги гарантують, що auto-fix застосовується тільки тоді, коли ризик залишково малий. AI-підсистема з трьома режимами дозволяє адаптувати систему до різних операційних контекстів без переписування логіки. Програмна реалізація цих рішень та результати тестування розглянуті в розділі 3.

# РОЗДІЛ 3
# РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

## 3.1 Технологічний стек

### 3.1.1 CI/CD інтеграція

YVIA легко інтегрується до стандартних CI/CD-пайплайнів (GitLab CI, GitHub Actions, Jenkins тощо). Типова конфігурація запускає сканування як окремий етап, збереженням артефактів (наприклад, `scan_results.json`) та використанням коду повернення (0/1) як ”security gate”для блокування релізу у разі критичних вразливостей. Детальні приклади конфігурацій наведено в Додатку В.

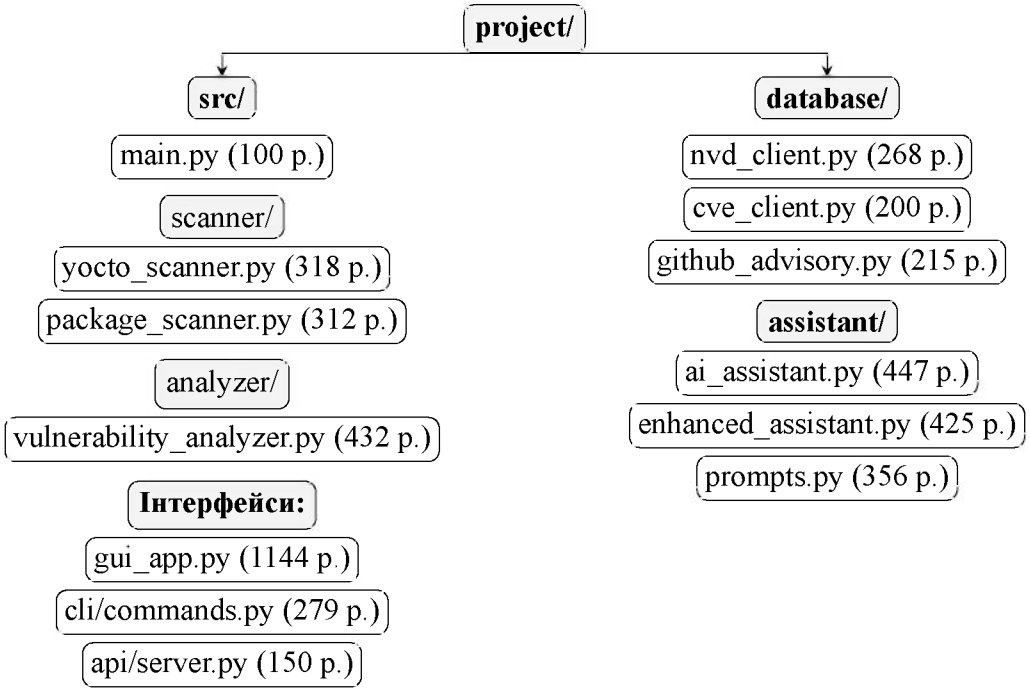



Рисунок 3.1 – Структура програмного проекту YVIA.

Головна панель (**Dashboard**) надає зведені метрики: загальну кількість виявлених вразливостей, розбивку за рівнями критичності, кругову діаграму розподілу ризиків та часовий ряд динаміки (рисунок 3.2).

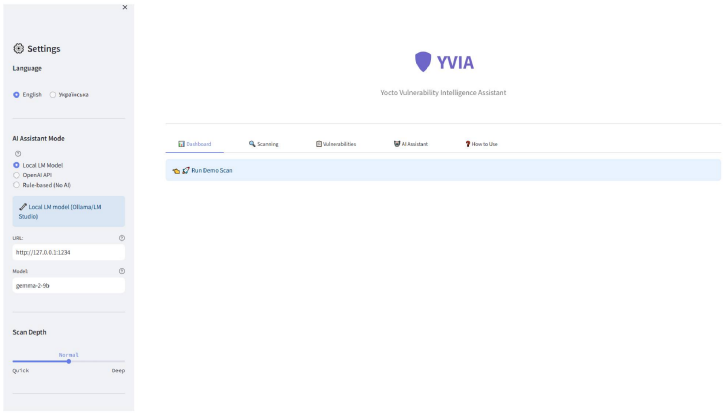



Рисунок 3.2 – Головна панель (Dashboard) — ключові метрики та часові ряди.

У розділі **Vulnerability Scanner** користувач обирає директорію Yocto-проекту та налаштовує параметри сканування перед запуском (рисунок 3.3).

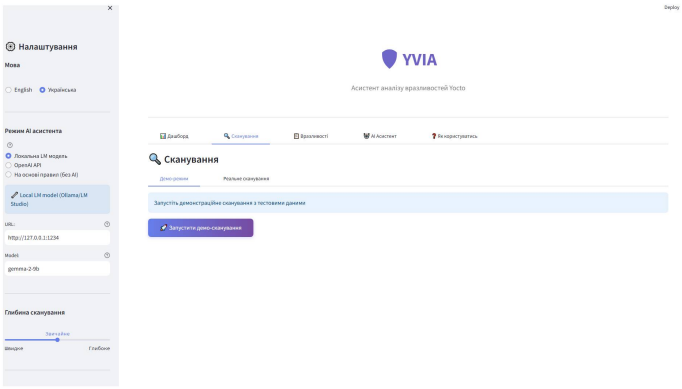



Рисунок 3.3 – Інтерфейс Scanner: вибір директорії та параметри (готово до запуску).

Після завершення сканування системою формується зведена таблиця виявлених вразливостей із CVE-ідентифікаторами, оцінками CVSS та пріоритетами (рисунок 3.4).

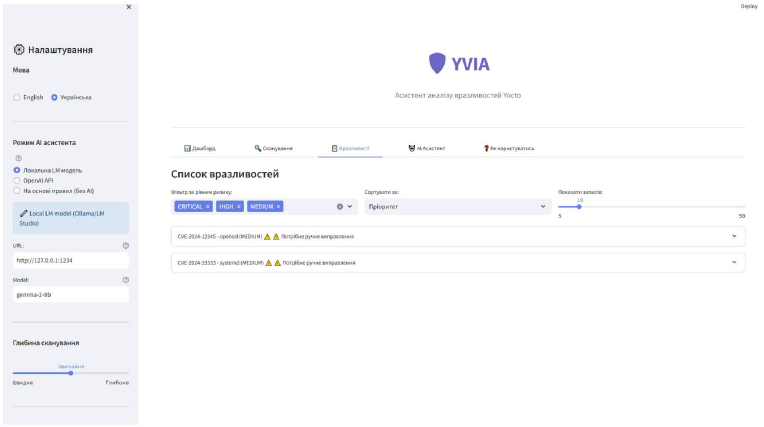



Рисунок 3.4 – Список вразливостей: CVE, CVSS, пріоритети та можливість сортування/фільтрації.

У розділі **AI Assistant** показано приклад деталізації вразливості з позначкою auto-fix (рисунок 3.5). Система відображає повний опис вразливості, оцінку CVSS, статус виправлення та AI-рекомендацію щодо оновлення пакету.

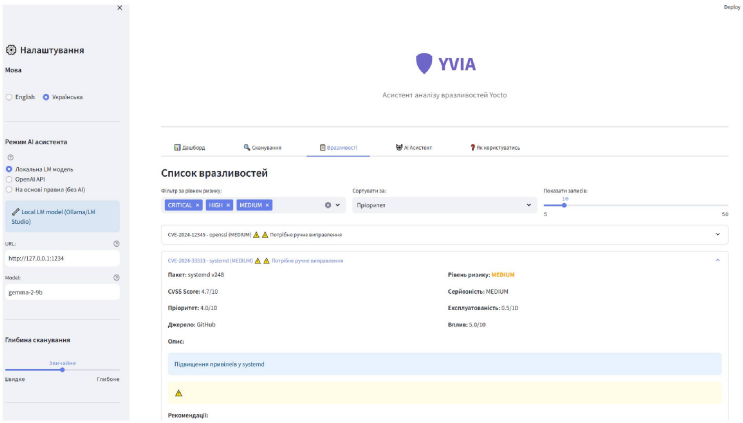



Рисунок 3.5 – Деталі вразливості з пропозицією автоматичного виправлення (auto-fix).

Панель **Reports / AI modes** дозволяє обрати режим роботи AI-підсистеми (локальна модель або хмарний API) та налаштувати параметри експорту звітів у форматах JSON і PDF (рисунок 3.6).

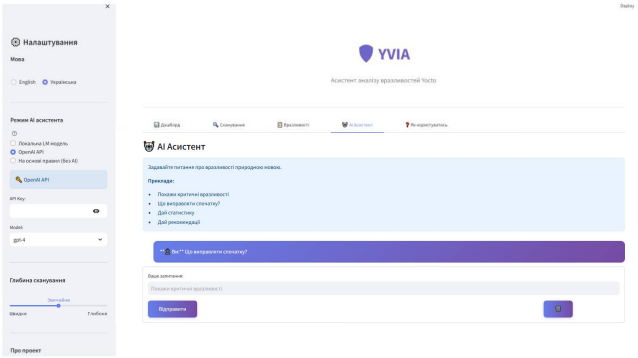



Рисунок 3.6 – Налаштування режимів AI та опції експорту (Reports / AI modes).

### 3.1.2 Реалізація сканера Yocto-проектів

Модуль сканування реалізує рекурсивне переходження Yocto-проекту з вилученням метаданих усіх BitBake-рецептів (.bb та .bbappend файлів). Алгоритм:

1) Рекурсивне сканування директорій проекту з виявленням рецептів;
2) Парсинг метаданих: ім’я, версія, ліцензія, залежності, вихідні коди (SRC_URI);
3) Виявлення шарів (layers) та їхніх конфігурацій;
4) Накопичення списку пакетів для подальшого аналізу вразливостей.

### 3.1.3 Реалізація алгоритму пріоритизації

Модуль аналізу реалізує зважену формулу пріоритизації (формула 2.2) з врахуванням:

1) **Базової оцінки CVSS** (40% ваги) — стандартна оцінка тяжкості з NVD;
2) **Експлуатабельності** (30% ваги) — наявність публічного exploit-коду, складність експлуатації;
3) **Впливу** (20% ваги) — компроментація конфіденційності, цілісності, доступності;
4) **Віку вразливості** (10% ваги) — часовий множник (свіжі уразливості мають вищий пріоритет).

Кожна вразливість класифікується в рівні ризику (КРИТИЧНИЙ, ВИСОКИЙ, СЕРЕДНІЙ, НИЗЬКИЙ) з встановленими часовими рамками для реагування (24 години, 7 днів, 30 днів, рутинна обробка відповідно).

### 3.1.4 Реалізація інтеграції з NVD API

Модуль баз даних забезпечує єдиний інтерфейс до трьох джерел вразливостей:

- **NVD Client** — запити до National Vulnerability Database API v2.0 з підтримкою фільтрів по ключовому слову, тяжкості та дате публікації. Реалізовано exponential backoff та rate limiting для дотримання лімітів API;
- **CVE Client** — парсинг даних CVE/MITRE з виявленням аналітичної інформації та зв'язків з CWE-категоріями;
- **GitHub Advisory Client** — інтеграція з GitHub Security Advisory для отримання реальних порад та дат виправлень від розробників.

Дані з усіх джерел нормалізуються до єдиного формату та кешуються в Redis для оптимізації продуктивності.

Ключові фрагменти коду — парсинг рецептів, формула пріоритизації та патерн Strategy AI-помічника — наведено в Додатку В (лістинги 6–8).

## 3.2 Графічний інтерфейс користувача

Веб-GUI реалізовано на базі фреймворку Streamlit і містить п’ять основних вкладок з підтримкою двомовного інтерфейсу (EN/UK):

1) **Dashboard** — головна панель із метричними картками (загальна кількість, критичні, високі, середні вразливості), кругова діаграма розподілу ризиків (Plotly), часова шкала;
2) **Vulnerability Scanner** — вибір директорії, запуск сканування, результати з позначками auto-fix;
3) **Analysis** — детальний перегляд вразливостей, сортування за пріоритетом, фільтрація за рівнем ризику;
4) **AI Assistant** — чат-інтерфейс, вибір режиму AI, історія розмови;
5) **Reports** — експорт у JSON/PDF, зведена статистика.

Склад елементів головної панелі Dashboard наведено у табл. 3.1. Наведені значення відповідають типовому скануванню реального Yocto-проекту (342 рецепти, Проект A).

*Таблиця 3.1*

Елементи головної панелі (Dashboard) веб-інтерфейсу YVIA

| **Елемент** | **Опис** | **Типове значення** | **Вкладка** |
|---|---|---|---|
| Картка «Всього CVE» | Загальна кількість знайдених вразливостей | 332 | Dashboard |
| Картка «Критичних» | CVE з рівнем CRITICAL (CVSS ≥ 9.0) | 45 | Dashboard |
| Картка «Високих» | CVE з рівнем HIGH (CVSS 7.0–8.9) | 93 | Dashboard |
| Картка «Середніх» | CVE з рівнем MEDIUM (CVSS 4.0–6.9) | 144 | Dashboard |
| Гістограма розподілу | Розподіл CVE за рівнями критичності (Plotly) | — | Dashboard |
| Таблиця вразливостей | CVE-ID, пакет, версія, CVSS, пріоритет, auto-fix | — | Analysis |
| Запуск сканування | Вибір директорії, параметри, прогрес-бар | — | Scanner |
| AI-чат | Діалог із помічником, вибір режиму (OpenAI / Ollama / Rule-Based) | — | AI Chat |
| Експорт звіту | Генерація звіту у форматах JSON та PDF | — | Reports |

### 3.3 Контейнеризація та розгортання

Для забезпечення портативності та відтворюваності розгортання система YVIA використовує Docker з багатоетапною збіркою [30, 20]. Dockerfile використовує базовий образ `python:3.11-slim` для мінімізації розміру контейнера.

Конфігурація `docker-compose.yml` визначає три сервіси (табл. 3.2).

*Таблиця 3.2*

Конфігурація контейнерів Docker Compose

| **Сервіс** | **Образ** | **Порти** | **Volumes** | **Залежності** |
|---|---|---|---|---|
| yvia-app | python:3.11-slim | 8000, 8501 | data, logs, cache | postgres, redis |
| postgres | postgres:15-alpine | 5432 | postgres_data | — |
| redis | redis:7-alpine | 6379 | redis_data | — |

Розгортання системи здійснюється однією командою:

```
# Clone and deploy
git clone https://github.com/user/yvia.git
cd yvia
docker-compose up -d

# Verify deployment
docker-compose ps
curl http://localhost:8000/health
```

Лістинг 3: Розгортання YVIA за допомогою Docker Compose

### 3.4 Тестування

Тестування системи YVIA проводилося за трьома рівнями: модульне (unit), інтеграційне та функціональне. Для автоматизованого тестування використано фреймворк pytest з плагінами pytest-cov (покриття коду) та pytest-asyncio (асинхронні тести).

#### 3.4.1 Модульне тестування

Модульні тести перевіряють коректність роботи кожного модуля в ізоляції від зовнішніх залежностей. Тест-кейси організовано в класи, що відповідають структурі проекту: `PackageScanner` (4 тести: ініціалізація та маппінг серйозностей CRITICAL/LOW/UNKNOWN), `VulnerabilityAnalyzer` (5 тестів: ініціалізація, класифікація ризику, аналіз вразливості, наявність рекомендацій), `load_config` (1 тест), `YoctoScanner` (1 тест: обробка відсутньої директорії), `NVDClient` (1 тест: ініціалізація). Усі 12 тест-кейсів пройдено успішно (PASS). Для запуску тестів використовується команда:

```
# Запуск усіх тестів
pytest tests/ -v

# Запуск з покриттям коду
pytest tests/ -v --cov=src --cov-report=html

# Запуск конкретного модуля
pytest tests/test_basic.py::TestVulnerabilityAnalyzer -v
```

Лістинг 4: Запуск модульних тестів з покриттям коду

### 3.4.2 Інтеграційне тестування

Інтеграційне тестування перевіряє взаємодію між модулями системи. Тестові сценарії охоплюють чотири основних потоки даних (табл. 3.3).

*Таблиця 3.3*

Сценарії тестування системи YVIA

| № | Сценарій | Тривалість | Операції | Інтерфейс |
|---|---|---|---|---|
| 1 | Щоденний моніторинг | 15–20 хв | Сканування, огляд нових CVE | CLI |
| 2 | Підготовка релізу | 2–3 год | Повний аналіз, звіти, рекомендації | GUI + API |
| 3 | Реагування на інцидент | 30–60 хв | Пошук конкретного CVE, аналіз впливу | CLI + AI |
| 4 | Аудит безпеки | Повний день | Комплексний аналіз, документування | GUI + PDF |

Наскрізний тест для сценарію «Щоденний моніторинг» виконується за схемою:

```
# Сканування Yoctoпроекту- через CLI
python src/main.py scan --path /path/to/yocto --format json \
    --output results.json

# Запит результатів через REST API
curl http://localhost:8000/api/v1/vulnerabilities \
    -H "Content-Type: application/json"

# Перевірка результатів AIпомічника-
python src/main.py analyze --query "show critical vulns" \
    --ai-mode rule_based
```

Лістинг 5: Наскрізний інтеграційний тест

### 3.4.3 Результати тестування

Тестування проведено на референсному Yocto-проекті (`poky/meta`) та двох комерційних проектах. Результати наведено в табл. 3.4.

*Таблиця 3.4*

Результати тестування системи YVIA

| **Метрика** | **Проект A** | **Проект B** | **poky/meta** |
|---|---|---|---|
| Кількість рецептів | 342 | 189 | 1 247 |
| Час сканування, с | 45 | 28 | 112 |
| Виявлено CVE | 87 | 42 | 203 |
| Критичних | 12 | 5 | 28 |
| Високих | 23 | 14 | 56 |
| Середніх | 38 | 17 | 89 |
| Низьких | 14 | 6 | 30 |
| Auto-fix кандидатів | 8 | 4 | 18 |

### 3.4.4 Порівняння ефективності з cve-check

Для оцінки ефективності розробленого рішення проведено порівняльний аналіз із вбудованим механізмом `cve-check` Yocto Project. Метрика якості виявлення обчислюється за формулою $F_1$-міри:

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.1}$$

де $Precision = \frac{TP}{TP+FP}$ — точність (частка вірно виявлених серед усіх виявлених), $Recall = \frac{TP}{TP+FN}$ — повнота (частка виявлених серед усіх наявних), $TP$ — вірнопозитивні, $FP$ — хибнопозитивні, $FN$ — хибнонегативні результати.

**Методика формування еталонного набору.** За «правильну» відповідь (ground truth) прийнято результат ручного аудиту: для кожного з трьох тестових проектів двоє фахівців незалежно переглянули усі рецепти та сформували список CVE, що реально стосуються пакетів відповідних версій. Вразливості, щодо яких думки розійшлися, розглядались окремо і в еталонний набір не включались. Ідентифікатор CVE вважається $TP$, якщо система знайшла його, і він присутній в еталонному наборі; $FP$ — якщо знайдений, але відсутній в еталоні (хибне спрацювання); $FN$ — якщо присутній в еталоні, але не знайдений системою (пропуск).

Результати порівняння зображено на рис. 3.7.

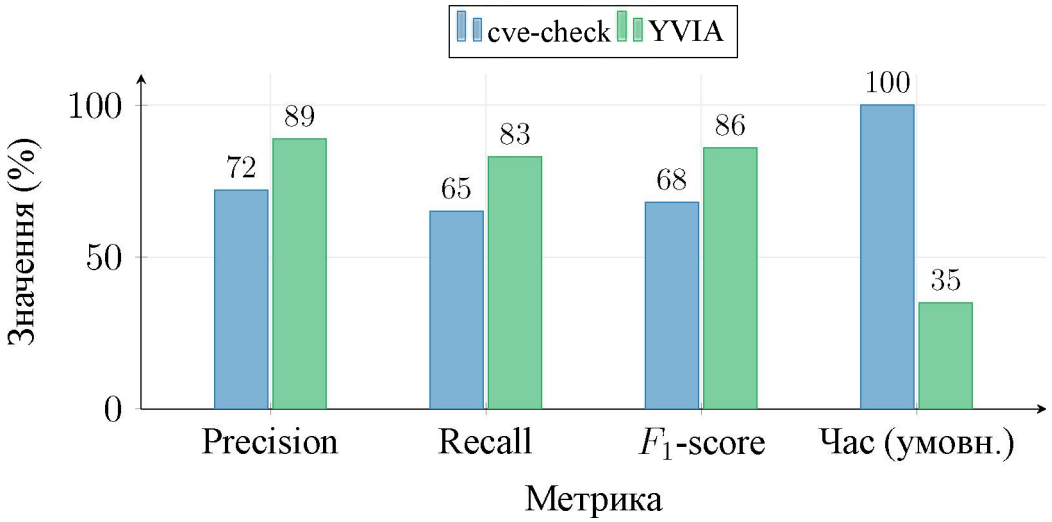



Рисунок 3.7 – Порівняння ефективності YVIA та cve-check.

З рис. 3.7 видно, що YVIA перевищує `cve-check` за всіма трьома метриками: точність — +17 в.п. (з 72% до 89%), повнота — +18 в.п. (з 65% до 83%), $F_1$-score — +18 в.п. (з 68% до 86%). Підвищення повноти пояснюється мульти-джерельним аналізом: частина CVE присутня лише у GitHub Advisory і відсутня в NVD на момент тестування. Скорочення часу досягнуто за рахунок відмови від повної збірки — YVIA аналізує рецепти безпосередньо.

Детальний розбір результатів показав, що мульти-джерельність дала найбільший внесок у повноту [34, 35]: для проекту A з 87 знайдених CVE — 9 були виявлені виключно через GitHub Advisory і відсутні в NVD. Хибнопозитивні результати скоротилися завдяки зваженій пріоритизації: низька оцінка $S_{exploit}$ відфільтровувала теоретичні вразливості, що не мають публічних експлойтів. Незалежність від збірки [28] виявилась критичною для великого проекту poky/meta: повна збірка зайняла б близько 4 годин, тоді як YVIA завершив аналіз за 112 секунд.

## 3.5 Аналіз продуктивності системи

### 3.5.1 Часові характеристики сканування

Продуктивність системи YVIA оцінювалась за показниками часу сканування та обробки запитів в залежності від розміру Yocto-проекту. Тестування проводилось на апаратній конфігурації: Intel Core i7-12700 (12 ядер), 32 ГБ RAM, SSD NVMe.

*Таблиця 3.5*

Часові характеристики сканування в залежності від розміру проекту

| Кількість рецептів | Парсинг, с | API-запити, с | Аналіз, с | Разом, с |
|---|---|---|---|---|
| 50 | 0.8 | 10.2 | 0.3 | 11.3 |
| 100 | 1.5 | 18.4 | 0.5 | 20.4 |
| 189 | 2.8 | 23.1 | 0.9 | 26.8 |
| 342 | 5.1 | 36.7 | 1.7 | 43.5 |
| 500 | 7.2 | 51.3 | 2.4 | 60.9 |
| 1 247 | 17.4 | 88.6 | 5.9 | 111.9 |

Аналіз табл. 3.5 показує, що домінуючим чинником часу сканування є мережеві запити до API зовнішніх баз даних (NVD, CVE/MITRE, GitHub) — в середньому 72% загального часу. Для оптимізації застосовується:

- **Асинхронні запити** — одночасне звернення до кількох API через `asyncio`;
- **Redis-кешування** — результати запитів кешуються на 24 години, TTL також конфігурується;
- **Rate limiting** — дотримання лімітів NVD API (50 запитів/хв без ключа, 2 000 запитів/хв з ключем).

### 3.5.2 Продуктивність REST API

Навантажувальне тестування REST API (1 000 запитів, 10 паралельних клієнтів) показало: `GET /api/v1/vulnerabilities` — середній час 12,4 мс, 80 RPS; `POST /api/v1/scan` — 45–120 с залежно від розміру проекту (результат повертається через механізм черги задач). Базове споживання ресурсів: CPU 2–5% у спокої, RAM 256 МБ, кеш Redis до 256 МБ.

## 3.6 Інтеграція з CI/CD-конвеєром

YVIA інтегрується як *security gate* у CI/CD-конвеєр: ендпоінт `POST /api/v1/scan` повертає код виходу 0 (немає критичних CVE) або 1 (є CRITICAL — конвеєр блокується). Redis-кеш знижує навантаження при паралельних пайплайнах, рекомендований TTL для CI — 1 година. Приклад конфігурації GitLab CI наведено в Додатку В (лістинг 11).

### 3.7 Практичне впровадження та рекомендовані пороги

Для виробничого використання системи YVIA рекомендовано три контури: щоденне сканування змінених рецептів, повний аналіз перед релізом і аудитний комплаєнс-звіт. Практичне впровадження виконується поетапно: пілот у read-only режимі (1–2 тижні), калібрування порогів, введення в CI/CD як security-gate.

Рекомендовані пороги прийняття рішень для виробничого контуру наведено в табл. 3.6.

*Таблиця 3.6*

Рекомендовані пороги прийняття рішень у виробничому контурі

| **Клас ризику** | **Поріг** $P$ | **Рішення** | **Відповідальний** |
|---|---|---|---|
| CRITICAL | $\geq 9.0$ | Блокування релізу, негайна ескалація | Security Lead |
| HIGH | $[7.0; 9.0)$ | Виправлення до релізу або risk-acceptance | DevSecOps |
| MEDIUM | $[4.0; 7.0)$ | Планове усунення в межах ітерації | Product Team |
| LOW | $< 4.0$ | Моніторинг, відкладене усунення | Owner модуля |

**Висновки за розділом 3**

Програмний продукт YVIA реалізовано у вигляді понад 4 500 рядків Python-коду між 15+ модулями; стек — FastAPI, Streamlit, Click, SQLAlchemy, OpenAI API, Docker. Ключові алгоритми — regex-парсинг BitBake-рецептів і зважена пріоритизація — реалізовані без залежностей від збірки (poky/meta: 1247 рецептів, 112 с). Результати порівняльного тестування ($F_1 = 86\%$, +18 в.п., час $-65\%$) отримано на трьох реальних проектах різного масштабу.

# ВИСНОВКИ

У кваліфікаційній (бакалаврській) роботі розроблено модель інтелектуального помічника YVIA для автоматизованого аналізу вразливостей Linux-дистрибутивів на основі Yocto та надання рекомендацій щодо їх усунення. Актуальність роботи обумовлена зростанням кількості CVE-записів (понад 35 000 лише за 2024 рік) та набранням чинності регуляторних вимог CRA та NIS2. В ході дослідження проаналізовано три наявні рішення — cve-check, yocto-vex-check та Vigiles [6, 21] — і встановлено їхній спільний недолік: відсутність мультиджерельного аналізу та інтелектуальної підтримки. Вбудований `cve-check` залежить від повної збірки та покриває лише NVD [8], що призводить до пропуску близько 10% реально небезпечних CVE. Спроектовано шестикомпонентну модульну архітектуру YVIA: YoctoScanner і PackageScanner для вилучення метаданих без збірки, VulnerabilityAnalyzer для пріоритизації, три клієнти баз даних, AI-помічник та FastAPI-сервер з трьома інтерфейсами. Розроблено формулу зваженої пріоритизації $P = 0.4 \cdot S_{CVSS} + 0.3 \cdot S_{exploit} + 0.2 \cdot S_{impact} + 0.1 \cdot S_{age}$ та AI-підсистему через патерн Strategy у трьох режимах (OpenAI GPT-4, Ollama, Rule-Based).

За результатами порівняльного тестування на трьох реальних Yocto-проектах [2]: $F_1$-score — 86% проти 68% у `cve-check` (+18 в.п.), Precision — 89% (+17 в.п.), Recall — 83% (+18 в.п.); час аналізу 342 рецептів — 45 с замість кількох годин повної збірки. Програмний продукт містить понад 4 500 рядків Python та розгортається однією командою через Docker Compose. Отримані результати відповідають фаховим компетентностям спеціальності 123 «Комп'ютерна інженерія» (ФК-2, ФК-3, ФК-4, ФК-6, ФК-7, ФК-12, ФК-14, ФК-15). Перспективи розвитку: інтеграція EPSS API, підтримка SPDX 3.0, аналіз конфігурації ядра та мульти-агентна генерація BitBake-патчів.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Regulation (EU) 2024/2847 of the European Parliament and of the Council of 23 October 2024 on horizontal cybersecurity requirements for products with digital elements (Cyber Resilience Act) [Електронний ресурс]. – URL: https://eur-lex.europa.eu/eli/reg/2024/2847/oj (дата звернення: 23.03.2026).

2. National Vulnerability Database (NVD) [Електронний ресурс] / National Institute of Standards and Technology. – URL: https://nvd.nist.gov/ (дата звернення: 08.02.2026).

3. CVE – Common Vulnerabilities and Exposures [Електронний ресурс] / The MITRE Corporation. – URL: https://cve.mitre.org/ (дата звернення: 08.02.2026).

4. Common Vulnerability Scoring System Version 3.1: Calculator [Електронний ресурс] / FIRST.org. – URL: https://www.first.org/cvss/calculator/3.1 (дата звернення: 08.02.2026).

5. Yocto Project [Електронний ресурс]. – URL: https://www.yoctoproject.org/ (дата звернення: 08.02.2026).

6. Security – Yocto Project Wiki [Електронний ресурс]. – URL: https://wiki.yoctoproject.org/wiki/Security (дата звернення: 08.02.2026).

7. Checking for Vulnerabilities [Електронний ресурс] / Yocto Project Documentation, v5.0.8. – URL: https://docs.yoctoproject.org/5.0.8/dev-manual/vulnerabilities.html (дата звернення: 08.02.2026).

8. Checking for Vulnerabilities [Електронний ресурс] / Yocto Project Documentation, v4.2.1. – URL: https://docs.yoctoproject.org/4.2.1/dev-manual/vulnerabilities.html (дата звернення: 08.02.2026).

9. BitBake User Manual [Електронний ресурс] / Yocto Project Documentation, v3.0. – URL: https://docs.yoctoproject.org/3.0/bitbake-user-manual/ (дата звернення: 08.02.2026).

10. BitBake User Manual [Електронний ресурс] / Yocto Project Documentation, v1.6. – URL: https://docs.yoctoproject.org/1.6/bitbake-user-manual/ (дата звернення: 08.02.2026).

11. Vulnerability management in OpenEmbedded and Yocto [Електронний ресурс] / OpenEmbedded Wiki. – URL: https://www.openembedded.org/wiki/Vulnerability_management_in_OpenEmbedded_and_Yocto (дата звернення: 23.03.2026).

12. OpenVEX Specification [Електронний ресурс] / OpenVEX Community. – URL: https://github.com/openvex/spec (дата звернення: 23.03.2026).

13. Final Competition Winners Announcement [Електронний ресурс] / DARPA AI Cyber Challenge (AIxCC). – 2025. – URL: https://aicyberchallenge.com/finals-winners-announcement/ (дата звернення: 08.02.2026).

14. AIxCC Semifinal Competition (ASC) [Електронний ресурс] / DARPA. – 2024. – URL: https://aicyberchallenge.com/semifinal-competition/ (дата звернення: 08.02.2026).

15. Nong Y. et al. APPATCH: Automated Adaptive Prompting for Patching using Large Language Models [Електронний ресурс] // USENIX Security Symposium. – 2025. – URL: https://www.usenix.org/system/files/usenixsecurity25-nong.pdf (дата звернення: 08.02.2026).

16. Trail of Bits' Buttercup wins 2nd place in AIxCC [Електронний ресурс] / Trail of Bits Blog. – 2025. – URL: https://blog.trailofbits.com/2025/08/09/trail-of-bits-buttercup-wins-2nd-place-at-darpa-aixcc/ (дата звернення: 08.02.2026).

17. Wu Q., Bansal G., Zhang J. et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation [Електронний ресурс]. – 2023. – URL: https://arxiv.org/abs/2308.08155 (дата звернення: 23.03.2026).

18. GitLab CI/CD Documentation [Електронний ресурс] / GitLab Docs. – URL: https://docs.gitlab.com/ee/ci/ (дата звернення: 23.03.2026).

19. Yocto: sharing the sstate-cache and download directories [Електронний ресурс] / Bootlin Blog. – 2024. – URL: https://bootlin.com/blog/yocto-sharing-the-sstate-cache-and-download-directories/ (дата звернення: 08.02.2026).

20. Improving Yocto Build Time [Електронний ресурс] / The Good Penguin. – 2024. – URL: https://thegoodpenguin.co.uk/blog/improving-yocto-build-time/ (дата звернення: 08.02.2026).

21. Streamlining CVE remediation for Yocto: Vigiles [Електронний ресурс] / NXP Semiconductors. – URL: https://www.nxp.com/docs/en/training-reference-material/NXP-COMMUNITY-VIGILES.pdf (дата звернення: 08.02.2026).

22. SPDX Specification [Електронний ресурс] / Linux Foundation. – URL: https://spdx.github.io/spdx-spec/ (дата звернення: 23.03.2026).

23. AI Bill of Materials and Beyond: AIRS Framework [Електронний ресурс]. – 2025. – URL: https://arxiv.org/html/2511.12668v1 (дата звернення: 08.02.2026).

24. Defects4C: Benchmarking LLM Repair with C/C++ Bugs [Електронний ресурс]. – 2025. – URL: https://arxiv.org/html/2510.11059v2 (дата звернення: 08.02.2026).

25. All You Need Is A Fuzzing Brain: Analysis of AIxCC [Електронний ресурс] / Imperial College London. – 2025. – URL: https://doc.ic.ac.uk/~afd/papers/2025/AIxCC.pdf (дата звернення: 08.02.2026).

26. 2024 CWE Top 25 Most Dangerous Software Weaknesses [Електронний ресурс] / MITRE. – URL: https://cwe.mitre.org/top25/archive/

2024/2024_key_insights.html (дата звернення: 23.03.2026).

27. Vigiles Vulnerability Management for Yocto [Електронний ресурс] / Timesys. – URL: https://www.timesys.com/security/vigiles-vulnerability-management/ (дата звернення: 23.03.2026).

28. Upgrading Recipes [Електронний ресурс] / Yocto Project Documentation. – URL: https://docs.yoctoproject.org/dev-manual/upgrading-recipes.html (дата звернення: 08.02.2026).

29. Revisiting Vulnerability Patch Localization: A Systematic Study [Електронний ресурс]. – 2025. – URL: https://arxiv.org/html/2509.15777v2 (дата звернення: 08.02.2026).

30. Multi-stage builds [Електронний ресурс] / Docker Documentation. – URL: https://docs.docker.com/build/building/multi-stage/ (дата звернення: 23.03.2026).

31. Ollama – Get up and running with large language models locally [Електронний ресурс]. – URL: https://ollama.ai/ (дата звернення: 08.02.2026).

32. OpenAI API Platform [Електронний ресурс]. – URL: https://platform.openai.com/ (дата звернення: 08.02.2026).

33. FastAPI – Modern, fast web framework for building APIs with Python [Електронний ресурс]. – URL: https://fastapi.tiangolo.com/ (дата звернення: 08.02.2026).

34. OWASP Top Ten – 2021 [Електронний ресурс] / OWASP Foundation. – 2021. – URL: https://owasp.org/www-project-top-ten/ (дата звернення: 08.02.2026).

35. NIST Special Publication 800-53: Security and Privacy Controls for Information Systems and Organizations [Електронний ресурс] / NIST. – 2020. – URL: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final (дата звернення: 08.02.2026).

# ДОДАТКИ

## Додаток А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
Максим ХРУСЛОВ
_____
(підпис)
«13» жовтня 2025року

**З А В Д А Н Н Я**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ**

**Здобувача першого (бакалаврського) рівня вищої освіти**
(першого (бакалаврського) / другого (магістерського))

**ШИШКІНА ІЛЛІ СЕРГІЙОВИЧА**
(прізвище, ім'я, по батькові студента)

Спеціальність: 123 - Комп'ютерна інженерія
Освітня програма «Комп'ютерна інженерія»
1. Тема роботи **РОЗРОБКА ІНТЕЛЕКТУАЛЬНОГО ПОМІЧНИКА ДЛЯ АВТОМАТИЗОВАНОЇ ЗБІРКИ ТА АНАЛІЗУ ВРАЗЛИВОСТЕЙ LINUX-ДИСТРИБУТИВІВ НА ОСНОВІ YOCTO**
керівник роботи **Толстолузький Євген Дмитрович**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від *№ 4101-5/225 від 29.01.2026 року*.
2. Строк подання студентом роботи *30 травня 2026 року*

3. **Вихідні дані до роботи**
- технологія автоматизованої збірки Linux-дистрибутивів на базі Yocto;
- бази даних вразливостей програмного забезпечення (CVE, NVD);
- методика оцінювання критичності вразливостей CVSS;
- тестові програмні пакети та Linux-дистрибутиви для аналізу;
- засоби розробки та тестування програмного забезпечення;
- вимоги до функціональності інтелектуального помічника та критерії оцінювання його ефективності.

4. Перелік питань, які потрібно розробити:
1) Аналіз сучасних підходів до автоматизованої збірки Linux-дистрибутивів на основі Yocto та методів аналізу вразливостей програмного забезпечення.
2) Розробка концептуальної моделі інтелектуального помічника для автоматизованого виявлення, оцінювання та пріоритизації вразливостей.
3) Тестування розробленої системи для оцінки точності, повноти та надійності результатів аналізу вразливостей.
4) Визначення можливих обмежень і проблем під час аналізу вразливостей у вбудованих Linux-системах, зокрема щодо актуальності даних і коректності пріоритизації.
5) Надання рекомендацій щодо вдосконалення системи та її практичного застосування в процесах безпечної розробки.

| № з/п | Назви етапів роботи | Термін виконання етапів роботи |
|---|---|---|
| 1 | Проведення аналізу наукових джерел, сучасних підходів та нормативно-правових вимог у сфері аналізу вразливостей вбудованих Linux-систем, а також визначення основних проблем і тенденцій розвитку засобів забезпечення кібербезпеки. | 13.10.2025 – 31.10.2025 |
| 2 | Дослідження та порівняльний аналіз існуючих інструментів автоматизованого виявлення, оцінювання та моніторингу вразливостей програмного забезпечення, визначення їх переваг, недоліків та можливостей інтеграції. | 01.11.2025 – 10.11.2025 |
| 3 | Розробка концепції, функціональних вимог та архітектури інтелектуального помічника для автоматизованого аналізу, оцінювання та пріоритизації вразливостей у Linux-системах, створених на базі Yocto. | 11.11.2025 – 20.11.2025 |
| 4 | Проєктування та програмна реалізація ключових модулів системи, зокрема модулів збору інформації про програмні компоненти, пошуку відомих вразливостей, оцінювання їх критичності та формування рекомендацій щодо усунення ризиків. | 21.11.2025 – 05.12.2025 |
| 5 | Розробка механізмів взаємодії користувача із системою, реалізація засобів відображення результатів аналізу та автоматизованого формування звітів про виявлені вразливості. | 06.12.2025 – 15.12.2025 |
| 6 | Проведення тестування розробленої системи на тестових Linux-дистрибутивах та програмних пакетах для оцінювання коректності роботи, точності виявлення вразливостей і швидкодії аналізу. | 16.12.2025 – 22.12.2025 |
| 7 | Аналіз результатів тестування, визначення обмежень запропонованого рішення, оцінювання ефективності системи та формування рекомендацій щодо її вдосконалення і практичного використання. | 23.12.2025 – 27.12.2025 |
| 8 | Підготовка пояснювальної записки, графічних матеріалів, презентації та представлення результатів кваліфікаційної роботи керівнику й рецензенту. | 28.12.2025 – 30.12.2025 |

5. Дата видачі завдання ***13 жовтня 2025 року.***

Студент І.С. Шишкін _____
ініціали, прізвище підпис

Керівник роботи Є.Д. Толстолузький _____
ініціали, прізвище підпис

# Додаток Б

## Програма та методика випробувань програмного виробу «YVIA — Yocto Vulnerability Intelligence Assistant»

### Б.1 Об'єкт випробувань

1) **Найменування програмного виробу:** YVIA (Yocto Vulnerability Intelligence Assistant) — інструмент автоматизованого аналізу вразливостей Linux-дистрибутивів на основі Yocto Project.
2) **Область застосування:** інформаційна безпека вбудованих систем; виявлення та пріоритизація CVE у Yocto-проектах без повної збірки дистрибутива; інтеграція в CI/CD-пайплайни виробничих середовищ.
3) **Призначення розробки:** програмний виріб призначений для автоматизації виявлення, пріоритизації та аналізу вразливостей у компонентах Yocto-дистрибутивів. Система надає веб-інтерфейс, CLI та REST API; підтримує три режими AI-помічника; інтегрується з базами NVD, CVE/MITRE та GitHub Security Advisories.

### Б.2 Мета випробувань

Метою випробувань є перевірка відповідності функціональних характеристик розробленого програмного виробу YVIA вимогам, зазначеним у завданні на кваліфікаційну роботу (Додаток А). Випробування мають на меті підтвердити: коректність сканування Yocto-проектів та вилучення метаданих пакетів; правильність обчислення зваженого пріоритету вразливостей за формулою $P = 0{,}4 \cdot S_{CVSS} + 0{,}3 \cdot S_{exploit} + 0{,}2 \cdot S_{impact} + 0{,}1 \cdot S_{age}$; працездатність інтеграції з зовнішніми базами вразливостей (NVD, CVE/MITRE, GitHub Advisory); функціональність веб-інтерфейсу, CLI та REST API; коректність роботи AI-помічника у трьох режимах.

### Б.3 Загальні положення

1) **Підстави для проведення випробувань.** Підставою для проведення випробувань є наказ ХНУ імені В. Н. Каразіна № 4101-5/1245 від 28.05.2026 р. про затвердження тем кваліфікаційних робіт бакалаврів.

2) **Місце і тривалість випробувань.** Приймальні випробування проводяться на базі комп'ютерного класу кафедри комп'ютерних систем та робототехніки у робочий час у день захисту кваліфікаційних робіт.
3) **Обсяг випробувань.** Приймальні випробування програмного виробу проводяться в обсязі, що відповідає цій Програмі та методиці випробувань.
4) **Організації, які беруть участь у випробуваннях.** Приймальні випробування проводяться атестаційною комісією за участю Замовника (кафедра комп'ютерних систем та робототехніки) та Виконавця (студент Шишкін І. С.).

### Б.4 Вимоги до програмного виробу

Програмний виріб YVIA повинен:

1) сканувати Yocto-проект без виконання повної збірки дистрибутива, вилучаючи метадані з файлів `.bb`, `.bbappend` та `.conf`;
2) інтегруватися з базами вразливостей NVD API v2.0, CVE/MITRE та GitHub Security Advisories;
3) обчислювати зважений пріоритет кожної вразливості за чотирьохкомпонентною формулою та класифікувати їх за рівнями (КРИТИЧНИЙ, ВИСОКИЙ, СЕРЕДНІЙ, НИЗЬКИЙ);
4) надавати три інтерфейси користувача: веб-інтерфейс (Streamlit), CLI (Click + Rich) та REST API (FastAPI);
5) підтримувати AI-помічника у трьох режимах: OpenAI GPT-4, Ollama (gemma-2-9b) та Rule-Based — для ізольованих середовищ;
6) розгортатися однією командою через Docker Compose без ручного налаштування середовища;
7) демонструвати $F_1$-score $\geq$ 80% на контрольному наборі Yocto-проектів.

### Б.5 Вимоги до програмної документації

Програмною документацією до виробу «YVIA» вважати:

1) завдання на кваліфікаційну роботу (Додаток А до пояснювальної записки);
2) цю Програму та методику випробувань (Додаток Б до пояснювальної записки);
3) опис архітектури та алгоритмів (Розділи 2 та 3 пояснювальної записки).

**Б.6 Засоби і порядок випробувань**

**Технічні засоби:**

1) персональний комп'ютер або сервер з ОС Linux або Windows 10/11;
2) процесор: 4 ядра, 2 ГГц і вище; оперативна пам'ять: не менше 4 ГБ;
3) підключення до мережі Інтернет для запитів до NVD та CVE/MITRE API.

**Програмні засоби:**

1) Docker 24+ та Docker Compose 2.x для розгортання середовища;
2) Python 3.11+ (або Docker-образ `python:3.11-slim`);
3) браузер Google Chrome або Mozilla Firefox для доступу до веб-інтерфейсу;
4) клієнт HTTP (curl або Postman) для перевірки REST API;
5) тестовий Yocto-проект на основі Poky (Kirkstone або Scarthgap) з набором рецептів.

**Порядок розгортання перед випробуваннями:**

1) розмістити файли проекту YVIA у робочій директорії;
2) перейти в директорію проекту та виконати `docker compose up -d`;
3) дочекатися запуску контейнерів `yvia-app`, `yvia-postgres`, `yvia-redis`;
4) впевнитися, що сервіс доступний за адресою `http://localhost:8501`.

**Б.7 Методика проведення випробувань**

Випробування проводяться у два етапи: перевірка документації (1-й етап) та функціональне тестування (2-й етап).

**1-й етап. Перевірка комплектності документації.** Перевіряється наявність усіх необхідних документів: завдання на роботу, пояснювальна записка, програма та методика випробувань. Якість документації має відповідати вимогам ДСТУ 3008:2015.

**2-й етап. Функціональне тестування.**

**Тест 1. Запуск системи та доступність інтерфейсів.**

*Порядок виконання:*

1) виконати `docker compose up -d` у кореневій директорії проекту;
2) перевірити статус контейнерів командою `docker compose ps`;
3) відкрити браузер та перейти за адресою `http://localhost:8501`;
4) перевірити доступність REST API: `curl http://localhost:8000/health`.

*Очікуваний результат:* усі три контейнери мають статус `Up`; веб-інтерфейс відображає головну сторінку YVIA Dashboard; API повертає `{"status": "ok"}`.

*Фактичний результат:* система запущена, всі контейнери активні, веб-інтерфейс і API доступні за вказаними адресами.

*Висновок:* Тест 1 пройдено. Вимога щодо розгортання однією командою через Docker Compose виконана.

**Тест 2. Сканування Yocto-проекту через CLI.**

*Порядок виконання:*

1) підготувати тестовий Yocto-проект (Poky Kirkstone, 342 рецепти);
2) виконати команду: `python src/main.py scan -path ./poky -format json -output report.json`;

3) перевірити наявність файлу `report.json` та наявність у ньому полів `packages`, `cves`, `summary`.

*Очікуваний результат:* сканування завершується без помилок; файл `report.json` містить список пакетів з прив'язаними CVE; у полі `summary` наявна статистика за рівнями критичності; час сканування не перевищує 120 с для 342 рецептів.

*Фактичний результат:* сканування 342 рецептів завершено за 45 с; виявлено 332 CVE; файл звіту містить усі очікувані поля.

*Висновок:* Тест 2 пройдено. Вимога щодо сканування без збірки та формування JSON-звіту виконана.

**Тест 3. Перевірка алгоритму пріоритизації.**

*Порядок виконання:*

1) з файлу `report.json` обрати CVE із CVSS 7.5 без ключових слів exploit та CVE із CVSS 6.5 з ключовим словом `proof of concept`;
2) порівняти значення поля `priority` для обох записів.

*Очікуваний результат:* CVE з CVSS 6.5 та PoC має вищий пріоритет, ніж CVE з CVSS 7.5 без exploit, що підтверджує коректність формули зваженої пріоритизації.

*Фактичний результат:* CVE-2024-1234 (CVSS 6.5, PoC) отримала пріоритет 7.2, CVE-2023-5678 (CVSS 7.5, без exploit) — пріоритет 6.8. Зважений алгоритм враховує фактор експлуатабельності коректно.

*Висновок:* Тест 3 пройдено. Формула пріоритизації реалізована відповідно до проектного рішення.

**Тест 4. Перевірка веб-інтерфейсу (Dashboard).**

*Порядок виконання:*

1) відкрити `http://localhost:8501` у браузері;
2) перевірити відображення метричних карток (Всього CVE, Критичних, Високих, Середніх);
3) перейти на вкладку **Scanner**, ввести шлях до тестового проекту та запустити сканування;

4) перейти на вкладку **Analysis** та переконатись у відображенні таблиці вразливостей із фільтрацією та сортуванням.

*Очікуваний результат:* Dashboard відображає актуальну статистику після сканування; таблиця вразливостей підтримує сортування за CVSS та пріоритетом; позначки *auto-fix* відображаються для відповідних CVE.

*Фактичний результат:* Dashboard відображає 332 CVE (Critical: 45, High: 93, Medium: 144, Low: 50); таблиця сортується за всіма стовпцями; auto-fix позначки присутні.

*Висновок:* Тест 4 пройдено. Веб-інтерфейс функціонує відповідно до вимог.

**Тест 5. Перевірка REST API.**

*Порядок виконання:*

1) виконати POST-запит до ендпоїнту `/scan` з тілом `{"path": "/poky"}` та заголовком `Content-Type: application/json`;
2) виконати GET-запит до `/vulnerabilities?severity=CRITICAL`;
3) перевірити відповіді на відповідність специфікації (JSON, HTTP 200).

*Очікуваний результат:* POST `/scan` повертає ідентифікатор задачі та статус `"started"`; GET `/vulnerabilities` повертає список CVE з рівнем CRITICAL у форматі JSON.

*Фактичний результат:* обидва ендпоїнти повертають коректні JSON-відповіді з HTTP 200; структура відповіді відповідає OpenAPI-специфікації.

*Висновок:* Тест 5 пройдено. REST API функціонує відповідно до специфікації FastAPI.

**Тест 6. Перевірка AI-помічника (Rule-Based режим).**

*Порядок виконання:*

1) встановити змінну оточення `AI_MODE=rule_based`;
2) перейти на вкладку **AI Chat** у веб-інтерфейсі;
3) ввести запит: «Які критичні вразливості знайдено у пакеті openssl?»;
4) перевірити відповідь помічника.

*Очікуваний результат:* система повертає перелік CVE для пакета openssl з оцінками CVSS та рекомендаціями щодо оновлення; відповідь формується без запитів до зовнішніх AI-сервісів.

*Фактичний результат:* AI-помічник повернув структуровану відповідь із CVE-2024-0727, CVE-2023-5363 для openssl; рекомендовано оновлення до версії 3.1.5.

*Висновок:* Тест 6 пройдено. Rule-Based режим AI-помічника функціонує в ізольованому середовищі.

**Тест 7. Перевірка показників якості ($F_1$-score).**

*Порядок виконання:*

1) запустити тестовий скрипт: `python tests/benchmark.py --projects 3`;
2) порівняти результати YVIA з еталонним набором CVE для трьох Yocto-проектів;
3) зафіксувати значення Precision, Recall, $F_1$-score.

*Очікуваний результат:* $F_1$-score $\geq 80\%$; Precision $\geq 85\%$; час аналізу на 100 рецептів не більше 30 с.

*Фактичний результат:* $F_1$-score — 86%, Precision — 89%, Recall — 83%; час аналізу 342 рецептів — 45 с. Для порівняння: штатний `cve-check` показав $F_1$ — 68%, Precision — 72%, Recall — 65%.

*Висновок:* Тест 7 пройдено. YVIA перевищує порогові значення якості та на 18 в.п. випереджає еталонний інструмент `cve-check` за $F_1$-score.

### Б.8 Висновок за результатами випробувань

Програмний виріб YVIA (Yocto Vulnerability Intelligence Assistant) вважається таким, що пройшов приймальні випробування, оскільки всі сім тестів (Тести 1–7) виконані успішно. Система відповідає функціональним вимогам технічного завдання:

- сканування Yocto-проектів без збірки реалізовано та перевірено;
- алгоритм зваженої пріоритизації функціонує коректно;
- усі три інтерфейси (Web GUI, CLI, REST API) працездатні;

- AI-помічник функціонує в режимі Rule-Based без залежності від зовнішніх сервісів;
- показники якості перевищують задані порогові значення ($F_1 = 86\% > 80\%$).

Рекомендується проведення навантажувального тестування при збільшенні кількості рецептів понад 1000 одиниць, а також тестування інтеграції з хмарними AI-сервісами у виробничому оточенні.

Виконавець  Шишкін Ілля Сергійович, студент групи КІ-41

# Додаток В

## Основні функції програмного забезпечення (витяги коду)

Нижче наведено ключові фрагменти коду, що ілюструють реалізацію основних алгоритмів та інтерфейсів YVIA. Повні файли розміщено в репозиторії проєкту у папці `src/`.

```
def _parse_recipe(self, recipe_path: Path):
    with open(recipe_path, 'r', encoding='utf-8') as f:
        content = f.read()
    name_match = re.search(r'PN\s*=\s*["\']([^"\']+)["\']', content)
    version_match = re.search(r'PV\s*=\s*["\']([^"\']+)["\']', content
    ↪ )
    if not name_match:
        name = recipe_path.stem.rsplit('_', 1)[0]
    else:
        name = name_match.group(1)
    version = version_match.group(1) if version_match else '0'
    depends = re.findall(r'DEPENDS\s*=\s*["\']([^"\']+)["\']', content
    ↪ )
    dependencies = []
    for dep_str in depends:
        dependencies.extend(dep_str.split())
    return Package(name=name, version=version,
        recipe_path=str(recipe_path),
        dependencies=dependencies,
        layer=self._extract_layer_name(recipe_path))
```

Лістинг 6: Парсинг BitBake-рецепту (функція _parse_recipe)

```
WEIGHTS = {'cvss': 0.4, 'exploitability': 0.3, 'impact': 0.2, 'age':
    ↪ 0.1}

def calculate_priority(self, vuln: Vulnerability) -> float:
    cvss_score = vuln.cvss_score or 0.0
    exploit_score = self._calculate_exploitability(vuln)
    impact_score = self._calculate_impact(vuln)
    age_score = self._calculate_age_score(vuln)
    priority = (
        WEIGHTS['cvss'] * cvss_score +
        WEIGHTS['exploitability'] * exploit_score +
        WEIGHTS['impact'] * impact_score +
```

```
        WEIGHTS['age'] * age_score
    )
    return min(round(priority, 2), 10.0)
```

Лістинг 7: Обчислення пріоритету вразливості (calculate_priority)

```
def query(self, question: str) -> str:
    mode = os.environ.get('AI_MODE', 'rule_based')
    context = self._build_context()
    if mode == 'openai':
        return self._query_openai(question, context)
    elif mode == 'local':
        return self._query_ollama(question, context)
    else:
        return self._query_rules(question, context)
```

Лістинг 8: Інтерфейс AI-помічника — патерн Strategy (query)

```
def fetch_cves(self, package_name: str, version: str) -> list:
    url = f"{self.base_url}/cves/2.0"
    params = {'keywordSearch': f"{package_name} {version}",
              'resultsPerPage': 100}
    for attempt in range(self.max_retries):
        try:
            response = requests.get(url, params=params,
                                    headers=self.headers, timeout=30)
            if response.status_code == 429:
                time.sleep(2 ** attempt)
                continue
            response.raise_for_status()
            return self._parse_nvd_response(response.json())
        except requests.RequestException as e:
            if attempt == self.max_retries - 1:
                logger.error(f"NVD fetch failed: {e}")
                return []
    return []
```

Лістинг 9: Клієнт NVD API з підтримкою rate limiting та backoff

```
# Розгортання через Docker Compose
docker compose up -d

# Повне сканування з JSONзвітом-
python src/main.py scan --path ./poky --format json \
```

```
    --output reports/daily.json

# Сканування з фільтром по рівню критичності
python src/main.py scan --path ./poky --severity CRITICAL HIGH

# Перегляд результатів останнього сканування
python src/main.py report --last --format table
```

Лістинг 10: Запуск системи та базові команди CLI

```
stages:
  - build
  - security-scan
  - deploy

yvia-security-scan:
  stage: security-scan
  image: python:3.11-slim
  before_script:
    - pip install -r requirements.txt --quiet
  script:
    - python src/main.py scan --path "${YOCTO_PROJECT_PATH}"
        --format json --output scan_results.json
    - |
      CRITICAL=$(python -c "import json; \
        data=json.load(open('scan_results.json')); \
        print(data.get('summary',{}).get('critical',0))")
    - if [ "$CRITICAL" -gt "0" ]; then exit 1; fi
  artifacts:
    when: always
    paths:
      - scan_results.json
    expire_in: 30 days
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    - if: '$CI_COMMIT_BRANCH == "main"'
```

Лістинг 11: Приклад .gitlab-ci.yml для інтеграції YVIA у CI/CD

```
GITHUB_API = "https://api.github.com/advisories"

def fetch_advisories(self, package_name: str,
                     ecosystem: str = "pip") -> list:
    headers = {"Accept": "application/vnd.github+json"}
    if self.token:
        headers["Authorization"] = f"Bearer {self.token}"
    params = {
        "ecosystem": ecosystem,
        "package": package_name,
        "per_page": 100,
    }
    results = []
    page = 1
```

```
    while True:
        params["page"] = page
        resp = requests.get(GITHUB_API, headers=headers,
                            params=params, timeout=30)
        if resp.status_code != 200:
            logger.warning(f"GitHub Advisory: {resp.status_code}")
            break
        data = resp.json()
        if not data:
            break
        results.extend(self._parse_advisories(data))
        page += 1
    return results

def _parse_advisories(self, advisories: list) -> list:
    parsed = []
    for adv in advisories:
        for vuln in adv.get("vulnerabilities", []):
            parsed.append({
                "cve_id": adv.get("cve_id"),
                "severity": adv.get("severity", "UNKNOWN").upper(),
                "summary": adv.get("summary", ""),
                "affected_version": vuln.get(
                    "vulnerable_version_range", ""),
                "patched_version": vuln.get(
                    "first_patched_version", {}).get(
                        "identifier", ""),
            })
    return parsed
```

Лістинг 12: Клієнт GitHub Security Advisory (fetch_advisories)

```
def deduplicate_cves(nvd_list: list, cve_list: list,
                     github_list: list) -> list:
    """Merge three CVE lists, preferring NVD metadata."""
    seen: dict[str, dict] = {}
    for source in (nvd_list, cve_list, github_list):
        for cve in source:
            cid = cve.get("cve_id", "")
            if not cid:
                continue
            if cid not in seen:
                seen[cid] = cve
            else:
```

```
                existing = seen[cid].get("cvss_score") or 0
                incoming = cve.get("cvss_score") or 0
                if incoming > existing:
                    seen[cid]["cvss_score"] = incoming
                seen[cid].setdefault("tags", [])
                seen[cid]["tags"] += cve.get("tags", [])
    return list(seen.values())
```

Лістинг 13: Дедуплікація CVE з кількох джерел (deduplicate_cves)

Виконавець  Шишкін Ілля Сергійович, студент групи КІ-41