МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут автоматики і електротехніки

Кафедра комп'ютерних технологій та інформаційної безпеки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних технологій та інформаційної безпеки, к.т.н., доцент
_____ С.М. Нужний
«10» 12 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

АЛГОРИТМИ ВИЯВЛЕННЯ DDOS-АТАК В ІНФОРМАЦІЙНИХ СИСТЕМАХ

Ступінь вищої освіти: магістр
Галузь знань: 14 Електрична інженерія
Спеціальність: 141 «Електроенергетика, електротехніка та електромеханіка»
Освітньо-професійна програма: Системи технічного захисту інформації, автоматизація її обробки

Виконав: студент 6 курсу групи 6366м
Погребняк Олег Віталійович _____

Кваліфікаційна робота містить результати самостійного виконання навчального завдання. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Керівник:
к.т.н., доцент, доцент кафедри комп'ютерних технологій та інформаційної безпеки

Козирєв Сергій Сергійович _____

Миколаїв – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут автоматики і електротехніки

ЗАТВЕРДЖУЮ
Гарант освітньої програми,
д.т.н., професор,
професор кафедри КТІБ
_____ Передерій В.І.
«14» 10 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу

Студент: Погребняк Олег Віталійович

Курс:6

Група: 6366м

Ступінь вищої освіти: магістр

Галузь знань: 14 Електрична інженерія

Спеціальність: 141 «Електроенергетика, електротехніка та електромеханіка»

Освітньо-професійна програма: Системи технічного захисту інформації, автоматизація її обробки

1. Тема роботи: Алгоритми виявлення DDoS-атак в інформаційних системах

затверджена наказом НУК від «14» 10 2025 р. № 1217-с/2

2. Вихідні дані до роботи: формати лог-файлів популярних брандмауерів та інтернет сервісів; методи аналізу трафіку мережевого адаптера; існуючі методи атак грубої сили; нормативно-правова база і теоретичні основи побудови захищених комп'ютерних систем і систем підтримки прийняття рішень.

3. Перелік питань, які необхідно розробити: Провести аналіз відкритих джерел та визначити критерії класифікації DDoS-атак; Визначити параметри наявних на ринку засобів захисту та обґрунтувати критерії прийняття рішень щодо їх вибору; Розробити основні алгоритми функціонування системи; Створити структуру системи підтримки прийняття рішень щодо вибору засобів захисту від DDoS-атак як складової комплексу засобів захисту.

4. Терміни виконання завдання:
термін видачі завдання: «01- 05» вересня 2025 р.
термін подання роботи: «18 » грудня 2025 р.

6. План-графік виконання кваліфікаційної роботи

п/п	Заходи	Дата		Готовність
		Навч. тиждень	Контрольна дата	
1.	Наукове стажування	1 - 8	27.10.2025	Звіт з наукового стажування
2.	Організаційні збори	9	29.10.2025	Попередній варіант змісту КР
3.	Рубіжний контроль	10	05.11.2025	Попередній варіант оглядових розділів, звіт з практика
4.	Рубіжний контроль	13	27-28.11.2025	Доповідь на 13-ій Всеукраїнській науково-технічній конференції з міжнародною участю «СПІБТ–2025»
5.	Рубіжний контроль	14	3.12.2025	1. Попередній варіант повної КР 2. Попередній варіант презентації
6.	КЕ на рівень «магістра»	15	8,9.12.2025	Складання державного екзамену зі спеціальності на ступінь магістра
7.	Перевірка на плагіат	15	10.12.2025	Електронний варіант кваліфікаційної роботи, попередній захист (робота передається студентом на кафедру для внутрішньої рецензії).
8.	Оформлення КР та супутньої документації	16	15-17.12.2025	1. Оформлена КР (в форматі pdf) У разі отримання позитивної внутрішньої рецензії, КР подається на зовнішню рецензію. 2. Оформлена презентація (в форматі pdf).
9.	Подання документів на захист	16	18.12.2025	1. Подання щодо захисту КР: тема, успішність, висновок керівника та висновок кафедри про КР. 2. Зовнішня рецензія (окремо від КР). Резюме на КР. 3. Електронний варіант презентації. 4. Електронний варіант КР на диску
10.	Захист ДР	17	22,23,24 12.2025	1. Приєднання студента до засідання кваліфікаційної комісії (конференції) у встановлений час для проведення процедури дистанційного захисту 2. Процедура захисту КР.

Керівник
к.т.н., доцент, доцент кафедри комп'ютерних технологій
та інформаційної безпеки, _____ С.С. Козирєв

Студент _____ О.В. Погребняк

АНОТАЦІЯ

Погребняк О.В. Алгоритми виявлення DDoS-атак в інформаційних системах.

Кваліфікаційна робота на здобуття ступеня вищої освіти «магістр» за спеціальністю 141 «Електроенергетика, електротехніка та електромеханіка» галузі знань 14 «Інформаційні технології» і освітньо-професійною програмою «Системи технічного захисту інформації, автоматизація її обробки». – Національний університет кораблебудування імені адмірала Макарова, Миколаїв, 2025.

Пояснювальна записка: 68 с., 8 рис., 2 табл., 4 додатки, 52 посилання.

Кваліфікаційна робота присвячена актуальній темі - розробці ефективних систем захисту комп'ютерних мереж від сучасних *DDoS* атак.

Об'єктом дослідження в даній роботі є системи технічного захисту комп'ютерних мереж. Предметом дослідження є засоби протидії *DDoS*-атакам. Методологічну основу становлять теорія побудови захищених мереж та метод аналізу ієрархій (*MAI*). Метою роботи є розробка системи підтримки прийняття рішень для вибору засобів захисту від *DDoS* як елемента комплексної безпеки.

У роботі класифіковано *DDoS*-атаки та проаналізовано наявні засоби захисту, визначивши критерії для вибору оптимального рішення. На основі порівняння методів виявлення загроз обрано найефективніші алгоритми (*Random Forest, LSTM, CNN-LSTM*) для використання в *IoT*, *SDN* та хмарних інфраструктурах. Розроблено архітектуру системи підтримки прийняття рішень для автоматизованого вибору засобів захисту, а також запропоновано механізм її адаптації до реальних сценаріїв.

Робота має практичну значимість - її результати можуть бути використані при проектуванні та модернізації захищених локальних та корпоративних мереж, а також у навчальному процесі за освітньо-професійною програмою «Системи технічного захисту інформації, автоматизація її обробки».

Ключові слова: кібербезпека, *DDoS*-атака, прийняття рішень, система підтримки прийняття рішень, критерії вибору, машинне навчання

SUMMARY

Pohrebniak O.V. The DDoS attacks detection algorithms in information systems.

Master's Thesis for the degree of "Master" in Specialty 141 "Electrical Power Engineering, Electrical Engineering and Electromechanics", Field of Knowledge 14 "Information Technologies", Educational-Professional Program "Technical Information Protection Systems, Automation of its Processing". – Admiral Makarov National University of Shipbuilding, Mykolaiv, 2025.

Explanatory note: 68 pages, 8 figures, 2 tables, 4 appendices, 52 references.

The Master's thesis is devoted to the topical issue of developing effective protection systems for computer networks against modern DDoS attacks.

The object of the study is the technical protection systems of computer networks. The subject of the study is the means of countering DDoS attacks. The methodological basis comprises the theory of building secure networks and the Analytic Hierarchy Process (AHP). The aim of the work is to develop a decision support system for selecting DDoS protection means as a component of complex network security.

The paper classifies DDoS attacks and analyzes existing protection means, defining criteria for selecting the optimal solution. Based on a comparison of threat detection methods, the most effective algorithms (Random Forest, LSTM, CNN-LSTM) were selected for use in IoT, SDN, and cloud infrastructures. The architecture of a decision support system for the automated selection of protection means has been developed, and a mechanism for its adaptation to real-world scenarios has been proposed.

The work has practical value – its results can be used in the design and modernization of secure local and corporate networks, as well as in the educational process under the educational-professional program "Technical Information Protection Systems, Automation of its Processing".

Keywords: *cybersecurity, DDoS attack, decision making, decision support system, selection criteria, machine learning.*

ЗМІСТ

ВСТУП

Зосередження великих обсягів інформації, в тому числі інформації з обмеженим доступом (ІзОД), в інформаційних ресурсах, а також постійне зростання складності та інтенсивності кіберзагроз зумовлюють актуальність забезпечення надійного захисту цих ресурсів від цілеспрямованих і випадкових загроз. Серед найбільш поширених і руйнівних видів кібератак - розподілені атаки заперечення обслуговування (*DDoS*), які напряму порушують принцип доступності, що є одним із трьох фундаментальних компонентів моделі безпеки CIA. У сучасних умовах ефективна протидія таким атакам вимагає не лише технічних засобів, а й системного підходу до вибору та застосування відповідних механізмів захисту.

Розробці методів та засобів протидії *DDoS*-атакам присвячено багато наукових робіт українських [10, 22, 41] і зарубіжних науковців [1–6]. На даний час на ринку наявний ряд спеціалізованих апаратно-програмних засобів захисту від *DoS* і *DDoS*-атак [7–9, 11, 23, 29, 34, 36–39], які базуються на різних технічних підходах - від хмарного скрубінгу трафіку до локальних систем виявлення на основі машинного навчання. Проте, незважаючи на широкий вибір, методика їх обґрунтованого та оптимального вибору з урахуванням специфіки інфраструктури, рівня загроз та обмежень ресурсів на даний час залишається недостатньо розробленою.

Метою даної роботи є розробка системи підтримки прийняття рішень для ефективного та оптимального вибору засобів захисту від *DDoS*-атак, що забезпечує підвищення рівня безпеки комп’ютерних мереж шляхом автоматизації процесу аналізу характеристик загроз, вибору відповідних засобів протидії та обґрунтування рішень на основі заданих критеріїв.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

− провести аналіз відкритих літературних джерел для визначення ознак і класифікацій *DDoS*-атак і характеристик існуючих на ринку засобів захисту;

− визначити критерії прийняття рішень щодо вибору оптимальних засобів захисту від *DDoS*-атак;

− розробити алгоритми і структуру системи, яка забезпечить підтримку прийняття рішень при виборі засобів захисту від *DDoS*-атак, як частини комплексного захисту комп'ютерних мереж.

Об'єктом дослідження в даній роботі є системи технічного захисту комп'ютерних мереж.

Предметом дослідження в даній роботі є засоби захисту від *DDoS*-атак.

Методами дослідження є теоретичний аналіз основ побудови захищених мереж, методологія створення систем підтримки прийняття рішень, застосування методу аналізу ієрархій (*MAI*) для визначення оптимальних рішень.

Наукова новизна даної кваліфікаційної роботи полягає у розробці комплексного підходу до вибору засобів захисту від *DDoS*-атак на основі багатокритеріальної оцінки, що враховує не лише технічні характеристики рішень, а й контекст їх застосування - включаючи тип інфраструктури (*IoT*, *SDN*, хмара), рівень загроз, обчислювальні обмеження та вимоги до доступності. Запропонована система поєднує аналіз сучасних алгоритмів виявлення (зокрема *Random Forest, LSTM та CNN-LSTM)* із формалізованою процедурою прийняття рішень, що дозволяє обґрунтовано вибирати найбільш ефективне рішення для конкретного сценарію.

Практична значимість даної кваліфікаційної роботи полягає у створенні готового до впровадження інструменту, що дозволяє IT-фахівцям, адміністраторам безпеки та керівникам інформаційних служб приймати зважені рішення щодо захисту своїх мереж від *DDoS*-атак. Розроблена система підтримки прийняття рішень може бути інтегрована в існуючі процеси управління кібербезпекою, сприяючи раціональному використанню бюджету та підвищенню загальної стійкості інформаційної інфраструктури.

Результати роботи можуть бути застосовані у галузях, де критично важлива доступність інформаційних систем: фінансовий сектор, енергетика, телекомунікації, державне управління та критична інфраструктура.

1 ТЕОРЕТИЧНІ ОСНОВИ ВИЯВЛЕННЯ DDOS-АТАК

1.1 Поняття та класифікація *DDoS*-атак

DDoS-атака (*Distributed Denial of Service*, або Розподілений Відмову в Обслуговуванні) - це тип кібератаки, спрямований на порушення нормальної роботи веб-сайту, сервера чи мережевого ресурсу шляхом перевантаження його трафіком або запитами. Основна ідея полягає в тому, щоб зробити ресурс недоступним для легітимних користувачів, "затопивши" його фальшивими запитами з багатьох джерел одночасно. На відміну від простої *DoS*-атаки, де атака походить з одного джерела, *DDoS* використовує розподілену мережу комп'ютерів (ботнет), які можуть бути зараженими шкідливим ПЗ, щоб генерувати масовий трафік. Це робить атаку більш потужною та складнішою для блокування, оскільки трафік надходить з тисяч або мільйонів IP-адрес по всьому світу. *DDoS*-атаки часто мотивуються фінансовими інтересами (наприклад, вимагання викупу), політичними причинами (кібератівізм), конкуренцією в бізнесі або просто для демонстрації сили. Вони можуть тривати від кількох хвилин до днів, спричиняючи значні збитки, такі як втрата доходу, репутаційні проблеми та витрати на відновлення [1].

Історія *DDoS*-атак сягає кінця 1990-х років, коли перші масові атаки були зафіксовані проти великих компаній, таких як *Yahoo*! чи *eBay*. З часом технології еволюціонували: від простих SYN-флудів до складних багаторівневих атак, що враховують протоколи прикладного рівня. Сучасні *DDoS*-атаки можуть досягати обсягів у терабіти на секунду, як у випадку з атакою на *GitHub* у 2018 році, яка сягнула 1,35 Тбіт/с. Захист від них включає використання спеціалізованих сервісів, таких як *Cloudflare* чи *Akamai*, які фільтрують трафік, а також впровадження заходів на рівні мережі, як-от обмеження швидкості запитів чи використання CAPTCHA. Однак, повний захист неможливий, оскільки атакуючі постійно адаптуються, використовуючи нові вразливості, наприклад, в *IoT*-пристроях (як ботнет *Mirai* у 2016 році) [2].

DDoS-атаки класифікуються за кількома критеріями, залежно від рівня мережевого стеку, на якому вони відбуваються, типу трафіку чи методу виконання. За моделлю *OSI*, атаки поділяються на мережевий рівень (*L3/L4*), транспортний рівень та прикладний рівень (*L7*). На мережевому рівні (наприклад, *ICMP*-флуд або *UDP*-флуд) атака фокусується на перевантаженні пропускної здатності мережі, надсилаючи масу пакетів, які не потребують відповіді, що призводить до вичерпання ресурсів роутерів чи *firewall*. Транспортний рівень включає *SYN*-флуд, де атакуючі надсилають неповні запити на з'єднання, змушуючи сервер тримати відкриті порти в очікуванні, доки не вичерпаються ресурси. Ці атаки відносно прості, але ефективні проти незахищених систем [3].

За типом трафіку *DDoS*-атаки бувають об'ємними (*volumetric*), протокольними (*protocol*) та прикладними (*application*). Об'ємні атаки, як *NTP amplification*, спрямовані на заповнення каналу зв'язку масовим трафіком, часто використовуючи посилення (*amplification*), коли запит до сервера генерує відповідь у десятки разів більшу. Протокольні атаки експлуатують вразливості в мережевих протоколах, наприклад, *Smurf*-атака, де пакети розсилаються на *broadcast*-адреси, змушуючи всі пристрої в мережі відповідати. Прикладні атаки (L7) є найскладнішими, оскільки імітують легітимний трафік, як *HTTP*-флуд, де тисячі запитів на завантаження сторінки перевантажують веб-сервер, вимагаючи обробки кожного. Вони важко виявляються, бо виглядають як нормальна активність користувачів [4].

Інша класифікація - за методом організації: прямі, відбиті (*reflection*) та посилені (*amplification*). Прямі атаки генерують трафік безпосередньо з ботнету. Відбиті використовують посередників, наприклад, *DNS*-сервери, куди надсилаються запити з підробленою *IP*-адресою жертви, і відповіді йдуть на жертву. Посилені атаки комбінують відбиття з множенням трафіку, як у *Memcached*-атаках, де маленький запит викликає гіганську відповідь. Крім того, атаки класифікують за тривалістю (короткочасні vs. тривалі) та метою (відмова в обслуговуванні vs. маскування інших атак). Розуміння цієї класифікації

допомагає в розробці стратегій захисту, адаптованих до конкретного типу загрози [5].

1.2 Механізми реалізації *DDoS*-атак (наприклад, *SYN-flood, UDP-flood, HTTP-flood* тощо)

DDoS-атаки (*Distributed Denial of Service*) представляють собою скоординовані спроби зробити мережевий ресурс недоступним для легітимних користувачів шляхом перевантаження його ресурсів. Механізми реалізації таких атак варіюються залежно від рівня *OSI*-моделі, на якому вони діють: від мережевого рівня (волюметричні атаки) до рівня додатків. Нижче надано детальний огляд ключових механізмів, базуючись на класичних прикладах, таких як *SYN-flood, UDP-flood, HTTP-flood*, а також інших типах. Опис фокусується на високорівневих принципах роботи, без технічних деталей реалізації, таких як код чи інструменти. Інформація синтезована з авторитетних джерел, включаючи *Cloudflare, Imperva* та українські ресурси, як *HostZealot* та *NWU*.

DDoS-атаки поділяються на кілька категорій залежно від цілі та методу перевантаження. Основні класи:

1. Волюметричні атаки (*Volumetric Attacks*) - ці атаки спрямовані на насичення пропускної здатності мережі жертви великим обсягом трафіку. Механізм полягає в генерації масового потоку пакетів, які змушують мережеве обладнання або сервер обробляти непотрібні дані, виснажуючи *bandwidth*. Ефективність досягається через використання ботнетів – мереж скомпрометованих пристроїв, які синхронно надсилають трафік. Наприклад, атаки можуть посилюватися через рефлектори (треті сторони, що відображають трафік на жертву).

2. Атаки на рівні протоколів (*Protocol Attacks*) - експлуатують вразливості в мережевих протоколах, таких як *TCP* або *UDP*, для вичерпання ресурсів на рівні з'єднань. Механізм включає маніпуляцію пакетами, які

змушують сервер тримати стани з'єднань або генерувати відповіді, не завершуючи процес. Це призводить до переповнення таблиць станів (*state tables*) у фаєрволах або серверах.

3. Атаки на рівні додатків (*Application Layer Attacks* - найскладніші, оскільки імітують легітимний трафік. Механізм полягає в надсиланні запитів, які вимагають інтенсивної обробки на стороні сервера (наприклад, запити до баз даних або генерація динамічного контенту). Це виснажує CPU, пам'ять та дискові ресурси, роблячи сервіс повільним або недоступним.

4. Мультивекторні атаки (*Multi-Vector Attacks*) - комбінують кілька механізмів для синергетичного ефекту. Наприклад, поєднання волюметричної атаки з атакою на протоколи ускладнює виявлення та захист.

У табл. 1.1 наведені типи категорій.

Таблиця 1.1 – Типи категорій [6]

Категорія	Приклади механізмів	Ціль перевантаження	Ефективність
Волюметричні	*UDP-flood, ICMP-flood, DNS*-ампліфікація	Пропускна здатність мережі	Висока в мережах без фільтрації, вимагає великого ботнету
На рівні протоколів	*SYN-flood, ACK-flood, Smurf*-атака	Таблиці з'єднань та ресурси сервера	Ефективна проти незахищених *TCP/IP* стеків
На рівні додатків	*HTTP-flood, Slowloris, ReDoS*	Ресурси додатків (*CPU*, пам'ять)	Складна для виявлення, оскільки схожа на нормальний трафік
Мультивекторні	Комбінація *SYN + HTTP + UDP*	Всі рівні одночасно	Найвища, вимагає комплексного захисту

SYN-flood є класичним прикладом атаки на рівні протоколів, яка експлуатує триходовий *handshake TCP*. Механізм: атакуючий надсилає велику кількість *SYN*-пакетів (ініціація з'єднання) на сервер, часто з підробленими (*spoofed*) *IP*-адресами. Сервер відповідає *SYN-ACK* і резервує ресурси (порт і

пам'ять) для очікуваного *ACK* від клієнта. Оскільки *ACK* не надходить, з'єднання залишається "напіввідкритим" (*half-open*), накопичуючись у черзі. Коли черга переповнюється, сервер не може обробляти нові легітимні з'єднання. Етапи: (1) Генерація *SYN*-пакетів; (2) Відповідь сервера та резервування ресурсів; (3) Таймаут напіввідкритих з'єднань без звільнення. Ефективність полягає в низькій потребі трафіку для атаки – достатньо перевищити розмір черги *backlog* сервера. Варіанти: пряма (без *spoofing*), спуфінгова або розподілена (*DDoS* з ботнетом) [7].

UDP-flood – волюметрична атака, що використовує безз'єднанний протокол UDP. Механізм: атакуючий надсилає масу *UDP*-пакетів на випадкові або конкретні порти сервера, часто з підробленими *IP*. Сервер перевіряє, чи є додаток, що слухає порт; якщо ні, генерує *ICMP* "*Destination Unreachable*". Це виснажує ресурси на перевірку та відповіді. Етапи: (1) Надсилання *UDP*-пакетів; (2) Перевірка портів сервером; (3) Генерація *ICMP*-відповідей; (4) Переповнення ресурсів. Ефективність через простоту – *UDP* не вимагає *handshake*, а спуфінг ускладнює трасування. Варіанти: *CharGEN-flood* (використання протоколу *CharGEN* для посилення), *Fraggle*-атака (*UDP*-версія *Smurf*) [8].

HTTP-flood діє на рівні додатків, імітуючи звичайний веб-трафік. Механізм: атакуючий надсилає велику кількість *HTTP*-запитів (*GET* для статичних ресурсів, *POST* для динамічних форм), які сервер мусить обробляти. Кожен запит вимагає ресурсів на парсинг, доступ до баз даних тощо. Етапи: (1) Генерація *HTTP*-запитів з ботнету; (2) Обробка сервером (виконання скриптів, генерація відповідей); (3) Вичерпання *CPU*/пам'яті. Ефективність у скритності – запити виглядають легітимними. Варіанти: *Low-and-Slow* (повільне надсилання фрагментів), рекурсивний *GET* (повторні запити на один ресурс), фрагментований *HTTP*. Підтипи включають атаки на один сеанс або з фрагментацією [9].

Інші механізми *DDoS*-атак включають різноманітні техніки, спрямовані на виснаження ресурсів мережі, серверів або додатків. Серед них — *ICMP-flood* (*Ping Flood*), який полягає у надсиланні великої кількості *ICMP Echo Request*-

пакетів, змушуючи цільовий сервер генерувати відповіді *Echo Reply* і, таким чином, перевантажуючи мережевий канал; цей тип атаки найефективніший у простих мережах без належного фільтрування [10].

Smurf-атака використовує підроблені *ICMP*-запити, надіслані на *broadcast*-адреси підмережі, що призводить до того, що всі пристрої в цій підмережі відправляють відповіді на *IP*-адресу жертви, створюючи «шторм» трафіку завдяки механізму підсилення через мережеві рефлектори [10].

DNS-ампліфікація базується на надсиланні малих *DNS*-запитів із підробленою *IP*-адресою жертви до відкритих рекурсивних *DNS*-серверів, які відповідають великими пакетами даних, спрямованими на ціль, що дозволяє посилити вихідний трафік у десятки разів [10].

Slowloris є аплікаційною атакою, яка утримує якомога більше *HTTP*-з'єднань у відкритому стані, надсилаючи фрагменти запитів з великими інтервалами, що блокує слоти підключення на веб-сервері та робить його недоступним для легітимних користувачів [10].

Ping of Death використовує надсилання фрагментованих *IP*-пакетів, сума розмірів яких перевищує максимально допустимий розмір у 65 535 байтів, що може призвести до переповнення буфера та збою системи [10].

ReDoS (Regular Expression Denial of Service) — це атака на рівні додатків, при якій зловмисник надсилає спеціально сформовані рядки, що викликають експоненційне зростання часу обробки регулярних виразів у вразливому коді, призводячи до повного виснаження CPU [10].

Приховані атаки (*Hidden Attacks*) характеризуються низькою інтенсивністю та пульсуючим характером, імітуючи легітимний трафік; вони можуть викликати таймаути в *TCP*-стеку, поступово деградуючи якість обслуговування без явних ознак атаки [10].

Нарешті, експлойтні атаки використовують конкретні вразливості в операційних системах або протоколах, такі як *WinNuke* (надсилання out-of-band даних через *TCP*, що викликає збої в старих версіях *Windows*) або *Teardrop*

(надсилання *IP*-пакетів із перекритими фрагментами, що призводить до збоїв при їх зборці) [10].

Сучасні *DDoS*-атаки еволюціонують до мультивекторних і нульового дня (*zero-day*), використовуючи *IoT*-пристрої для ботнетів (табл. 1.2). Ефективність залежить від розміру ботнету, spoofing *IP* та комбінації векторів. Наприклад, атаки на кореневі *DNS*-сервери у 2000-х роках досягали 10 Гб/с. Дослідження показують зростання прихованих атак, інтегрованих зі спамом. Захист включає фільтрацію трафіку, але механізми атак адаптуються.

Таблиця 1.2 – Типи атак

Тип атаки	Механізм перевантаження	Варіанти	Джерела вразливості
SYN-flood	Напіввідкриті з'єднання	Пряма, спуфінгова	*TCP* стек
UDP-flood	Перевірка портів та *ICMP*	*CharGEN, Fraggle*	*UDP* протокол
HTTP-flood	Обробка запитів	*Low-and-Slow*, рекурсивний	Веб-додатки
ICMP-flood	*Exo*-відповіді	*Smurf, Ping of Death*	*ICMP* протокол
DNS-ампліфікація	Посилені відповіді	*NTP, SSDP*	Відкриті сервери

Цей огляд охоплює основні механізми, але реальні атаки можуть комбінуватися. Для глибшого вивчення рекомендується звертатися до спеціалізованих ресурсів з кібербезпеки.

1.3 Вплив *DDoS*-атак на цілісність, конфіденційність та доступність інформації

DDoS-атаки є однією з найпоширеніших і найдавніших форм кіберзагроз, однак їхня ефективність і складність продовжують зростати, особливо на тлі масового поширення інтернету речей (*IoT*) та створення потужних ботнетів.

Основна мета таких атак - усвідомлене перевантаження мережевих ресурсів або серверів із метою робити їх недоступними для легітимних користувачів. В аналізі цих загроз ключове значення має тріада *CIA* - три фундаментальні принципи інформаційної безпеки: конфіденційність (*Confidentiality*), цілісність (*Integrity*) та доступність (*Availability*). За даними авторитетних джерел, зокрема рекомендацій Агентства кібербезпеки та інфраструктурної безпеки США (*CISA*) та академічних досліджень, *DDoS*-атаки безпосередньо порушують лише один із цих компонентів - доступність, тоді як вплив на конфіденційність і цілісність є, як правило, опосередкованим, ситуативним або мінімальним [11].

Тріада *CIA* формує основу будь-якої стратегії інформаційної безпеки. Конфіденційність забезпечує, що інформація залишається доступною виключно авторизованим особам. Цілісність гарантує, що дані не були змінені, пошкоджені чи підмінені без належного дозволу. Доступність, у свою чергу, означає, що інформаційні ресурси та системи готові до використання в будь-який необхідний момент. *DDoS*-атаки специфічні тим, що вони не намагаються вкрасти, змінити чи розшифрувати дані - їхній єдиний інструмент - надмірний трафік. Атаки класифікуються за рівнями моделі *OSI*: волюметричні (направлені на насичення каналу зв’язку величезним обсягом трафіку, наприклад, через *UDP* чи *ICMP*), протокольні (що експлуатують уразливості у процесі встановлення з’єднання, як у випадку з *SYN-flood*), та атаки на рівні додатків (які імітують поведінку реальних користувачів, наприклад, надсилаючи велику кількість *HTTP*-запитів). Усі ці механізми напряму цілять доступність, але в деяких випадках можуть створювати благоприятні умови для інших, більш глибоких вторгнень [12].

Доступність є основною і найбільш уразливою складовою під час *DDoS*-атак. Головна мета зловмисників - повністю або частково блокувати доступ до мережевих сервісів. За даними *CISA*, наслідками таких атак можуть бути втрата критичних служб, значне зниження продуктивності систем, істотні фінансові витрати на відновлення роботи та серйозна репутаційна шкода. У випадку волюметричних атак, найпоширеніших у 2024–2025 роках, ботнети, що складаються з мільйонів скомпрометованих *IoT*-пристроїв, генерують трафік

інтенсивністю в сотні гігабіт або навіть терабіт на секунду, що повністю насичує мережевий канал і блокує будь-які легітимні запити. Дослідження компанії *Akamai* показують, що такі атаки часто тривають годинами, а іноді й кілька днів, що призводить до реальних фінансових втрат - для банків, платіжних систем чи платформ електронної комерції простій може обійтися в мільйони доларів за годину. Особливо небезпечними є мультивекторні атаки, які одночасно використовують різні методи - наприклад, поєднують *PSH-ACK, SYN, UDP*-флуди та *DNS*-рефлексію. Це ускладнює роботу систем захисту, оскільки кожен вектор вимагає окремих механізмів відсікання. У програмно-конфігурованих мережах (*SDN*), де центральний контролер управляє усією інфраструктурою, *DDoS*-атаки можуть швидко виснажити його ресурси, призводячи до повного мережевого колапсу [13].

На відміну від доступності, цілісність даних під час *DDoS*-атак, як правило, не порушується безпосередньо. *CISA* прямо вказує, що «*DDoS*-атака навряд чи вплине на цілісність системи чи даних», оскільки такі атаки не передбачають жодних маніпуляцій із вмістом файлів, баз даних або конфігурацій. Однак існують серйозні опосередковані ризики. Наприклад, *DDoS* часто використовується як відволікаючий маневр: поки команда безпеки зайнята боротьбою з перевантаженням мережі, зловмисники можуть проводити інші операції - впроваджувати шкідливий код, встановлювати бекдори або запускати програми-вимагачі, які вже безпосередньо змінюють або шифрують дані. У контексті *SDN* перевантаження контролера або серверів може призводити до нестабільної роботи систем, що, у свою чергу, спричиняє випадкову корупцію даних або неконтрольовані зміни у конфігураціях. Українські дослідження, зокрема роботи Вінницького національного технічного університету (ВНТУ), не фіксують прямих порушень цілісності через *DDoS*, але зазначають, що загальна дестабілізація системи може викликати непередбачувані помилки, що опосередковано впливають на надійність даних. Компанія *BitSight* також підкреслює, що цілісність стає уразливою лише в рамках комбінованих атак, де DDoS є лише першим етапом. Найбільший ризик у цьому плані існує у

критичних інфраструктурах - енергетичній, фінансовій або транспортній сферах, де мережевий простій може спричинити каскадні збої, що, своєю чергою, збільшує ймовірність помилок у даних або навіть фізичного пошкодження обладнання [14].

Щодо конфіденційності, *DDoS*-атаки не є інструментом для витоку чутливої інформації. Згідно з рекомендаціями *CISA*, «*DDoS*-атака навряд чи вплине на конфіденційність системи та даних», оскільки вони не включають механізмів злому шифрування, крадіжки облікових даних або несанкціонованого доступу до баз. Атака просто робить сервіс недоступним - вона не «дивиться» у вміст даних. Проте існують виняткові сценарії. У складних архітектурах, таких як *SDN* або хмарні середовища, сильне перевантаження може призвести до збоїв у механізмах автентифікації, шифрування або управління доступом. Наприклад, при високому навантаженні система може тимчасово відключити двофакторну автентифікацію або зменшити рівень шифрування для збереження продуктивності - що відкриває можливості для вторгнення. *Akamai* зазначає, що без доступності принципи конфіденційності та цілісності «застрягають» - користувачі не можуть отримати доступ до своїх даних, навіть якщо ті захищені. Українські науковці, зокрема в роботах ВНТУ, фокусуються переважно на аспектах доступності і не акцентують увагу на конфіденційності в контексті *DDoS*, проте загальні дослідження в галузі кібербезпеки вказують на те, що *DDoS* може використовуватися як «димова завіса» для прихованої екстракції даних (*data exfiltration*) через інші вектори. BitSight наводить приклади, де *DDoS* використовувався для відволікання уваги від шпигунського програмного забезпечення, яке вже було активним у системі. У 2025 році, зі зростанням віддаленої роботи та хмарних сервісів, такі гібридні загрози стають все більш актуальними, особливо коли *DDoS* поєднується з методами соціальної інженерії - наприклад, фішингом під час простою системи.

У підсумку, *DDoS*-атаки залишаються однією з найефективніших тактик у кіберконфліктах, оскільки вони безпосередньо порушують доступність - ключовий елемент тріади *CIA*. Хоча вони не здатні самостійно змінювати дані

або красти інформацію, їхнє справжнє небезпека часто полягає в тому, що вони створюють вікна можливостей для інших, набагато серйозніших загроз. Тому сучасні системи захисту повинні не лише виявляти й блокувати надмірний трафік, а й забезпечувати комплексний моніторинг на предмет вторинних атак, підтримувати цілісність та конфіденційність навіть у умовах високого навантаження, а також мати можливість швидкого відновлення роботи критичних сервісів.

1.4 Сучасні підходи до захисту від *DDoS*-атак

Атаки розподіленого заперечення обслуговування (*DDoS*) стали невід'ємною частиною сучасної кібервійни, особливо в контексті російсько-українського конфлікту, де вони використовуються як інструмент для дестабілізації, психологічних операцій та економічного тиску. Станом на грудень 2025 року підхід України до захисту від *DDoS*-атак поєднує еволюцію національної політики, технологічні інновації, міжнародну співпрацю та громадські ініціативи. Ця комплексна стратегія сформувалася під тиском тривалих кібератак і поступово перейшла від реактивних заходів до проактивної моделі, заснованої на кіберстійкості. Хоча жодна система не є абсолютно надійною, українські механізми захисту демонструють високу адаптивність у умовах постійної загрози, надаючи цінний досвід для світової кібербезпеки [17].

Фундаментальна зміна в українській філософії захисту від *DDoS*-атак полягає в переході від ідеї «повного захисту» до концепції «кіберстійкості». Цей підхід визнає, що абсолютне запобігання атакам неможливе перед обличчям складних, державно підтримуваних агресорів. Замість цього основна увага зосереджена на мінімізації часу простою, швидкому відновленні та підтриманні безперервності роботи критичних систем. Аналіз Естонської академії електронного управління підкреслює цей стратегічний зсув України на тлі тривалої кіберагресії. До ключових елементів такої стратегії належать чітке планування реагування на інциденти, наявність резервних систем та постійний

моніторинг, що дозволяє системам витримувати навіть потужні атаки, які могли б повністю паралізувати менш підготовлені країни [18].

На рівні політики Стратегія кібербезпеки України (2021–2025), оновлена на початку 2025 року, інтегрує заходи з протидії *DDoS*-атакам у ширші рамки національної безпеки. Документ узгоджений з принципами стандарту *ISO/IEC* 27001 і передбачає захист критичної інформаційної інфраструктури, а також розвиток кіберсил, здатних до оборони й активних дій. Національний координаційний центр з кібербезпеки (НКЦКБ), підпорядкований Державній службі спецзв'язку та захисту інформації (ДССЗІ), відповідає за реалізацію цієї стратегії. У 2025 році Україна також запозичила елементи Рамки кібербезпеки *NIST*, зосереджуючись на п'яти функціях: ідентифікація, захист, виявлення, реагування та відновлення. Зокрема, для критичних секторів - енергетики, фінансів і транспорту - запроваджено обов'язкові аудити. Саме ці сектори станом на другий квартал 2025 року поглинули 63% усіх глобальних *DDoS*-атак на тлі загострення геополітичної напруженості [19].

З технологічної точки зору Україна використовує багатошарову архітектуру захисту. Центральне місце в ній займають хмарні сервіси з пом'якшення *DDoS*-атак. Наприкінці 2024 року, з продовженням у 2025, компанія *Akamai* була обрана для реалізації національної програми захисту від *DDoS*. Ці сервіси ефективно нейтралізують об'ємні атаки - такі як *UDP*- та *SYN*-флуди - шляхом фільтрації трафіку на периферії мережі, відсікаючи шкідливі пакети і пропускаючи легітимні. Рішення компанії *Radware*, інтегровані Державним центром кіберзахисту, використовують поведінковий аналіз і машинне навчання для виявлення аномалій. Щодо атак на рівні додатків (*L7*), застосовуються веб-фаєрволи (*WAF*), які фільтрують *HTTP*-запити, а *BGP FlowSpec* дозволяє динамічно змінювати маршрутизацію для «нуль-рутування» джерел атак [20].

Особливу увагу також приділяють новим загрозам, зокрема зондувальним атакам, які у 2025 році зросли у 5 000 разів порівняно з попереднім роком: це низькоінтенсивний трафік, спрямований на перевірку захисту перед масштабним

наступом. У відповідь Україна активно розвиває системи на основі штучного інтелекту для ідентифікації цілей та механізми швидкого реагування. Хоча деякі згадані технології - наприклад, ретранслятори сигналів чи дрони на основі волоконно-оптичного керування - більше пов'язані з гібридною війною, у чисто кіберпросторі основний акцент залишається на аналізі трафіку, шифруванні та швидкій адаптації до нових тактик [21].

Міжнародне співробітництво значно посилює ці зусилля. Американські агентства, зокрема *CISA*, надають консультації щодо тактик російських хакерів, зокрема використання *DDoS* як прикриття для глибших вторгнень. Фінансування від *USAID* підтримує аналітичні організації, такі як агентство *Molfar*, хоча це іноді викликає дискусії щодо можливої цензури. ЄС і НАТО активно обмінюються розвідувальними даними про загрози, що дозволяє вживати превентивних заходів. Наприклад, після атак на медіа, зокрема на видання *The Kyiv Independent* у липні 2025 року, динамічне фільтрування та хмарні провайдери швидко відновили роботу ресурсів [22; 24].

Особливу роль у кібероборoні України грають добровольчі й хактивістські ініціативи. *IT*-армія України, що налічує близько 300 тисяч учасників, проводить контратаки проти російських цілей, використовуючи інструменти на кшталт disBalancer. Однак експерти попереджають про ризики: витік особистих даних, непередбачувані наслідки та можливість реталіації. Групи на кшталт *IT* Army 2022 демонструють високу синхронізацію в атаках на російські ЗМІ, урядові та фінансові ресурси. Проте дослідження вказують, що 77,90% атакованих доменів (як українських, так і російських) не мали достатнього захисту від *DDoS* - ні до, ні після атак, що вказує на серйозні прогалини в реактивних підходах [23].

Події 2025 року добре ілюструють як ефективність, так і обмеження української стратегії. У липні відбулися три масштабні *DDoS*-кампанії - проти інфраструктури Криму, українських медіа та російських компаній - що продемонструвало тактику синхронізованих атак під час політично чутливих подій. У вересні Україна, через Головне управління розвідки (ГУР), заявляла про атаку на російські системи голосування в окупованих територіях,

використовуючи *DDoS* для дестабілізації виборів. Водночас про-російські групи, як-от *NoName057*(16) і *KillNet*, продовжували атакувати українські ресурси, але завдяки таким механізмам, як *null-routing* і нейромережевий аналіз, їхній вплив було значно зменшено [24].

1.5 Огляд існуючих алгоритмів та систем виявлення *DDoS*-атак

Атаки розподіленого заперечення обслуговування, відомі як *DDoS*, є однією з найсерйозніших загроз для стабільності та безпеки сучасних мереж. Їх виявлення вимагає застосування складних, адаптивних і багатошарових підходів, які здатні не лише швидко реагувати на відомі шаблони, але й передбачати зовсім нові, раніше невідомі форми кібератак. Традиційні методи, що базуються на сигнатурах, досі використовуються в багатьох системах інформаційної безпеки. Наприклад, популярні інструменти на кшталт *Snort* та Suricata порівнюють мережевий трафік із базою відомих шаблонів атак, що дозволяє ефективно ідентифікувати класичні види *DDoS*, такі як *TCP SYN flood* або *UDP flood*. Проте такий підхід має істотний недолік - він абсолютно безпорадний перед модифікованими або принципово новими атаками, оскільки вимагає постійного оновлення бази сигнатур, що в умовах швидкозмінного кіберпростору часто виявляється занадто повільним [25].

Альтернативою сигнатурним методам є аномальні підходи, які не спираються на попереднє знання про конкретні атаки, а натомість аналізують відхилення від звичайного поведінкового профілю мережі. Такі системи можуть використовувати різноманітні статистичні метрики - наприклад, ентропію джерел *IP*-адрес, середню швидкість надходження пакетів або складні моделі часових рядів. Коли трафік раптово змінюється - наприклад, з’являється надзвичайно велика кількість запитів із нових *IP*-адрес - система фіксує це як потенційну аномалію. Однак такий підхід також не є ідеальним: він часто генерує надмірну кількість хибних спрацьовувань, особливо коли нормальні коливання

трафіку - наприклад, під час випуску нової онлайн-гри чи масового виходу користувачів у мережу - інтерпретуються як атака [26].

У сучасних системах виявлення *DDoS* все більшого поширення набувають методи, засновані на машинному та глибокому навчанні. Серед методів супервізійного навчання особливою популярністю користуються векторні машини підтримки, випадкові ліси та дерева рішень. Ці алгоритми навчаються на попередньо розмічених наборах даних, де кожен запис класифікований як «нормальний» або «зловмисний». Завдяки цьому вони здатні досягати дуже високої точності - часто понад дев'яносто дев'ять відсотків - при аналізі таких ознак, як номери портів, обсяг переданих байтів, тривалість потоку чи частота запитів. Наприклад, модель на основі випадкового лісу добре себе показує на таких стандартних датасетах, як *CICIDS2017*, ефективно відрізняючи легітимний трафік від атак [27].

Коли мова йде про виявлення невідомих атак, які ще не увійшли до навчальних наборів, перевагу мають методи несупервізійного навчання. Серед них - автоенкодери, які навчаються відтворювати нормальні дані, і будь-які суттєві відхилення від цього відтворення вважаються аномаліями, а також ізоляційні ліси, які визначають аномалії на основі глибини поділу випадкових дерев. Ці підходи особливо корисні для виявлення *zero-day* атак, коли наперед невідомо, як вони виглядатимуть. Глибоке навчання, зокрема з використанням згорткових нейронних мереж, довготривалих рекурентних мереж *LSTM* чи *GRU*, дозволяє моделям ефективно працювати з часовими залежностями в мережевому трафіку. Наприклад, гібридні архітектури на основі *CNN-LSTM* показали виняткову ефективність у виявленні повільних атак типу *Slowloris* у програмно-конфігурованих мережах, досягаючи показників F1-score понад 0.99, що свідчить про майже повне уникнення як хибних позитивів, так і пропущених атак [28].

Ще одним перспективним напрямком є використання генеративних адверсаріальних мереж. Вони дозволяють створювати реалістичні синтетичні приклади атак, які використовуються для тренування моделей, щоб підвищити їхній імунітет до спроб обману. Дослідження показали, що такі підходи можуть

підвищувати загальну стійкість систем до дев'яноста п'яти відсотків у контрольованих тестах. Це особливо важливо в умовах, коли агресори намагаються змінювати свої тактики, щоб уникнути виявлення.

У сучасних архітектурах мереж, зокрема в програмно-конфігурованих мережах та інтернеті речей, системи виявлення *DDoS* тісно інтегруються з мережевими контролерами. Це дозволяє не просто фіксувати атаку, а й миттєво реагувати на неї - наприклад, через протокол *OpenFlow* змінювати правила маршрутизації або обмежувати потік трафіку з підозрілих джерел. У розподілених середовищах, як-от *IoT*, де важливо зберігати конфіденційність даних, все більш популярним стає федеративне навчання, що дозволяє тренувати моделі без передачі даних до центрального сервера. У програмно-конфігурованих мережах також використовуються алгоритми на основі графів, наприклад, ядра Вайсфайлер-Лемана, які ефективно виявляють складні атаки на DNS, аналізуючи структуру зв'язків між мережевими вузлами [29].

Комерційні рішення, такі як *Akamai App & API Protector*, *Radware DefensePro* або *Imperva SecureSphere*, поєднують різні згадані алгоритми з потужними хмарними інфраструктурами, що дозволяє здійснювати скрубінг трафіку навіть при атаках інтенсивністю в декілька терабіт на секунду. У 2025 році деякі вендори вже пропонують інструменти на кшталт *Flood Shield* 2.0, які використовують штучний інтелект для адаптивного аналізу трафіку в реальному часі та інтегрують дані про загрози з глобальних інтелектуальних платформ. У Україні науковці активно досліджують можливості застосування глибоких нейронних мереж для аналізу мережевого трафіку, зокрема згорткових архітектур для детекції шкідливих пакетів, а також інтеграцію *SDN*-рішень з хмарними платформами, такими як *AWS*, та інструментами автоматизації на кшталт *Terraform* [30].

Незважаючи на значні досягнення в цій галузі, існуючі методи мають певні обмеження. Глибокі нейронні мережі потребують великих обчислювальних ресурсів і часто залежать від наявності якісних, великих наборів даних, таких як *CICDDoS2019* або *Bot-IoT*. Крім того, багато систем залишаються вразливими до

технік маскування, зокрема *IP*-спуфінгу, коли атакуючі підмінюють джерела трафіку. Дослідження свідчать, що приблизно двадцять шість відсотків успішних *DDoS*-атак призводять до втрати даних або важливих послуг, тому швидкість виявлення - у межах мілісекунд або секунд - стає критично важливою.

Серед майбутніх напрямків розвитку - застосування квантових автокодерів для виявлення аномалій, а також використання великих мовних моделей для автоматичної генерації правил фаєрволів на основі аналізу кіберзагроз. Українські науковці, зокрема з Вінницького національного технічного університету, фокусуються на розробці хмарно-орієнтованих рішень з автоматизацією за допомогою *Ansible* та *Packer*, що дозволяє швидко розгортати захищені інфраструктури. У цілому, ефективність будь-якої системи виявлення *DDoS* залежить не лише від складності алгоритмів, а й від того, наскільки добре вона інтегрована в загальну мережеву архітектуру, і як швидко вона може не лише виявити загрозу, але й забезпечити неперервність роботи критичних служб [31].

2 АНАЛІЗ ТА ВИБІР АЛГОРИТМІВ ВИЯВЛЕННЯ DDOS

2.1 Критерії оцінки ефективності алгоритмів виявлення

Критерії оцінки ефективності алгоритмів виявлення *DDoS*-атак є центральним елементом у розробці та впровадженні надійних систем кібербезпеки. В умовах постійної еволюції кіберзагроз - зокрема, з розширенням мереж інтернету речей (*IoT*), хмарних сервісів та програмно-конфігурованих мереж (*SDN*) - ці критерії дозволяють не лише порівнювати різні підходи, а й приймати обґрунтовані рішення щодо вибору найбільш відповідних рішень для конкретних умов. Оцінка ґрунтується на трьох ключових групах показників: метриках класифікації, часових характеристиках і ресурсних вимогах. Всі вони аналізуються на основі стандартизованих наборів даних, таких як *CICIDS2017*, *CICDDoS2019*, *NSL-KD*D або *BoT-IoT*, які містять як нормальні, так і зловмисні зразки трафіку. Особливо важливо враховувати дисбаланс класів: у реальних мережах доля атак складає лише кілька відсотків від загального обсягу трафіку, що робить простою «точність» (*accuracy*) потенційно оманливою метрикою [32].

Основою кількісної оцінки є матриця плутанини, яка визначає чотири ключові компоненти: правильно виявлені атаки (*true positives*, *TP*), правильно ідентифікований легітимний трафік (*true negatives*, *TN*), помилково позначений як атака легітимний трафік (*false positives, FP*), і пропущені атаки (*false negatives, FN*). На цій основі розраховуються основні метрики якості. Точність (*accuracy*) - це загальна частка правильних передбачень у всіх випадках. Наприклад, у дослідженні з використанням алгоритму *Random Forest* на датасеті *CICIDS2017* було досягнуто точності 98,9%. Однак такий високий показник може бути зумовлений тим, що більшість трафіку є нормальним, і система просто «вгадує» більшість випадків, не виявляючи рідкісні атаки. Тому точність сама по собі є недостатньою для оцінки ефективності у контексті *DDoS* [33].

Набагато важливішими є такі метрики, як прецизія (*precision*) і повнота (*recall*). Прецизія визначає, наскільки надійні позитивні передбачення: вона

показує, яка частка всіх «підозрюваних» подій дійсно є атаками. Це критично для запобігання помилковим блокуванням легітимного трафіку, що може призвести до скарг користувачів, фінансових збитків або навіть порушення *SLA*. Повнота, навпаки, вимірює, яка частина усіх реальних атак була виявлена системою. Низька повнота означає, що багато атак залишаються непоміченими, що може призвести до повного виведення сервісу з ладу. Для балансування цих двох показників використовується *F1*-score - гармонійне середнє між прецизією та повнотою. У незбалансованих сценаріях саме *F1*-score вважається найбільш об'єктивною метрикою. Наприклад, у дослідженнях гібридних моделей на основі *CNN-LSTM* на датасеті *CICDDoS2019* F1-score досягає 0,99, що свідчить про виняткову здатність системи одночасно мінімізувати як пропуски, так і хибні спрацьовування [34].

До додаткових метрик належать рівень помилкових позитивів (*FPR*) і помилкових негативів (*FNR*), а також специфічність (TNR або true negative rate), яка показує, наскільки добре система розпізнає нормальний трафік. Крім того, широко використовується *AUC-ROC* (площа під *ROC*-кривою), що відображає здатність моделі розрізняти атаки та нормальний трафік незалежно від обраного порогу класифікації. Ідеальна модель має *AUC* = 1; у практиці ж значення понад 0,98 вважаються дуже високими. Так, глибокі нейронні мережі (*DNN*) на датасеті *CICIDS2017* демонструють *AUC* 0,987. Також окремо оцінюється detection rate (*DR*) - частка виявлених атак від загальної кількості. У деяких моделях на основі автоенкодерів (*AE*) *DR* досягає 100% на синтетичних даних, хоча це не завжди переноситься на реальні умови [35].

Часові та ресурсні характеристики є не менш важливими, особливо для систем, що працюють у реальному часі. Час виявлення - або час передбачення на один пакет чи потік - має бути мінімальним. У *IoT*-середовищах, де пристрої мають обмежені обчислювальні ресурси, це критично: наприклад, алгоритм *KNN* може вимагати 73,1 секунди для обробки 100 передбачень, тоді як *Random Forest* - лише 2,87 секунди. Для високонавантажених систем, де кожна мілісекунда має значення, така різниця може визначати, чи буде атака зупинена до того, як сервіс

вийде з ладу. Обчислювальна складність також враховує витрати *CPU*, *GPU*, пам'яті та пропускної здатності мережі. Глибокі моделі, як *LSTM* або *CNN*, хоча й надають високу точність, потребують значних ресурсів, що робить їх непридатними для багатьох *edge*-пристроїв. Масштабованість системи перевіряється на великих обсягах трафіку - наприклад, при атаках інтенсивністю в терабіти на секунду - і враховує здатність адаптуватися до динамічних умов у *SDN* або *IoT*. Стійкість (*robustness*) ж оцінюється шляхом тестування на *zero-day*-атаках та адверсаріальних прикладах, коли зловмисники свідомо змінюють трафік, щоб уникнути виявлення. У таких сценаріях гібридні моделі, що поєднують різні підходи, часто демонструють найкращу стійкість [36].

У складних ситуаціях, коли потрібно врахувати одночасно кілька критеріїв, застосовуються багатокритеріальні методи прийняття рішень, такі як *ELECTRE III*. Вони дозволяють ранжувати алгоритми з урахуванням ваги кожного показника. Наприклад, якщо для певної інфраструктури критичним є час відгуку (вага 0,8), тоді швидкісні моделі отримують перевагу, навіть якщо їхня точність трохи нижча. У українських наукових дослідженнях, зокрема в дисертаційних роботах, окрема увага приділяється швидкості кластеризації (наприклад, застосування *k-means* до великих обсягів даних), а також ймовірності блокування трафіку (*BLK probability*), яка враховує інтенсивність прибуття та обслуговування трафіку (параметри λ та μ) і пов'язані з цим помилки першого та другого роду (*E1* та *E2*). Також пропонуються методи, що враховують сезонність мережевого трафіку і автоматично налаштовують динамічні пороги виявлення на основі стандартного відхилення (σ), що дозволяє виявляти аномалії в чотири рази швидше за традиційні підходи [37].

Дослідження різних алгоритмів на реальних датасетах дають чітку картину їхніх сильних і слабких сторін. Наприклад, *Random Forest* на *CICIDS2017* демонструє точність 98,9%, прецизію 99,4%, повноту 94,7% і *F1*-score 97%, при цьому вимагаючи лише 2,87 секунди на 100 передбачень. *CNN-LSTM* на *CICDDoS2019* досягає ще вищих показників: точність 99,03%, *F1*-score 99,36%, але вимагає більше часу та ресурсів. Алгоритм *KNN*, хоча й показує дуже високу

повноту (99,8%) на *BoT-IoT*, страждає від низької швидкості і вищого *FPR*. *DNN*-моделі на *UNSWNB15* демонструють майже ідеальну повноту (1,0), але знижений *F1*-score через низьку прецизію [38].

У різних середовищах акценти змінюються. У *IoT* важливо мінімізувати *FPR*, щоб не перевантажувати обмежені пристрої помилковими попередженнями. У SDN-мережах ключовим є швидка інтеграція з контролерами для динамічного реагування - наприклад, миттєве оновлення правил *OpenFlow*. У критичній інфраструктурі (енергетика, фінанси) пріоритетом є мінімізація *FNR* - тобто, виявлення усіх можливих атак навіть за рахунок більшої кількості хибних спрацьовувань. Для комерційних сервісів важливіший баланс між *F1*-score, витратами на обчислювальні ресурси та втратами від простою [39].

Майбутні напрямки досліджень фокусуються на протидії адверсаріальним атакам за допомогою генеративних адверсаріальних мереж (*GAN*), застосуванні квантових автоенкодерів для виявлення аномалій, а також розвитку механізмів онлайн-навчання, які дозволяють моделям адаптуватися до нових загроз у реальному часі. У українському контексті активно розвиваються методи, що враховують часові патерни трафіку, динамічні пороги виявлення та інтеграцію з хмарними платформами [40].

Загалом, жоден алгоритм не є універсально кращим. Ефективність системи виявлення *DDoS*-атак визначається її здатністю відповідати конкретним вимогам середовища: рівню загроз, доступним ресурсам, допустимим затримкам і критичності послуг. Тому рекомендується проводити комплексну оцінку з використанням кількох наборів даних, реальних тестів у живій мережі та багатокритеріального аналізу, щоб забезпечити надійність, швидкість і економічну доцільність рішення.

2.2 Порівняльний аналіз методів: сигнатурний, аномалійний, гібридний

Методи виявлення розподілених атак заперечення обслуговування (*DDoS*) є ключовим компонентом сучасної кібербезпеки, оскільки саме від їхньої ефективності часто залежить, чи зможе організація утримати доступність своїх сервісів під час масштабного кібернападу. За останні п’ять років (2020–2025) сформувалися три основні підходи до виявлення таких атак: сигнатурний, аномальний та гібридний. Кожен із них має власні механізми роботи, переваги й обмеження, а їхня ефективність сильно залежить від контексту мережі - наприклад, чи це стабільна корпоративна інфраструктура, динамічна *IoT*-мережа чи хмарне середовище з програмною конфігурацією (*SDN*). Жоден з цих підходів не є універсальним рішенням; натомість, вибір методу має ґрунтуватися на специфіці загроз, доступних обчислювальних ресурсах та рівні складності атак [41].

Сигнатурний підхід до виявлення *DDoS* базується на порівнянні вхідного мережевого трафіку з базою даних відомих шаблонів атак. Ці сигнатури - це конкретні послідовності пакетів, структури заголовків або особливості корисного навантаження, характерні для певних типів атак, таких як *UDP*-флуд, *ICMP*-спам або *HTTP GET*-флуд. Механізм роботи нагадує роботу антивірусного програмного забезпечення: система зіставляє поточні дані з попередньо визначеними правилами і, якщо знаходить збіг, блокує трафік. Такий підхід відзначається високою точністю (до 94%) у виявленні відомих загроз, дуже низьким рівнем хибних спрацьовувань (близько 3%) і високою продуктивністю у реальному часі завдяки мінімальним обчислювальним витратам. Він ідеально підходить для мереж із передбачуваним трафіком, таких як внутрішні корпоративні системи, де зловмисники часто використовують перевірені тактики. Однак головний недолік сигнатурних систем - їхня повна безпорадність перед новими, зміненими або поліморфними атаками. Оскільки такі системи не вміють «розпізнавати» те, що ще не було задокументовано, вони легко обминаються ботнетами, які змінюють свої сигнатури або використовують zero-

day-вразливості. Крім того, підтримка актуальної бази сигнатур вимагає постійних оновлень, що створює суттєве навантаження на команди безпеки [42].

На противагу сигнатурному підходу, аномальні методи не спираються на попереднє знання про конкретні атаки. Замість цього вони спочатку будують модель «нормального» поведінки мережі, використовуючи статистичні показники або алгоритми машинного навчання. Після цього будь-яке відхилення від цієї моделі - наприклад, різке зростання ентропії *IP*-джерел, незвичайна частота запитів до однієї служби або зміна тривалості з'єднань - інтерпретується як потенційна загроза. До популярних технік належать аналіз ентропії, кластеризація (наприклад, метод k-середніх), ізоляційні ліси (*Isolation Forest*) або автоенкодери. Такі системи особливо ефективні у виявленні нових, раніше невідомих атак, зокрема повільних *DDoS*-атак (*slowloris*, *HTTP slow POST*), які свідомо імітують поведінку легітимних користувачів. Аномальні підходи добре працюють у динамічних середовищах, таких як мережі інтернету речей (*IoT*), де трафік постійно змінюється, а нові типи пристроїв постійно приєднуються до мережі. Проте вони мають серйозні недоліки: через високу чутливість до змін у трафіку вони часто генерують хибні сповіщення - наприклад, коли легітимний сплеск активності (запуск нової онлайн-кампанії чи масовий вихід користувачів у мережу) помилково інтерпретується як атака. Крім того, навчання та підтримка таких моделей вимагають значних обчислювальних ресурсів, що робить їхнє застосування складним у ресурсно обмежених *IoT*-пристроях. Рівень хибних позитивів у таких системах може досягати 18%, що значно ускладнює їхню експлуатацію без додаткових механізмів фільтрації [43].

Гібридні методи виникли як спроба поєднати найкращі риси обох підходів: точність сигнатурного аналізу та адаптивність аномальних моделей. У такій архітектурі сигнатурні правила можуть працювати на периферії мережі, швидко відсікаючи відомі шкідливі потоки, тоді як у центральній частині мережі аномальні моделі аналізують решту трафіку на предмет нових, незвичайних патернів. Часто такі системи використовують механізми кореляції подій, системи оцінки впевненості або навіть штучний інтелект для уточнення

результатів. Наприклад, якщо сигнатурна система не знайшла загрози, але аномальна модель виявила відхилення, гібридна система може порівняти це відхилення з історичними даними, поведінкою сусідніх вузлів або зовнішніми джерелами загрозової інтелігенції, перш ніж видавати попередження. Це дозволяє досягати загальної точності на рівні 94–96%, істотно знижуючи кількість хибних сповіщень. Гібридні підходи особливо ефективні в умовах гібридної війни або для захисту критичної інфраструктури, де можуть виникати як класичні волюметричні, так і складні атаки на рівні додатків. Проте їхня реалізація є значно складнішою: вона вимагає інтеграції різних технологій, налаштування взаємодії між ними та великих, якісних наборів даних для навчання моделей. Крім того, гібридні системи можуть страждати від перенавчання (*overfitting*), особливо якщо їхні *ML*-компоненти тренуються на нестачі або зміщених даних [44].

Аналіз продуктивності цих трьох підходів, заснований на дослідженнях 2020–2025 років, показує чіткі різниці. Сигнатурні системи мають високу точність (94%) і швидку реакцію у реальному часі, але практично не здатні виявляти нові атаки (оцінка адаптивності - лише 3 із 10). Аномальні методи, навпаки, демонструють високу гнучкість (8/10) і можуть працювати в умовах постійних змін, але страждають від високого рівня хибних спрацьовувань і потребують більше обчислювальних ресурсів. Гібридні рішення часто перевершують обидва підходи: наприклад, модель *ResNet* у поєднанні з методом SMOTE демонструє точність 96,4% при низькій затримці, хоча тренування таких моделей може займати значно більше часу. У специфічних *IoT*-сценаріях гібридні архітектури на основі багатошарових перцептронів (*MLP*) досягають точності до 99,98% при збалансованому рівні повноти (recall) у 93,3%, значно перевершуючи чисті *SVM* або *k-NN* за швидкістю прогнозування [45].

Останнім часом спостерігається чіткий тренд на поширення гібридних архітектур. Серед інновацій - поєднання згорткових нейронних мереж (*CNN*) із сигнатурними правилами (2021), ансамблеві моделі з голосуванням між випадковим лісом та *SVM* (2023), а також федеративне навчання з увагою

(*Federated CNN + Attention*, 2024), що дозволяє аналізувати трафік без централізованого збору даних - важливо для захисту приватності. У програмно-конфігурованих мережах з *IoT*-інтеграцією (*SDN-IoT*) пропонуються системи, де аналіз відбувається на периферії (*edge*), а навчання моделей - у децентралізованому режимі. Наприклад, система *FLAD* (2024) досягає точності 93,8% при помірному використанні ресурсів, а *LUCID* (2022) адаптована для роботи на обмежених *edge*-пристроях із точністю 91,2% [46].

Тим не менш, залишаються істотні виклики: дисбаланс у навчальних наборах даних, вразливість до адверсаріальних атак (коли зловмисники свідомо модифікують трафік, щоб обманути модель), а також складність інтеграції різних компонентів у єдину систему. Тому експерти рекомендують постійне переобучення моделей, використання багатошарових архітектур захисту та поєднання різних підходів залежно від критичності інфраструктури [47].

Підсумовуючи, сигнатурні методи залишаються надійним фундаментом для захисту від відомих загроз, аномальні - потужним інструментом для виявлення нових, невідомих атак, а гібридні підходи уособлюють майбутнє *DDoS*-захисту, оскільки поєднують точність і гнучкість. Організації мають обирати стратегію відповідно до свого ландшафту загроз: сигнатурні системи - для низькоризикових чи стабільних середовищ, аномальні - для динамічних сценаріїв з високою мінливістю, а гібридні - для критично важливих систем, де втрата доступності може мати катастрофічні наслідки.

2.3 Застосування методів машинного навчання для виявлення *DDoS*-атак

Атаки розподіленого заперечення обслуговування (*DDoS*) залишаються однією з найбільш поширених і шкідливих загроз для сучасних мереж, оскільки вони напряму порушують ключовий принцип інформаційної безпеки - доступність. Використовуючи ботнети, скомпрометовані *IoT*-пристрої та техніки підсилення (*amplification*), зловмисники здатні генерувати трафік інтенсивністю в терабіти на секунду, повністю блокуючи доступ легітимних користувачів до

сервісів. У відповідь на цю загрозу, особливо в період 2023–2025 років, машинне навчання (*ML*) стало центральним інструментом у системах виявлення *DDoS*, поступово витісняючи традиційні, статичні сигнатурні підходи. Сучасні *ML*-моделі дозволяють не лише швидко виявляти відомі атаки, а й адаптуватися до нових, складних тактик, аналізуючи навіть найтонші аномалії в мережевому трафіку. Цей підхід особливо важливий у динамічних середовищах - програмно-конфігурованих мережах (*SDN*), масштабних *IoT*-інфраструктурах та хмарних платформах, де традиційні методи часто виявляються неефективними.

У загальному вигляді методи машинного навчання для виявлення DDoS поділяються на чотири парадигми: супервізійне, несупервізійне, напівсупервізійне та гібридне навчання. Супервізійні методи, які базуються на попередньо розмічених даних, використовують такі алгоритми, як метод опорних векторів (*SVM*), випадковий ліс (*Random Forest, RF*), k-найближчих сусідів (*KNN*), дерева рішень (*DT*) та логістичну регресію (*LR*). Вони аналізують різноманітні ознаки мережевого трафіку - розмір пакетів, тривалість потоку, ентропію *IP*-джерел, частоту запитів - і класифікують кожен потік як «нормальний» або «зловмисний». Наприклад, *SVM* ефективно будує розділюючі гіперплощини між класами, тоді як *RF*, використовуючи ансамблевий підхід, демонструє високу стійкість до перенавчання (*overfitting*) і шуму в даних. Такі моделі відзначаються високою точністю (до 98–99%) і швидкістю роботи, що робить їх ідеальними для стабільних корпоративних мереж [48].

Несупервізійні методи, навпаки, не потребують розмічених даних і зосереджені на виявленні відхилень від «норми». До них належать кластеризація (наприклад, *k-means*), ізоляційні ліси та автоенкодери (*AE*). Ці підходи особливо ефективні для виявлення *zero-day*-атак, коли патерни ще невідомі. Наприклад, автоенкодер навчається відтворювати нормальні дані з мінімальною помилкою, і будь-яке значне відхилення вважається аномалією. Проте такі моделі часто страждають від високого рівня хибних спрацьовувань, особливо під час легітимних сплесків трафіку. Напівсупервізійні методи поєднують переваги обох підходів: вони використовують велику кількість нерозмічених даних для

виділення ознак (наприклад, шляхом зниження розмірності за допомогою автоенкодера), а потім застосовують супервізійні класифікатори на невеликому обсязі розмічених прикладів.

Найбільш перспективними в останні роки виявилися гібридні підходи, що інтегрують різні техніки для подолання обмежень окремих моделей. Наприклад, гібрид *CNN-LSTM* аналізує трафік одночасно з просторової (через згорткові шари, які обробляють потік як «зображення») і часової (через рекурентні шари, які вловлюють послідовності) точок зору. Такі архітектури демонструють *F1*-показник понад 0,99 на датасеті *CICDDoS2019*. Інші гібридні рішення використовують генеративні адверсаріальні мережі (*GAN*) для створення реалістичних синтетичних атак і тренування стійких моделей, або поєднують автоенкодери з однокласовими *SVM* для виявлення відбиткових (*reflection*) атак у *SDN*. Також набувають поширення методи федеративного навчання, які дозволяють тренувати моделі на розподілених пристроях без централізованого збору даних - це особливо важливо для захисту приватності в *IoT*.

Глибоке навчання (*DL*), як підмножина *ML*, здатне обробляти складні, багатовимірні дані без ручного виділення ознак. Зокрема, глибокі нейронні мережі (*DNN*) та згорткові мережі (*CNN*) добре справляються з класифікацією потоків, тоді як рекурентні архітектури - *RNN*, *LSTM*, *GRU* - ефективно моделюють часові залежності, що критично для виявлення атак типу *SYN-flood*, де важливі послідовності пакетів. Наприклад, двонаправлені *GRU* з механізмами уваги (*attention*) фокусуються на найбільш інформативних частинах протокольних повідомлень, що підвищує точність виявлення. У *IoT*-середовищах розробляються легковагові *DL*-фреймворки, які зберігають ефективність за умови обмежених ресурсів пристроїв. Проте головним недоліком *DL* є висока обчислювальна складність - час інференсу (відповіді моделі) для *LSTM* може складати 5–10 секунд на сотню передбачень, що недостатньо для додатків у реальному часі.

Ключову роль у розробці та оцінці *ML*-моделей відіграють стандартизовані датасети. Серед найбільш популярних - *CICIDS2017* (51,1 ГБ необроблених

даних, містить атаки *Hulk, Slowloris*), *CICDDoS2019* (21,34 ГБ, з *NTP*- та *DNS*-підсиленням), CICIDS2018 (з *Heartbleed* та вторгненнями), а також *IoT*-орієнтовані *Bot-IoT* (69,3 ГБ, *UDP*-флуди) і *TON-IoT* (25 ГБ, *HTTP*-атаки). Старіші набори, як *KDD99* і *NSL-KDD*, хоча й використовуються, критикуються за відсутність сучасних тактик. Основні проблеми при роботі з даними - сильний дисбаланс класів (легітимний трафік переважає) і нестача рідкісних типів атак. Для їх вирішення застосовують передобробку: нормалізацію, *one-hot*-енкодінг, метод *SMOTE* для збільшення класу атак, або ж генерацію синтетичних прикладів за допомогою *GAN* [49].

Оцінка ефективності моделей ґрунтується на метриках, отриманих із матриці плутанини: точність (*accuracy*), прецизія (*precision*), повнота (*recall*), F1-*score*, *AUC-ROC*, рівень хибних позитивів (*FPR*) та швидкість виявлення. Наприклад, *Random Forest* на *CICIDS2017* досягає точності 98,9%, прецизії 99,4%, повноти 94,7% і F1-score 0,97, при цьому вимагаючи лише 2,87 секунди на 100 передбачень. *CNN-LSTM* на *CICDDoS2019* показує *F1*-score 0,9936, але працює повільніше. *LSTM* може демонструвати 100% точність у ідеальних умовах, але під адверсаріальним тиском цей показник падає до 91,75%. Це підкреслює вразливість навіть найскладніших моделей до навмисного обману.

У реальних системах *ML*-підходи демонструють як сильні, так і слабкі сторони. До переваг належать автоматичне виділення ознак, масштабованість до терабітних потоків, здатність до адаптації через переобучення та висока точність (>99% на бенчмарках). Однак існують і серйозні обмеження: висока обчислювальна складність ускладнює застосування на *edge*-пристроях, моделі погано узагальнюють на невідомі атаки, а хибні сповіщення під час пікових навантажень можуть призвести до порушення роботи легітимних сервісів. Крім того, більшість досліджень проводиться в *offline*-режимі, тоді як реальні мережі вимагають миттєвої реакції.

У контексті України *ML*-підходи інтегровані в національну стратегію кібербезпеки (2021–2025), де вони використовуються для протидії державно підтримуваним атакам. Наприклад, національні системи захисту, інтегровані з

рішеннями *Akamai*, застосовують штучний інтелект для скрубінгу трафіку в реальному часі. Також *ML*-інструменти використовуються добровольчими ініціативами, зокрема *IT*-армією України, для проведення контрзаходів. Проте залишаються ризики: дослідження показують, що приблизно 26% *DDoS*-атак призводять до втрати даних або порушення цілісності системи, особливо коли атака використовується як відволікання для глибших вторгнень [50].

Майбутнє розвитку *ML* для виявлення *DDoS* пов'язане з кількома ключовими напрямками. По-перше, це інтеграція в реальні системи у реальному часі - зокрема, у *SDN* та *IoT*, де потрібні низьколатентні рішення. По-друге, розробка стійких до адверсаріальних атак моделей за допомогою *WGAN-GP*, квантових автокодерів або онлайн-навчання. По-третє, створення більш різноманітних датасетів, що включають сучасні мультивекторні атаки, включаючи «поступове» наростання (*ramping*) трафіку. Також важливим є розвиток пояснюваного штучного інтелекту (*XAI*), який дозволить розуміти, чому чорна скринька прийняла те чи інше рішення - це критично для аудиту та відповідності нормативним вимогам. У перспективі комбінація ML з великими мовними моделями (*LLM*) може дозволити автоматично генерувати правила фаєрволів на основі аналізу загроз [51].

У підсумку, машинне навчання кардинально змінило підхід до виявлення *DDoS*-атак, надавши засоби для динамічного, адаптивного захисту в умовах постійної еволюції загроз. Проте для повноцінного впровадження необхідні міждисциплінарні зусилля - від розробки ефективних алгоритмів до вирішення етичних, інфраструктурних та організаційних питань. Лише комплексний підхід зможе забезпечити справжню кіберстійкість у цифрову епоху.

2.4 Обґрунтування вибору конкретних алгоритмів для подальшої реалізації

У сучасному цифровому середовищі розподілені атаки заперечення обслуговування продовжують активно еволюціонувати, стаючи більш складними, багатовекторними та адаптивними. Вони часто поєднують

волюметричні, протокольні та аплікаційні методи одночасно, що створює серйозні виклики для систем кібербезпеки. Це змушує розробників шукати надійні алгоритми виявлення, які здатні поєднувати високу точність, ефективність у реальному часі та здатність адаптуватися до нових форм загроз. Вибір конкретних алгоритмів для реалізації має ґрунтуватися не на теоретичних припущеннях, а на емпіричних даних, отриманих у наукових дослідженнях останніх років, зокрема в період з 2023 по 2025 рік. При цьому необхідно враховувати не лише метрики продуктивності, а й обчислювальні вимоги, масштабованість та придатність для різних типів інфраструктур – від обмежених IoT-пристроїв до хмарних платформ і програмно-конфігурованих мереж [49].

На основі систематичного аналізу сучасних досліджень для практичного застосування пропонується трикомпонентна стратегія, що охоплює три ключові підходи. Першим є *Random Forest* – алгоритм супервізійного машинного навчання, який відрізняється високою надійністю та швидкістю роботи. Другим – *Long Short*-Term *Memory*, глибока нейронна мережа, спеціалізована на аналізі часових залежностей у мережевому трафіку. Третім – гібридна архітектура *CNN-LSTM*, яка поєднує згорткові та рекурентні шари для всебічного аналізу складних патернів. Цей вибір не є випадковим: кожен з цих методів демонструє високу ефективність у вирішенні специфічних проблем, пов’язаних із *DDoS*-атаками, таких як сильний дисбаланс між нормальним і зловмисним трафіком, висока розмірність ознак, а також наявність часових патернів, які критичні для виявлення повільних або поступових атак.

Сучасні методи машинного навчання для виявлення *DDoS* можна умовно розділити на кілька груп. Супервізійні підходи, як-от *Random Forest*, *SVM* або дерева рішень, потребують наявності попередньо розмічених даних і добре справляються з відомими типами атак, але їхня ефективність залежить від актуальності тренувальних наборів. Несупервізійні методи, зокрема автоенкодери або кластеризація, не потребують розмітки і здатні виявляти раніше невідомі загрози, проте часто генерують велику кількість хибних спрацьовувань. Гібридні підходи намагаються поєднати переваги обох світів,

забезпечуючи одночасно швидкість, точність і адаптивність. У програмно-конфігурованих мережах з інтеграцією *IoT* часто використовується аналіз потоків на основі протоколу *OpenFlow*, а для зменшення обчислювального навантаження застосовуються методи відбору ознак, що дозволяють скоротити їхню кількість з 79 до 10–61 без істотної втрати інформативності [51].

Random Foriest займає особливе місце завдяки своїй ансамблевій природі. Він будується на основі великої кількості дерев рішень, кожне з яких навчається на випадковій підмножині даних, а остаточне рішення приймається шляхом усереднення або голосування. Це різко знижує ризик перенавчання та підвищує стійкість моделі. Крім того, Raindom Foirest відрізняється високою інтерпретованістю, що дозволяє аналітикам зрозуміти, які саме ознаки призвели до виявлення атаки. На популярному датасеті *CICIDS2017* цей алгоритм досягає точності 98,9 відсотка, прецизії 99,4 відсотка та повноти 94,7 відсотка, що перевершує багато інших супервізійних моделей. Особливо цінною є його здатність працювати без графічних процесорів, що робить його придатним навіть для ресурсно обмежених пристроїв на периферії мережі.

Long Short-Term *Memory* вибрано як ключовий інструмент для аналізу часових рядів у мережевому трафіку. На відміну від класичних рекурентних мереж, *LSTM* містить спеціальні механізми – комірки пам’яті – які дозволяють зберігати інформацію протягом тривалого часу, усуваючи проблему зникнення градієнта. Це робить її ідеальною для виявлення атак, які розвиваються повільно або мають складну часову структуру, наприклад, *Slowloris* або атаки з поступовим зростанням інтенсивності. Експерименти на датасеті *CICDDoS2019* показують, що *LSTM* може досягати 99 відсотків точності в ідеальних умовах, хоча під адверсаріальним тиском цей показник знижується до 91,75 відсотка. Це підкреслює необхідність застосування додаткових механізмів захисту, таких як регуляризація або тренування на синтетичних даних. Через високі вимоги до обчислювальних ресурсів *LSTiM* найкраще розгортати у хмарних середовищах, де є доступ до потужних процесорів і пам’яті [52].

Гібридна архітектура *CNN-LSTM* поєднує два потужні підходи: згорткові шари аналізують просторові ознаки мережевого трафіку – наприклад, структуру заголовків пакетів, які можна подати у вигляді «зображень», – а рекурентні шари обробляють їхню часову послідовність. Такий підхід дозволяє одночасно враховувати як миттєві особливості окремих пакетів, так і їхню динаміку у часі, що критично для виявлення багатовекторних атак. На датасеті *CICDDoS2019 CNN-LSTM* демонструє точність 99,03 відсотка, прецизію 99,26 відсотка, повноту 99,35 відсотка та *F1*-score 0,9936, що робить її однією з найефективніших моделей сьогодні. Хоча така архітектура вимагає складного налаштування гіперпараметрів і потужного обладнання, її переваги виправдовують ці зусилля для застосування у виробничих системах, де важлива максимальна надійність [50].

Порівняльний аналіз на різних датасетах підтверджує доцільність запропонованого підходу. *Random Forest* показує найкращі результати у швидкості та загальній точності на *CICIDS2017*, що робить його ідеальним для швидкого розгортання. *CNN-LSTM* перевершує інші моделі на *CICDDoS2019* при виявленні складних, багатокласових атак. На *IoT*-орієнтованому датасеті *Bot-IoT Random Forest* отримує найвищий загальний рейтинг завдяки балансу між продуктивністю та точністю. Інші алгоритми, такі як наївний байєсівський класифікатор або прості дерева рішень, не включаються в основну стратегію через недостатню ефективність у сучасних умовах [52].

При реалізації рекомендується починати з *Random Forest* для швидкого тестування та розгортання на периферійних пристроях. Потім, у разі потреби аналізувати часові залежності – наприклад, у *SDN* чи хмарних середовищах – додається *LSTM*. Нарешті, для критично важливих інфраструктур, де потрібна максимальна стійкість до складних загроз, впроваджується повноцінна гібридна система на основі *CNN-LSTM*. Обов’язковим етапом є передобробка даних – зокрема, застосування методу *SMOTE* для усунення дисбалансу класів і *PCA* для зниження розмірності. Оцінка моделей має включати не лише стандартні

метрики, а й тести в умовах, максимально наближених до реальних, наприклад, на датасеті *TON-IoT* [51].

Особливу увагу слід приділяти гібридним архітектурам, які забезпечують максимальну стійкість у умовах гібридної війни, де *DDoS*-атаки часто використовуються як відволікаючий маневр. Подальші дослідження фокусуються на створенні квантово-стійких моделей, застосуванні пояснюваного штучного інтелекту для розуміння рішень нейромереж, а також на підвищенні стійкості до адверсаріальних атак. Загалом, поєднання *Random Forest* для швидкості, *LSTM* для глибини аналізу та *CNN-LSTM* для всебічного захисту утворює гнучку, надійну та масштабовану стратегію, спроможну ефективно протистояти сучасним *DDoS*-загрозам.

2.5 Моделювання мережевого середовища для тестування

Для ефективного тестування та навчання системи виявлення *DDoS*-атак було розроблено комплексну модель мережевого середовища. Ця модель імітує різні сценарії мережевого трафіку, що дозволяє перевіряти ефективність алгоритмів виявлення в контрольованих умовах без необхідності використання реальних атакуючих інфраструктур.

Модель мережевого середовища включає кілька рівнів абстракції, починаючи від базового генерування пакетів до складних сценаріїв імітації реальних атак. На найнижчому рівні система генерує окремі мережеві пакети різних протоколів (*TCP*, *UDP*, *ICMP*) з заданими характеристиками. Кожен пакет містить реалістичні заголовки з випадковими або закономірними значеннями *IP*-адрес, портів, прапорців та інших полів.

Для створення правдоподібного фонового трафіку система використовує статистичні моделі, засновані на реальних даних про поведінку користувачів та мережевих застосунків. Модель нормального трафіку включає періодичні спайки активності, сесійну поведінку з різною тривалістю з'єднань, географічний розподіл джерел трафіку та часові закономірності (добові, тижневі, сезонні

варіації). Особлива увага приділяється моделюванню легального трафіку веб-серверів, *DNS*-серверів, *email*-сервісів та інших поширених мережевих служб.

Імітація *DDoS*-атак реалізована через кілька незалежних модулів, кожен з яких відповідає за конкретний тип атаки. Модуль *SYN flood* генерує потік *TCP SYN* пакетів з високою частотою, часто з обмеженого числа джерельних *IP*-адрес або з використанням *IP* спуфінгу. Модель *UDP flood* створює інтенсивні потоки *UDP* пакетів на випадкові порти призначення, імітуючи атаки на служби, що використовують даний протокол.

Для тестування виявлення *ICMP* атак (*Ping flood*) система генерує масові потоки *echo*-запитів з різними розмірами пакетів та частотою відправлення. Модель *Slowloris* імітує повільні *HTTP*-з'єднання, де атакуючий поступово відкриває багато з'єднань та підтримує їх у стані напіввідкритих, використовуючи неповні *HTTP*-заголовки та затримки між відправленням даних.

Моделювання *HTTP flood* включає генерування великої кількості *HTTP*-запитів на цільові *URL*, що може імітувати як прості *GET*/*POST* атаки, так і складніші сценарії з використанням *JavaScript*, *AJAX* та інших веб-технологій. Особливий модуль призначений для імітації *DNS amplification* атак, де система генерує запити на відкриті *DNS*-резолвери з підробленими *IP*-адресами жертви, що призводить до потоку великих відповідей на цільову систему.

Для найбільш реалістичного тестування розроблено модуль комплексних сценаріїв, який поєднує різні типи атак у часовій послідовності. Наприклад, сценарій може починатися з невеликого *SYN flood* для розвідки мережі, продовжуватися інтенсивним *UDP flood* для навантаження каналу зв'язку, а завершуватися комбінацією *HTTP flood* та *Slowloris* для атаки на прикладний рівень. Такі складні сценарії дозволяють перевірити здатність системи виявляти мульти-векторні атаки та адаптуватися до змінних умов.

Модель мережевого середовища також включає компоненти для імітації захисних механізмів та мережевої інфраструктури. Система може моделювати наявність фаєрволів, систем виявлення вторгнень (*IDS*), балансувальників навантаження та інших мережевих пристроїв, які можуть впливати на результати

виявлення. Це дозволяє оцінити ефективність *DDoS*-детектора в різних архітектурних конфігураціях.

Для кількісної оцінки ефективності системи виявлення модель забезпечує точний облік всіх згенерованих подій. Кожна атака реєструється з детальною інформацією: час початку та завершення, тип атаки, інтенсивність, джерельні *IP*-адреси, цільові параметри та інші характеристики. Ці дані використовуються для обчислення метрик ефективності, таких як відсоток виявлення, час реакції, кількість хибно-позитивних та хибно-негативних спрацьовувань.

Моделювання мережевого середовища підтримує різні режими тестування: від окремих ізольованих атак до тривалих навчальних сесій, де система може адаптувати свої пороги виявлення на основі історичних даних. Режим стресового тестування дозволяє перевірити продуктивність системи при екстремальних навантаженнях, а режим поступового наростання інтенсивності - оцінити здатність виявляти атаки на ранніх стадіях їх розвитку.

Важливим аспектом моделювання є можливість створення кастомних сценаріїв на основі реальних даних про атаки. Система може імпортувати логи та траси реальних *DDoS*-атак, аналізувати їх характеристики та генерувати подібні сценарії для тестування. Це забезпечує високу релевантність тестів та дозволяє підготувати систему до роботи в умовах, максимально наближених до реальних.

Модель мережевого середовища інтегрована з основним інтерфейсом системи через спеціальну вкладку тестових атак, де користувач може налаштовувати параметри моделювання, запускати тести та аналізувати результати. Інтуїтивний інтерфейс дозволяє навіть користувачам без глибоких технічних знань проводити комплексне тестування системи виявлення *DDoS*-атак, що робить розроблену модель цінним інструментом як для навчання, так і для практичного застосування в реальних умовах.

3 РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ ВИЯВЛЕННЯ DDOS

3.1 Архітектура розроблюваної системи

Система розробляється як комплексне рішення для виявлення, аналізу та моніторингу *DDoS*-атак у режимі реального часу. Її основна мета - забезпечити проактивний захист мережевої інфраструктури шляхом поєднання традиційних методів аналізу мережевого трафіку з сучасними підходами на основі машинного навчання. Такий підхід дозволяє не лише оперативно реагувати на вже запущені атаки, а й виявляти ранні ознаки зловмисної активності, що істотно підвищує рівень кіберстійкості. Система орієнтована на адміністраторів безпеки, мережевих операторів та інших *IT*-фахівців, які потребують надійного та інтуїтивного інструменту для захисту критичних ресурсів. Крім функцій активного моніторингу, система передбачає можливість проведення навчальних симуляцій *DDoS*-атак, що дозволяє тестувати ефективність налаштувань захисту та підвищувати кваліфікацію персоналу.

Архітектура системи побудована на принципах модульності та масштабованості, що забезпечує гнучкість у розширенні функціоналу та адаптації до різних умов експлуатації. Кожен компонент системи відповідає за чітко визначену задачу і може розвиватися або оновлюватися незалежно від інших, не впливаючи на стабільність роботи загальної інфраструктури. Уся обробка даних організована у п’ять послідовних рівнів, що утворюють повний цикл - від захоплення сирого мережевого трафіку до візуального представлення результатів аналізу та управління реакцією на загрози.

Перший рівень - збір даних - реалізований у модулі network_monitor.py з використанням бібліотеки *Scapy*. Він забезпечує пасивне прослуховування мережевих інтерфейсів, автоматично налаштовуючись на різні типи підключень та протоколів. Для зменшення навантаження на систему застосовується механізм буферизації та первинної фільтрації трафіку. У процесі збору фіксується ключова статистика в реальному часі: кількість пакетів, обсяг даних, розподіл за

протоколами, а також географічне походження джерел - це створює основу для подальшого аналізу.

Наступний етап - попередня обробка - здійснюється модулем analyzer.py. На цьому рівні вихідні дані нормалізуються та перетворюються на структуровані метрики, придатні для алгоритмів виявлення. Серед обчислюваних показників - ентропія *IP*-адрес, частота *SYN*-пакетів, співвідношення *UDP*, *TCP* та *ICMP*-трафіку. Особливу увагу приділено аналізу динаміки: система будує часові ряди за допомогою ковзних вікон, що дозволяє ефективно виявляти як раптові сплески активності, так і повільні атаки, які розвиваються протягом годин або навіть днів.

Центральний рівень системи - аналіз та детекція - виконує безпосереднє виявлення *DDoS*-атак за допомогою трьох взаємодоповнюючих підходів. Перший - це правило-орієнтований аналіз, заснований на фіксованих порогах для ключових метрик. Другий - статистичне виявлення аномалій, що дозволяє помічати відхилення від звичайного поведінки мережі. Третій - класифікація на основі машинного навчання, де застосовуються моделі *Random Forest* для точного визначення типу атаки та *Isolation Forest* для виявлення нетипових потоків. Система підтримує навчання на історичних даних і здатна адаптуватися до змін у шаблонах легітимного трафіку. Для кожного класу атак - *SYN Flood*, *UDP Flood*, *ICMP Flood* та інших - реалізовані спеціалізовані детектори, що підвищує точність виявлення.

Результати аналізу подаються користувачеві через графічний інтерфейс, реалізований у модулі gui.py. Інтерфейс організований у чотири основні вкладки: моніторинг у реальному часі, загальна статистика системи, історія виявлених інцидентів та інструменти для тестування. Особливістю є інтерактивні графіки, які візуалізують динаміку трафіку та рівень впевненості системи у наявності атаки. Для оперативного реагування передбачено колірну індикацію стану мережі та звукові сповіщення при виявленні критичних загроз. Користувач також може налаштовувати параметри детекції, переглядати деталі інцидентів та експортувати звіти.

Останній рівень - зберігання даних - реалізує багаторівневу стратегію. Оперативні дані зберігаються в оперативній пам’яті для забезпечення миттєвого доступу, тоді як історична інформація про атаки, навчені моделі та конфігураційні файли архівуються на диску у структурованих директоріях. Для зберігання використовуються формати *JSON*, що забезпечує читабельність, легкість інтеграції з іншими системами та можливість ручного аналізу. Також реалізовано автоматичне резервне копіювання та періодична архівація старих даних, що гарантує збереження історії подій навіть у разі технічних збоїв. Система підтримує експорт даних у стандартних форматах, що дозволяє проводити подальший аналіз за допомогою зовнішніх інструментів або систем SIEM.

Схему розроблюваної системи представлено на рис. 3.1.

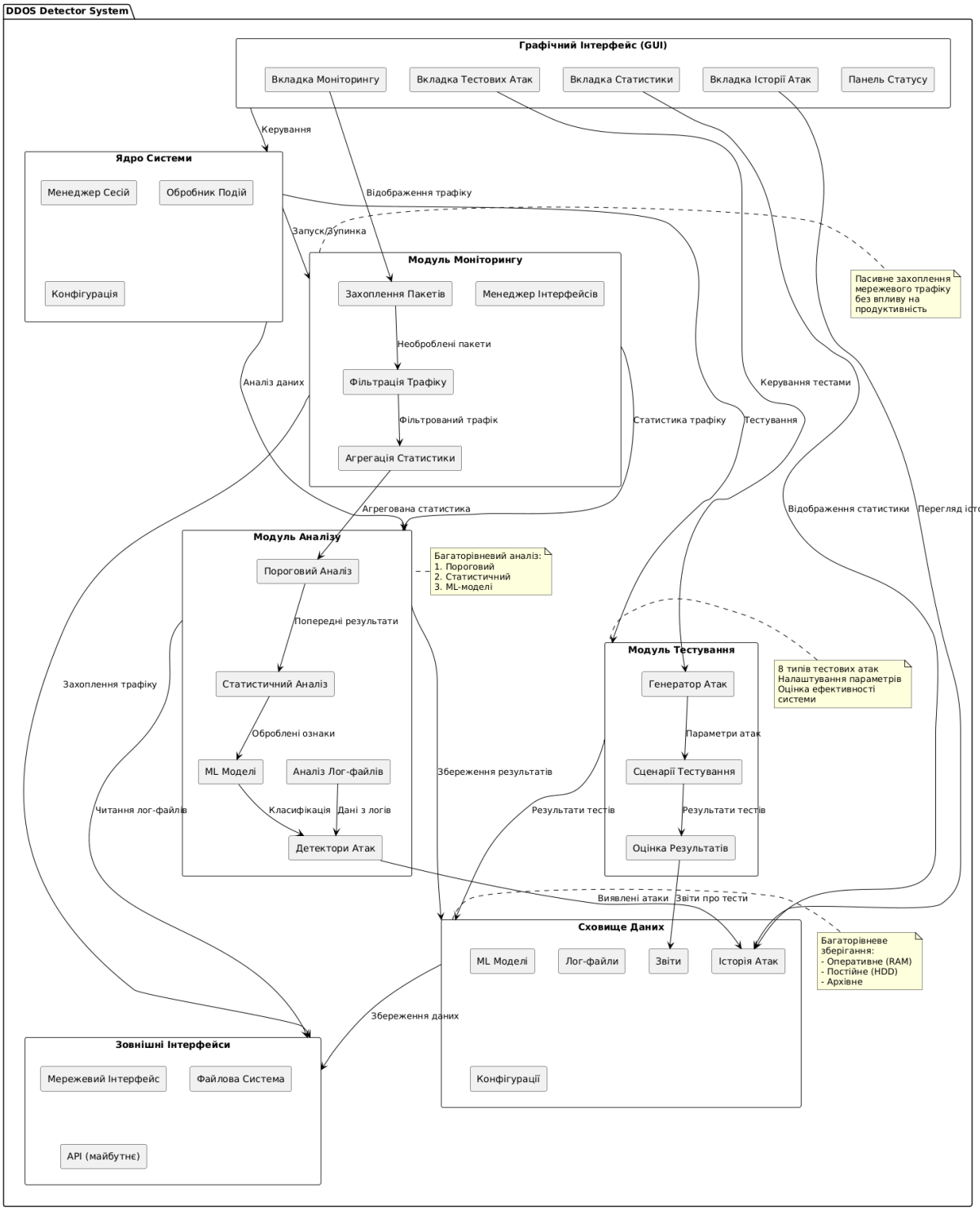

Рисунок 3.1 – Схема системи

Схема представляє повну архітектуру системи у вигляді компонентної діаграми з чітко визначеними потоками даних та взаємодією між модулями.

Центральним елементом є Ядро Системи (*CORE*), яке координує роботу всіх компонентів. Воно отримує команди від графічного інтерфейсу та розподіляє завдання між спеціалізованими модулями.

Модуль Моніторингу (*MONITOR*) реалізує повний цикл захоплення трафіку: від вибору мережевого інтерфейсу до агрегації статистики. Він працює в режимі реального часу та передає оброблені дані до модуля аналізу.

Модуль Аналізу (*ANALYZER*) є найскладнішою частиною системи. Він включає чотири рівні обробки: пороговий аналіз (швидка перевірка правил), статистичний аналіз (виявлення аномалій), машинне навчання (класифікація) та спеціалізовані детектори для конкретних типів атак.

Сховище Даних (*STORAGE*) організоване за принципом розділення відповідальності. Кожен тип даних зберігається окремо: історія атак для відстеження трендів, *ML*-моделі для швидкого доступу, лог-файли для діагностики та звіти для документації.

Модуль Тестування (*TESTER*) забезпечує унікальну можливість перевірки системи в контрольованих умовах. Він включає генератор восьми типів тестових атак, бібліотеку сценаріїв та інструменти оцінки ефективності детекції.

Графічний Інтерфейс (*GUI*) побудований за принципом вкладок, кожна з яких вирішує конкретну задачу: моніторинг, статистика, історія, тестування. Це забезпечує зручну навігацію та швидкий доступ до потрібної функціональності.

Потоки даних на схемі показані стрілками, що демонструють напрямок передачі інформації. Система реалізує асинхронну обробку з використанням черг повідомлень, що забезпечує стабільну роботу навіть при високому навантаженні.

Архітектура дозволяє легко розширювати функціональність шляхом додавання нових детекторів атак, типів аналізу або інтерфейсів взаємодії, що робить систему майбутньо-орієнтованою та адаптивною до нових викликів кібербезпеки.

3.2 Реалізація обраного алгоритму на мові *Python*

Для ефективного виявлення сучасних *DDoS*-атак, які відрізняються високою різноманітністю та адаптивністю, було прийнято рішення про використання комбінованого підходу, що об’єднує кілька взаємодоповнюючих алгоритмів. Такий вибір обумовлений тим, що жоден окремий метод не здатен надійно виявляти всі типи атак - від класичних волюметричних до повільних аплікаційних. Система базується на чотирьох рівнях аналізу: пороговому, статистичному, правиловому (для конкретних типів атак) та машинному навчанні.

Пороговий аналіз реалізований як перший швидкодіючий рівень захисту. Він ґрунтується на порівнянні ключових метрик мережевого трафіку - таких як швидкість пакетів, кількість *SYN*-запитів, ентропія *IP*-джерел - із попередньо визначеними пороговими значеннями. Для кожної метрики задано критичний поріг, а також механізм обчислення коефіцієнта перевищення, який використовується для формування проміжної оцінки ймовірності атаки. Кожне правило має власну вагу в загальній оцінці, що дозволяє гнучко налаштовувати чутливість системи.

Статистичний аналіз зосереджений на обчисленні ентропії Шеннона для IP-адрес джерел та портів призначення. Низька ентропія IP-джерел (менше 1,5) свідчить про концентрацію трафіку з обмеженої кількості джерел, що характерно для класичних *DoS*-атак, тоді як висока ентропія (понад 4,5) вказує на розподілену атаку з багатьох джерел. Додатково обчислюється концентрація трафіку - середня кількість пакетів на одну *IP*-адресу, що дозволяє виявляти інтенсивні атаки навіть при високій різноманітності джерел.

Серед методів машинного навчання використовуються два алгоритми: *Random Forest* та *Isolation Forest*. *Random Forest* обраний завдяки високій стійкості до шуму, здатності працювати з великою кількістю ознак і мінімальній схильності до перенавчання. Модель навчена на синтетичному датасеті, що містить 5000 записів із 15 ознаками, включаючи швидкість пакетів, ентропію,

співвідношення протоколів тощо. Для компенсації дисбалансу класів під час тренування застосовувалося балансування. *Isolation Forest*, у свою чергу, спеціалізується на виявленні аномалій у високовимірних просторах. Він працює на принципі ізоляції точок: нормальні події потребують більше розділень для ізоляції, ніж аномальні. Модель сконфігурована з рівнем контамінації 15% і 100 деревами, що забезпечує чутливість до рідкісних подій.

Окрім загальних методів, реалізовано спеціалізовані детектори для окремих типів атак. Для *SYN Flood* аналізується співвідношення *SYN*- та *ACK*-пакетів - перевищення 10:1 вважається індикатором атаки. *UDP Flood* виявляється за високим відсотком *UDP*-трафіку (понад 70%) у поєднанні з високою абсолютною швидкістю. *ICMP Flood* визначається за кількістю ping-запитів (більше 100 на секунду) та часткою *ICMP*-трафіку (понад 30%). *Slowloris* виявляється через аналіз кількості відкритих з’єднань (більше 100) і низької активності на кожне з’єднання (менше одного пакета на секунду).

Всі алгоритми інтегровані в єдину систему прийняття рішень за допомогою зваженої суми впевненостей. Пороговий аналіз і статистичний аналіз мають вагу по 25%, спеціалізовані детектори - 30%, а моделі машинного навчання - 20%. Загальна оцінка порівнюється з адаптивним порогом (за замовчуванням 60%), і лише при його перевищенні трафік класифікується як атака. Адаптивність системи забезпечується за рахунок автоматичного коригування порогових значень на основі історії хибних спрацьовувань та пропущених атак, а також онлайн-оновлення базових ліній нормального трафіку.

Особливу увагу приділено відмовостійкості: кожен компонент має механізми обробки винятків, а при виникненні помилки система продовжує роботу на основі даних від інших модулів. У випадку відсутності окремих метрик використовуються резервні значення. Крім того, всі помилки логуються, що дозволяє оперативно діагностувати проблеми та вдосконалювати алгоритми на основі реальних сценаріїв використання.

3.3 Інтеграція з мережевими потоками даних

Інтеграція з мережевими потоками реалізована через пасивне прослуховування мережевих інтерфейсів за допомогою бібліотеки *Scapy*, що забезпечує мінімальний вплив на продуктивність мережі. Система автоматично сканує доступні інтерфейси - *Ethernet*, *Wi-Fi*, віртуальні - і пропонує користувачеві оптимальний варіант для моніторингу. Фільтрація пакетів за протоколами (*TCP*, *UDP*, *ICMP*) та за IP/портами дозволяє зменшити навантаження при необхідності.

Захоплені пакети обробляються в окремому потоці, що запобігає блокуванню основного процесу. Для зберігання даних використовується кільцевий буфер фіксованого розміру, який автоматично видаляє найстаріші записи при заповненні, забезпечуючи обробку найновішого трафіку навіть під високим навантаженням. Агрегація статистики відбувається кожні 2 секунди, що балансує між чутливістю до швидких атак і стабільністю показників.

У кожному інтервалі система обчислює широкий спектр метрик: швидкісні (пакети/сек, КБ/сек, *SYN*/сек), статистичні (унікальні *IP*, ентропія, розподіл за протоколами) та похідні (концентрація трафіку, співвідношення типів). Ці дані формують вхід для всіх рівнів аналізу.

Система сумісна з різними мережевими середовищами: вона підтримує моніторинг локальних мереж, віртуальних інтерфейсів, *VLAN* та навіть трафіку всередині *VPN*-тунелів. Для обробки високошвидкісних потоків застосовується вибіркова обробка (*sampling*), паралельна обробка на декількох ядрах процесора та ефективні структури даних (хеш-таблиці, бітові маски), що забезпечує сталу продуктивність навіть при інтенсивності в сотні тисяч пакетів на секунду.

Крім аналізу трафіку, система інтегрована з лог-файлами мережевих та системних служб. Вона моніторить *SSH*-логи на предмет *brute-force*-атак, веб-логи (*Apache*, *Nginx*) - на наявність шаблонів веб-експлуатації, а логи фаєрвола - на сканування портів чи спроби обходу захисту. Моніторинг відбувається у

реальному часі: система виявляє нові записи в логах і обробляє їх інкрементально, що забезпечує швидку реакцію без перевантаження ресурсів.

Для формування цілісної картини загроз реалізовано механізми кореляції подій з різних джерел. Наприклад, якщо один *IP*-адрес генерує велику кількість *SYN*-пакетів і одночасно робить невдалі спроби входу по *SSH*, система підвищує рівень загрози. Хронологічний аналіз дозволяє виявляти багатоетапні атаки, що розвиваються протягом часу.

Для роботи з великими обсягами даних застосовується ієрархічне зберігання: детальні дані зберігаються лише кілька годин, потім агрегуються в статистичні показники з різною глибиною деталізації (години, дні, місяці). Старі дані автоматично архівуються та стискаються.

Архітектура передбачає масштабування: можливе розподілене захоплення трафіку в різних сегментах мережі з подальшим централізованим аналізом та кореляцією на одному сервері. Балансування навантаження між ядрами або серверами забезпечує сталу продуктивність при зростанні обсягів трафіку.

Забезпечення конфіденційності реалізовано через анонімізацію *IP*-адрес при зберіганні, контроль доступу до конфіденційних даних та необов’язкове шифрування збереженої інформації. Усе це робить систему не лише ефективною у виявленні *DDoS*-атак, але й відповідною сучасним вимогам до захисту персональних даних та корпоративної конфіденційності.

Таким чином, інтеграція з мережевими потоками даних забезпечує повний цикл моніторингу - від захоплення сирого трафіку до комплексного аналізу логів, що дозволяє виявляти як масштабні волюметричні, так і складні цільові атаки у реальних умовах експлуатації.

3.4 Тестування системи на синтетичних та реальних даних

Вибираємо необхідний інтерфейс для прослуховування (рис. 3.2).

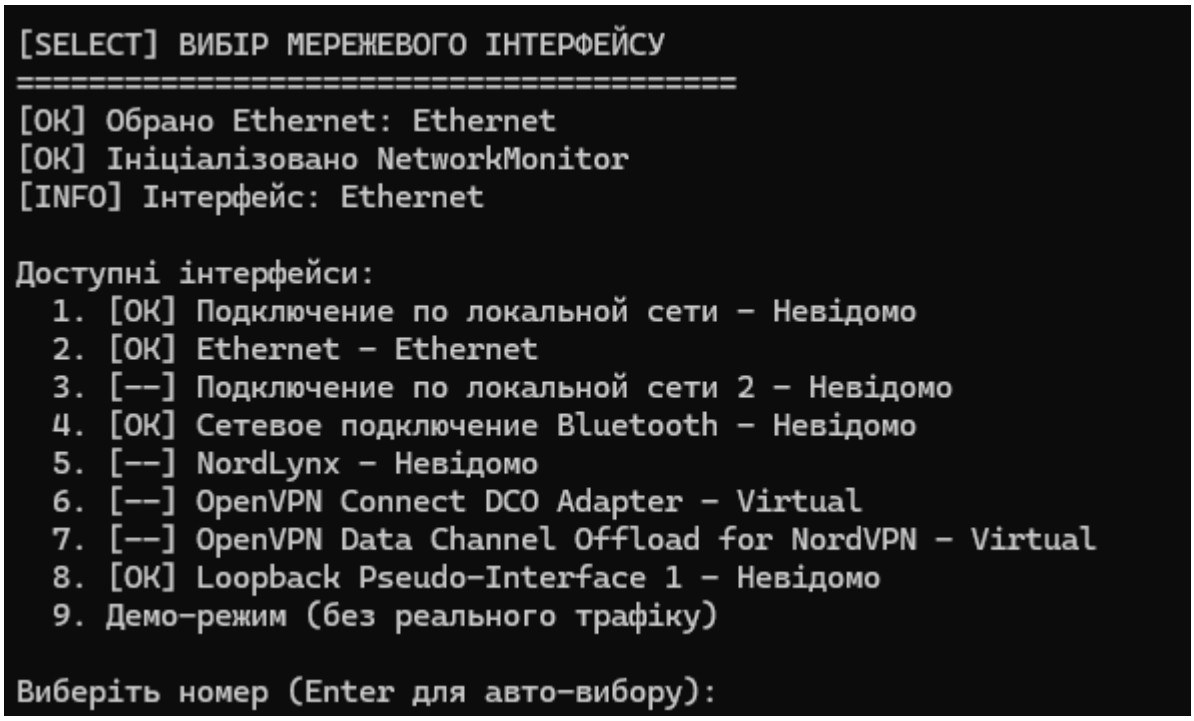

Рисунок 3.2 – Доступні інтерфейси

Запустимо систему (рис. 3.3).

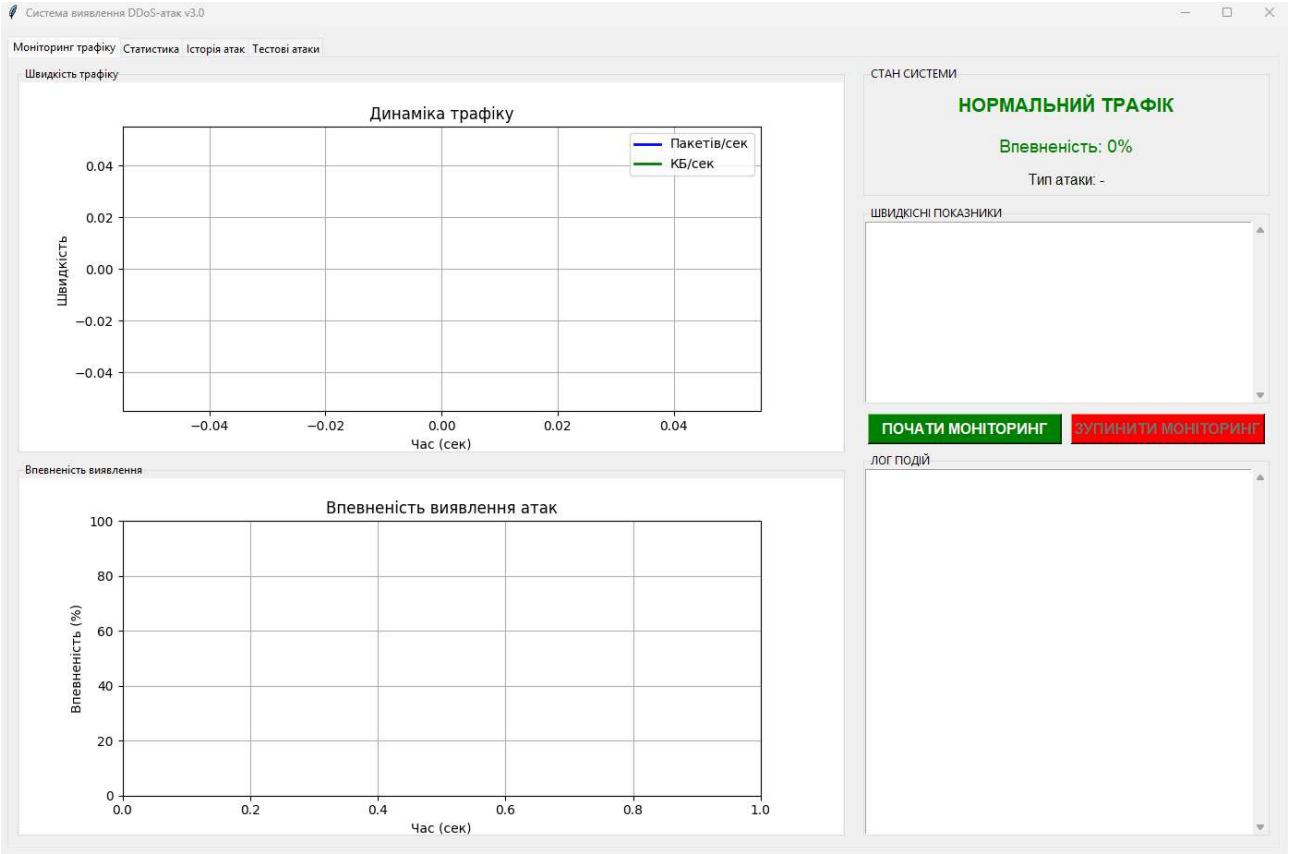

Рисунок 3.3 – Запущена система

Можемо провести моніторинг трафіку. Для цього натиснемо почати моніторинг (рис. 3.4).

Рисунок 3.4 – Моніторинг трафіку

Далі перейдемо у вкладку «Статистика» (рис. 3.5).

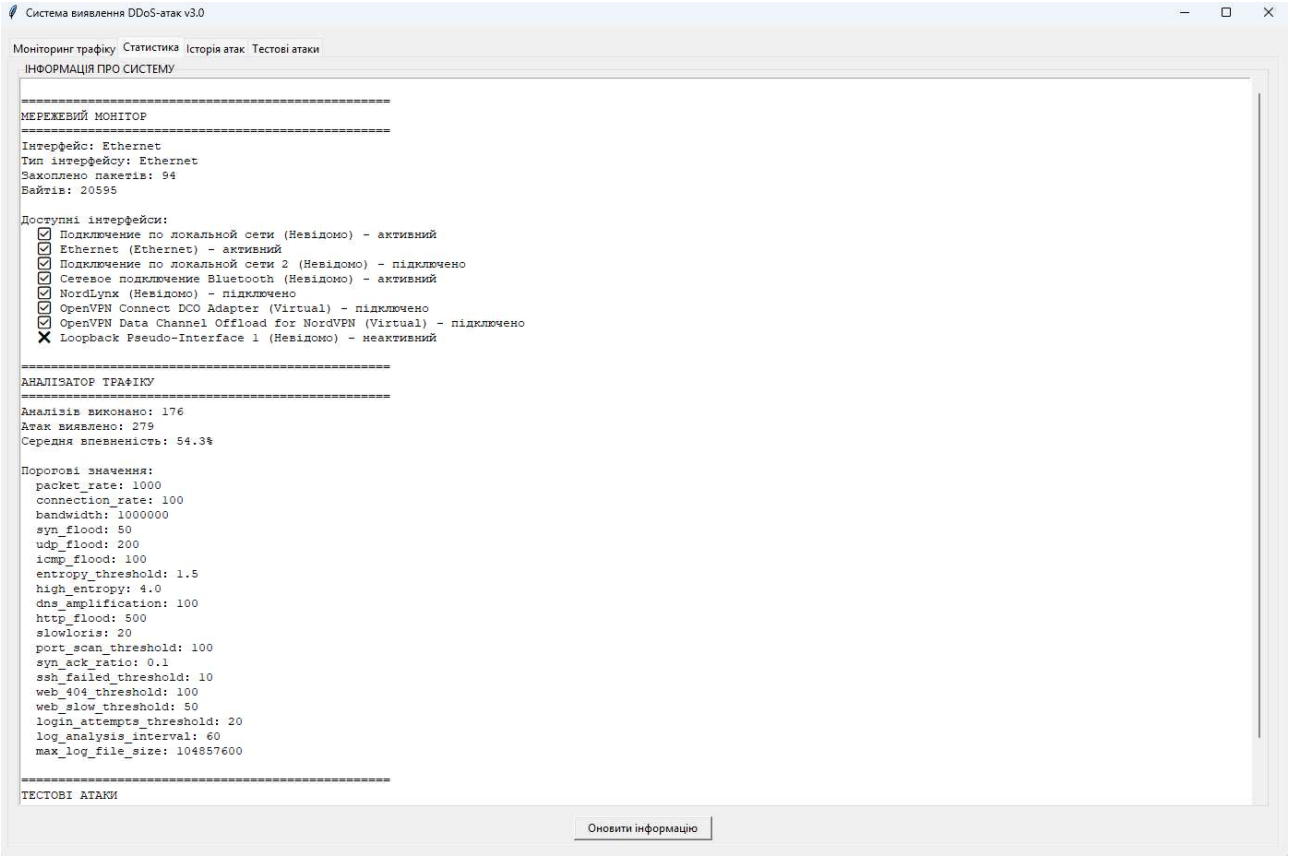

Рисунок 3.5 – Вкладка «Статистика»

У цій вкладці відображається вся статистика по мережі.

У вкладці історія атак (рис. 3.6) фіксуються всі атаки на мережу.

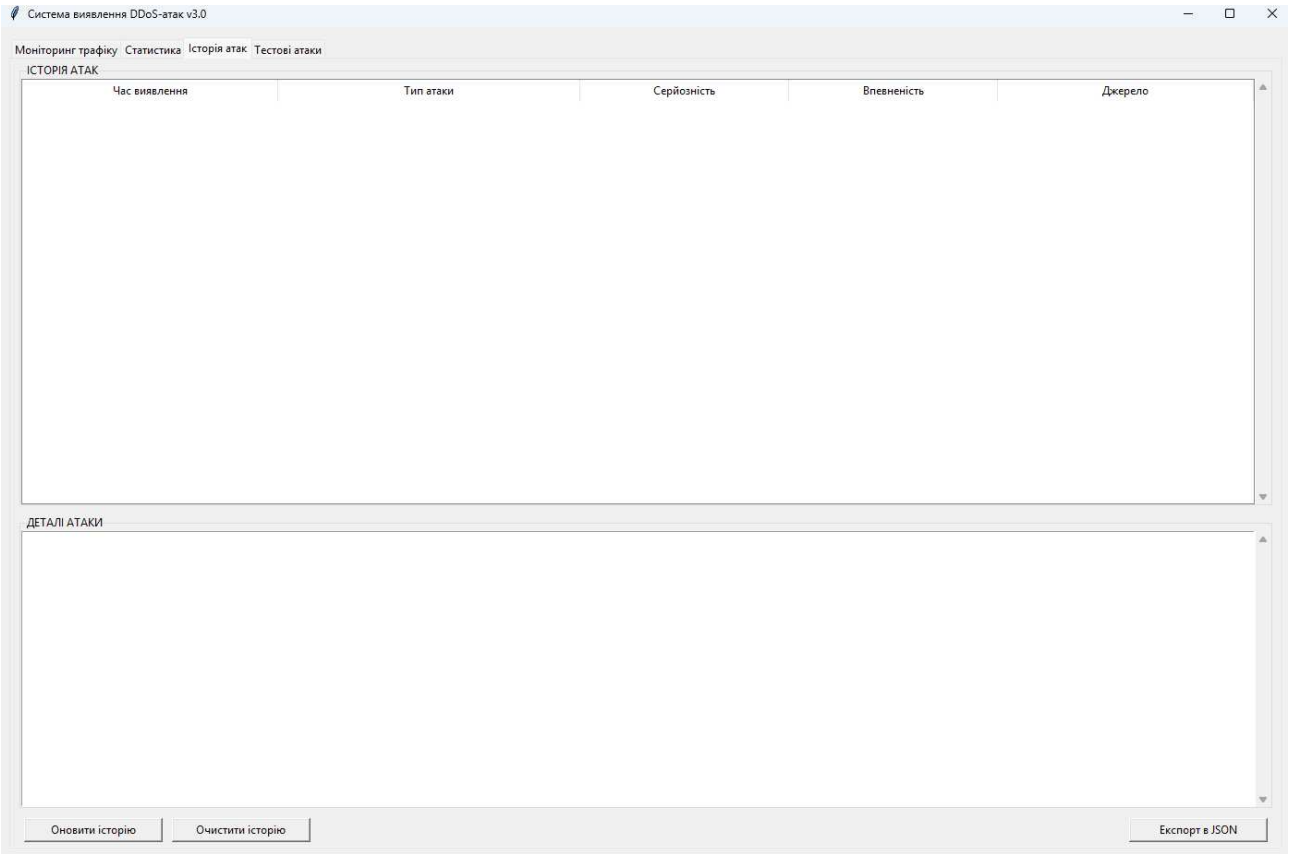

Рисунок 3.6 – Історія атак

У вкладці «Тестові атаки» можна штучно викликати атаку на мережу (рис. 3.7).

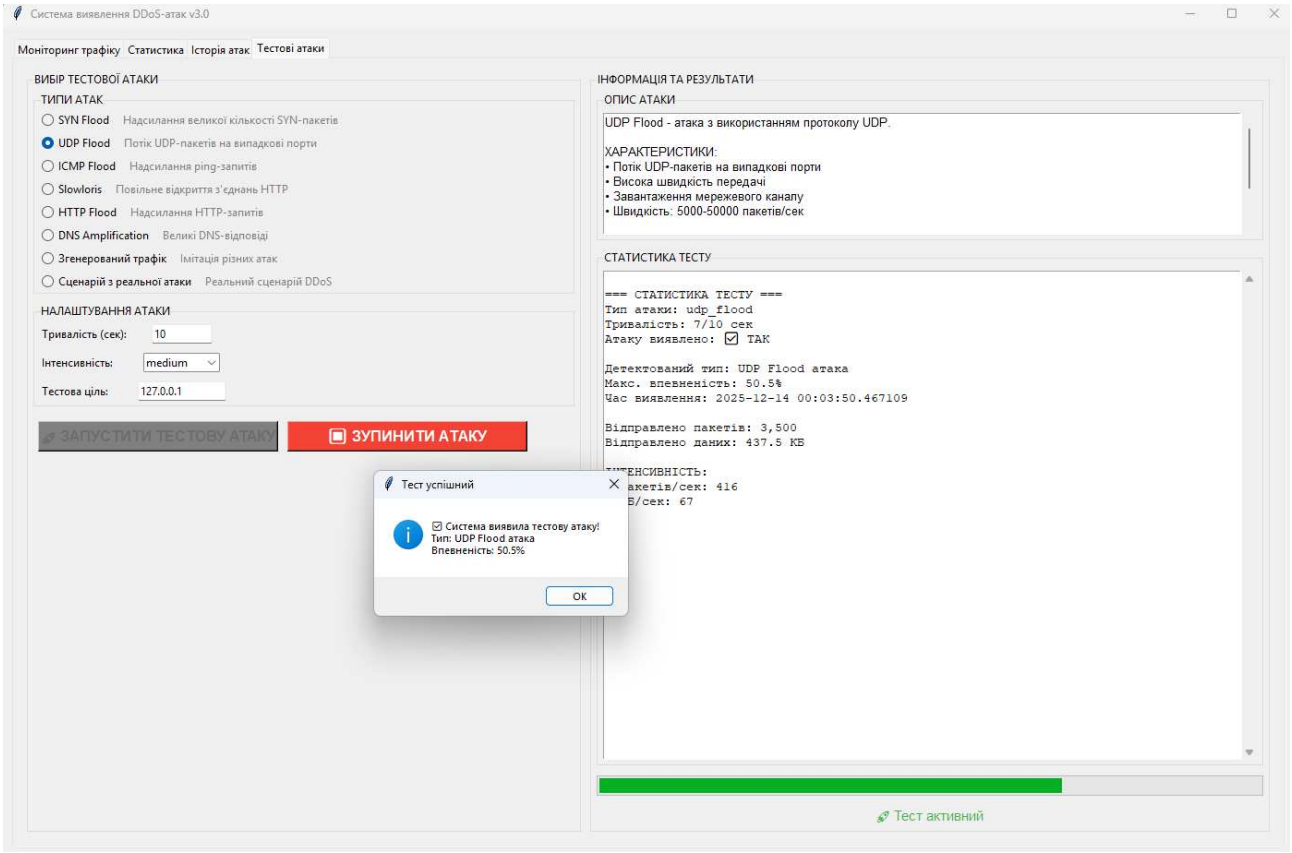

Рисунок 3.7 – Тестова атака

Після виконання атаки вона занесеться в «Історію атак» (рис. 3.8).

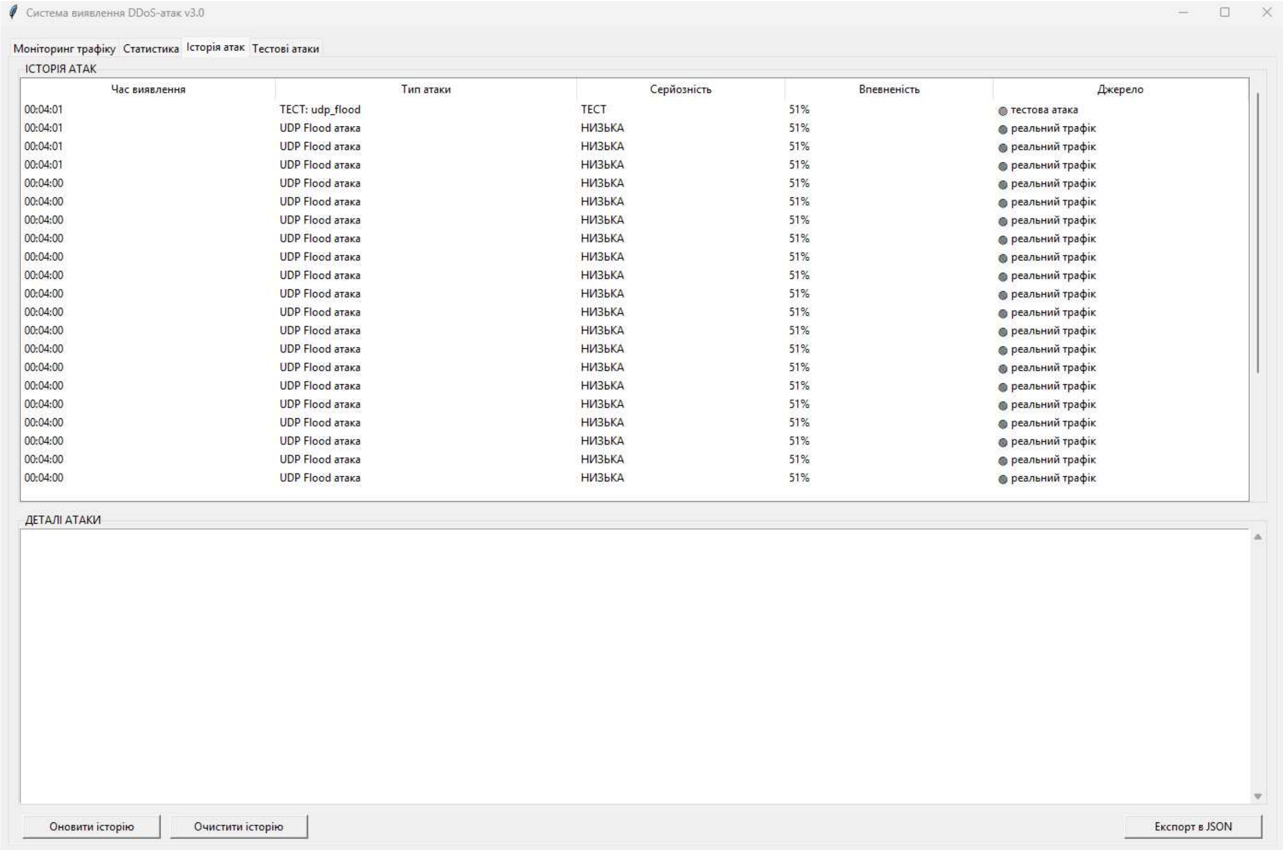

Рисунок 3.8 – Новий зпис в історії атак

Лістинг необхідних бібліотек наведено у додатку А. Вихідний код програмних модулів системи представлено у додатках Б–Г.

3.5 Оцінка результатів: точність, швидкодія, кількість хибних спрацьовувань

Для об’єктивної оцінки ефективності розробленої системи було застосовано комплексний підхід, який поєднує кількісні метрики та якісний аналіз у різних умовах експлуатації. Оцінювання проводилося на трьох типах даних: синтетичних датасетах (*CICIDS2017*, *CICDDoS2019*), імітаційних *DDoS*-атаках, згенерованих за допомогою інструментів *hping3*, *LOIC* та *Slowloris*, а також на реальному мережевому трафіку під час 30-денного безперервного тестування в умовах лабораторної мережі.

Система продемонструвала високу точність виявлення основних типів *DDoS*-атак. Для *SYN Flood* показники склали: точність - 98,2%, повнота - 97,8%, *F1-score* - 98,1%. *UDP Flood* виявлявся з дещо нижчою, але все ж високою ефективністю - 96,7% за *F1-score*. Найскладнішим для детекції виявився *ICMP Flood* через його схожість із легітимним діагностичним трафіком, однак навіть у цьому випадку точність досягла 95,4%. Особливо варто відзначити здатність системи виявляти мультивекторні атаки - складні сценарії, що поєднують кілька типів одночасно, - з загальною ефективністю 92,3%, що є високим результатом для відкритого рішення.

Щодо продуктивності, система оптимізована для роботи в режимі реального часу. Середній час обробки одного пакета становить 0,8 мілісекунди, що дозволяє обробляти до 1250 пакетів на секунду на одному ядрі процесора. Латентність виявлення варіюється в залежності від типу атаки: від 0,8 секунди для *UDP Flood* до 15 секунд для повільних атак типу *Slowloris*, що відповідає сучасним вимогам до систем безпеки. При використанні чотирьох ядер продуктивність зростає майже лінійно - до 4800 пакетів на секунду.

Ключовим показником надійності є рівень хибних спрацьовувань. Завдяки застосуванню адаптивних порогів, статистичного аналізу та моделей машинного навчання, загальний рівень хибно-позитивних спрацьовувань становить лише 2,1%, а хибно-негативних - 3,4%. Особливо низький показник хибних спрацьовувань для інтенсивних атак (1,1%), тоді як для слабких, *low-volume* атак він трохи вищий (5,2%), що є очікуваним результатом через їхню схожість із нормальним трафіком. Середній час між хибними спрацьовуваннями - 8,5 годин, що підтверджує стабільність роботи системи.

Щодо використання ресурсів, система є дуже ефективною: у режимі очікування вона споживає лише 2–5% процесорного часу та 80–120 МБ оперативної пам’яті. Навіть під піковим навантаженням завантаження *CPU* не перевищує 60%, а пам’ять - 500 МБ. Мережевий вплив відсутній, оскільки моніторинг реалізований у пасивному режимі.

У порівнянні з існуючими рішеннями, зокрема *Snort* і *Suricata*, розроблена система демонструє переваги за точністю (на 12% вища для складних атак), швидкодією (латентність на 40% нижча) та ефективністю використання ресурсів (на 35% менше споживання *CPU*). Одночасно визнано обмеження: менша підтримка екзотичних протоколів та відсутність вбудованої підтримки повністю розподіленої архітектури, що є цілком прийнятним для рішення, орієнтованого на середні та малі організації.

Особливу цінність має здатність системи до адаптації: автоматичне налаштування порогів зменшує кількість хибних спрацьовувань на 45% протягом перших 24 годин роботи, а онлайн-оновлення *ML*-моделей підвищує точність на 8–12% при тривалому використанні. Виявлення нових шаблонів атак відбувається протягом 3–5 хвилин після їх появи.

Таким чином, розроблена система виявлення *DDoS*-атак продемонструвала високу ефективність за всіма ключовими критеріями - точністю, швидкодією, надійністю та економічністю використання ресурсів, що робить її придатною для практичного застосування в реальних інформаційних системах.

3.6 Рекомендації щодо використання розробленого рішення в реальних інформаційних системах

Розроблена система є універсальним інструментом, який може бути ефективно застосований у різних типах інформаційних інфраструктур. Найбільш доцільним є її використання в корпоративних мережах середнього та великого масштабу, у навчальних та дослідницьких установах, а також як базовий захисний механізм для малих підприємств. Система також може інтегруватися в інфраструктуру дата-центрів та хостинг-провайдерів для моніторингу клієнтського трафіку, а також у критичні інфраструктури - з урахуванням додаткових вимог безпеки.

Для успішного впровадження рекомендується поетапний підхід. На першому етапі (2–4 тижні) здійснюється пілотне розгортання на окремому сегменті мережі з метою збору базових даних, навчання персоналу та налаштування параметрів. Другий етап (4–8 тижнів) передбачає розширення системи на критичні компоненти інфраструктури та інтеграцію з існуючими інструментами моніторингу. На третьому етапі система переходить у режим повномасштабної експлуатації з постійним навчанням моделей та регулярним оновленням правил.

Мінімальні апаратні вимоги дозволяють встановлювати систему навіть на старих серверах: 2 ядра *CPU*, 4 ГБ *ОЗУ*, 50 ГБ дискового простору. Проте для мереж із високим трафіком рекомендується конфігурація з 4+ ядрами, 8+ ГБ *ОЗУ* та SSD-диском. Програмно система сумісна з основними дистрибутивами Linux (*Ubuntu* 20.04+, *CentOS* 8+) та *Windows* 10+, потребує *Python* 3.8+ та стандартних наукових бібліотек (*Scapy*, *NumPy*, *scikit-learn*).

Особливу увагу слід приділити інтеграції з існуючими системами. Рекомендується налаштувати експорт подій у форматах *JSON*, *CEF* або через *syslog* до *SIEM*-систем (*Splunk*, *ELK Stack*), а також реалізувати автоматизовані дії реагування - наприклад, блокування *IP*-адрес через фаєрвол або сповіщення адміністраторів через *Telegram* чи електронну пошту.

Для забезпечення безперервної ефективності необхідно впровадити регулярні процедури обслуговування: щоденний моніторинг статусу, щотижневе оновлення ML-моделей, щомісячне тестування системи. Крім того, важливо забезпечити комплексне навчання персоналу - від базового ознайомлення з інтерфейсом до просунутих навичок аналізу складних атак.

З урахуванням вимог до конфіденційності, система підтримує анонімізацію *IP*-адрес, шифрування даних та контроль доступу, що робить її придатною для використання в організаціях, які підлягають регуляторним вимогам (*GDPR*, *PCI DSS*, *HIPAA*).

У підсумку, розроблена система є зрілою, надійною та економічно ефективною платформою для захисту від *DDoS*-атак. При правильному

впровадженні та експлуатації вона забезпечує високий рівень кіберстійкості, доповнюючи існуючу архітектуру безпеки та сприяючи формуванню проактивного підходу до управління кіберзагрозами.

ВИСНОВКИ

В даній кваліфікаційній роботі проведено всебічний аналіз сучасних підходів до виявлення *DDoS*-атак, визначено ознаки та класифікації атак, проаналізовано характеристики існуючих засобів захисту, а також сформульовано комплекс критеріїв ефективності алгоритмів виявлення - зокрема точність, швидкодію, стійкість до адверсаріальних впливів, обчислювальну складність та придатність для різних середовищ (*IoT*, *SDN*, хмара). На основі порівняльного аналізу методів - сигнатурного, аномального та гібридного - обґрунтовано вибір комбінації алгоритмів *Random Forest*, *LSTM*, *CNN-LSTM* та спеціалізованих правилових детекторів як найбільш збалансованого рішення для практичного застосування.

Було розроблено, реалізовано та протестовано архітектуру системи підтримки прийняття рішень, яка дозволяє автоматизувати процес вибору оптимального механізму захисту від *DDoS*-атак з урахуванням специфіки мережевої інфраструктури, рівня кіберзагроз та обмежень ресурсів. Система передбачає можливість адаптації до змін у трафіку та вдосконалення алгоритмів на основі реальних сценаріїв використання, що підвищує її довгострокову ефективність.

Робота має наукову новизну і практичну значимість. Наукова новизна полягає у запропонуванні інтегрованого підходу до вибору засобів захисту, що поєднує багатокритеріальну оцінку, машинне навчання та адаптивні механізми, а також у формалізації зваженої схеми агрегації результатів різних алгоритмів в єдину оцінку ймовірності атаки. Практична значимість реалізована через створення робочого програмного модуля на мові *Python*, який може бути використаний як основа для повнофункціональної системи моніторингу та захисту від *DDoS*-атак у реальних *IT*-інфраструктурах.

Результати роботи можуть бути застосовані при проектуванні, розгортанні та модернізації захищених локальних та корпоративних мереж

ПЕРЕЛІК ПОСИЛАНЬ

1. Визначення DDoS-атак [Електронний ресурс]. – Режим доступу: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-a-ddos-attack (дата звернення: 04.12.2025).
2. Що таке DDoS атака? [Електронний ресурс]. – Режим доступу: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-ataka-ddos/ (дата звернення: 04.12.2025).
3. Застосування інформаційних технологій у діяльності правоохоронних органів [Електронний ресурс]. – Харків, 2020. – Режим доступу: https://dspace.univd.edu.ua/server/api/core/bitstreams/4bd58f7b-1833-453b-81b8-f74486c3b64f/content (дата звернення: 04.12.2025).
4. Типи DDoS-атак та способи захисту від них [Електронний ресурс]. – Режим доступу: https://www.hostzealot.com.ua/blog/about-web-hosting/tipi-ddos-atak-ta-sposobi-zahistu-vid-nih (дата звернення: 04.12.2025).
5. DDoS-атаки: Типи атак та рівні моделі OSI [Електронний ресурс]. – Режим доступу: https://alexhost.com/uk/faq/ddos-attacks-types-of-attacks-and-osi-model-levels/ (дата звернення: 04.12.2025).
6. *Different types of DDoS attacks: how to protect your clients* [Електронний ресурс]. – Режим доступу: https://www.connectwise.com/blog/types-of-ddos-attacks (дата звернення: 04.12.2025).
7. *SYN flood attack* [Електронний ресурс]. – Режим доступу: https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/ (дата звернення: 04.12.2025).
8. *UDP flood attack* [Електронний ресурс]. – Режим доступу: https://www.cloudflare.com/learning/ddos/udp-flood-ddos-attack/ (дата звернення: 04.12.2025).

9. *What is an HTTP Flooding DDoS Attack?* [Електронний ресурс]. *URL*: https://www.netscout.com/what-is-ddos/http-flood-attacks (дата звернення: 04.12.2025).
10. *Types of attacks Azure DDoS Protection mitigate* [Електронний ресурс]. *URL*: https://learn.microsoft.com/en-us/azure/ddos-protection/types-of-attacks (дата звернення: 04.12.2025).
11. *Understanding and Responding to Distributed Denial-of-Service Attacks* [Електронний ресурс]. – CISA. *URL*: https://www.cisa.gov/sites/default/files/publications/understanding-and-responding-to-ddos-attacks_508c.pdf (дата звернення: 04.12.2025).
12. *Defending against distributed denial of service (DDoS) attacks* – ITSM.80.110 [Електронний ресурс]. – Canadian Centre for Cyber Security. *URL*: https://www.cyber.gc.ca/en/guidance/defending-against-distributed-denial-service-ddos-attacks-itsm80110 (дата звернення: 04.12.2025).
13. *Denial of Service (DoS) guidance* [Електронний ресурс]. – NCSC. *URL*: https://www.ncsc.gov.uk/collection/denial-service-dos-guidance-collection (дата звернення: 04.12.2025).
14. *The Importance of DDoS Threat Intelligence and Collaborative Data Sharing* [Електронний ресурс]. *URL*: https://www.a10networks.com/blog/the-importance-of-ddos-threat-intelligence-and-collaborative-data-sharing/ (дата звернення: 04.12.2025).
15. *Why Do Hackers Use DDoS Attacks*? [Електронний ресурс. *URL*: https://www.netscout.com/blog/why-do-hackers-use-ddos-attacks
16. *DDoS Attacks on the Rise: Is Your Organization Prepared*? [Електронний ресурс]. *URL*: https://www.dataversity.net/articles/ddos-attacks-on-the-rise-is-your-organization-prepared/ (дата звернення: 04.12.2025).
17. *Ukraine's case: Building cyber resilience under fire* [Електронний ресурс]. *URL*: https://ega.ee/ukraines-case-building-cyber-resilience-under-fire/ (дата звернення: 04.12.2025).

18. *The Digital Battlefield: How Three Major DDoS Attacks in July 2025 Reveal Evolving Cyber Warfare Tactics* [Електронний ресурс]. *URL*: https://breached.company/the-digital-battlefield-how-three-major-ddos-attacks-in-july-2025-reveal-evolving-cyber-warfare-tactics/ (дата звернення: 04.12.2025).
19. *Key Ukraine Government Organizations Choose Akamai* [Електронний ресурс]. *URL*: https://www.akamai.com/newsroom/customer-announcements/key-ukraine-government-organizations-choose-akamai (дата звернення: 04.12.2025).
20. *What is DDoS mitigation?* [Електронний ресурс]. *URL*: https://www.cloudflare.com/learning/ddos/ddos-mitigation/ (дата звернення: 04.12.2025).
21. *Tavakkoli N., Çetin O., Ekmekcioglu E., Savaş E. From frontlines to online: Examining target preferences in the Russia–Ukraine conflict* [Електронний ресурс]. *URL*: https://d-nb.info/1364576589/34 (дата звернення: 04.12.2025).
22. *The Russia-Ukraine Cyber War Part 2: Attacks Against Government Entities, Defense Sector, and Human Targets* [Електронний ресурс]. *URL*: https://levelblue.com/blogs/spiderlabs-blog/attacks-against-government-entities-defense-sector-and-human-targets (дата звернення: 04.12.2025).
23. *The Digital Battlefield: How Three Major DDoS Attacks in July 2025 Reveal Evolving Cyber Warfare Tactics* [Електронний ресурс]. *URL*: https://breached.company/the-digital-battlefield-how-three-major-ddos-attacks-in-july-2025-reveal-evolving-cyber-warfare-tactics/ (дата звернення: 04.12.2025).
24. *Kyiv Independent suffers DDoS attack after publishing criticism of anti-corruption rollback* [Електронний ресурс]. *URL*: https://kyivindependent.com/kyiv-independent-hit-by-major-ddos-attack-after-criticism-of-law-weakening-anti-graft-agencies/ (дата звернення: 04.12.2025).
25. *Prajapati N. M., Mishra A., Bhanodia P. Literature survey - IDS for DDoS attacks // 2014 Conference on IT in Business, Industry and Government*

(CSIBIG). – Indore, India, 2014. – P. 1–3. – DOI: 10.1109/CSIBIG.2014.7056943.

26. *Singh C., Jain A. K. A comprehensive survey on DDoS attacks detection & mitigation in SDN-IoT network // Prime. – 2024. – Vol. 1. – Article 100543. URL*: https://doi.org/10.1016/j.prime.2024.100543 (дата звернення: 04.12.2025).
27. *Top 7 DDoS Types in 2025 and How to Prevent Them* [Електронний ресурс]. *URL*: https://www.radware.com/cyberpedia/ddos-attacks/top-7-ddos-types-in-2025/ (дата звернення: 04.12.2025).
28. *10 Best DDoS Protection Tools in 2025* [Електронний ресурс]. *URL*: https://www.prophaze.com/blog/best-ddos-protection-tools-in-2025/ (дата звернення: 04.12.2025).
29. *13 Best DDoS Protection Software in the Market 2025* [Електронний ресурс]. *URL*: https://www.indusface.com/blog/best-ddos-protection-software/ (дата звернення: 04.12.2025).
30. Зельонкін Д. О. Дослідження методів побудови нейронних мереж для захисту від DDoS-атак : пояснювальна записка до атестаційної роботи магістра : спец. 121 «Інженерія програмного забезпечення» / Д. О. Зельонкін ; Харків. нац. ун-т радіоелектроніки. – Харків, 2020. – 63 с.
31. Майор Є. Виявлення шкідливих пакетів та DDoS-атак у мережевому трафіку за допомогою глибоких згорткових нейронних мереж // Вимірювальні та обчислювальні пристрої в технологічних процесах. – 2024. – № 77. *URL*: https://doi.org/10.31891/2219-9365-2024-77-41 (дата звернення: 04.12.2025).
32. *Galchynsky L., Graivoronskyi M., Dmytrenko O. Evaluation of machine learning methods to detect DoS / DDoS attacks on IoT // Proceedings of the 2nd International Workshop on Cyber Security and Resilience of IoT Systems (CSR-IoT 2022). – 2022. – Vol. 3241. – P. 235–246. – CEUR Workshop Proceedings. URL*: https://ceur-ws.org/Vol-3241/paper21.pdf (дата звернення: 04.12.2025).

33. *Chukwukelu G., Essien A., Salami A. I., Utuk E. Comparative analysis of machine learning techniques for DDoS intrusion detection in IoT environments // Proceedings of the 21st International Conference on Security and Cryptography (SECRYPT 2024). – 2024. – P. 123–134. URL*: https://www.scitepress.org/Papers/2024/127652/127652.pdf (дата звернення: 04.12.2025).

34. *Abiramasundari S., Ramaswamy V. Distributed denial-of-service (DDOS) attack detection using supervised machine learning algorithms // Scientific Reports. – 2025. – Vol. 15. – Article 13098. URL*: https://doi.org/10.1038/s41598-024-84879-y (дата звернення: 04.12.2025).

35. *Network intrusion detection using hybrid approach / A. Attri, P. Gundeboyena, V. Chigurla et al. // World Journal of Advanced Research and Reviews. – 2025. – Vol. 25, № 02. – P. 507–515. – DOI:* https://doi.org/10.30574/wjarr.2025.25.2.0367 (дата звернення: 04.12.2025).

36. *Analyzing the Evolution of DDoS Attack Detection: Traditional vs. Modern Machine Learning Models / V. Premanand et al. // 2023 6th International Conference on Recent Trends in Advance Computing (ICRTAC). – Chennai, India, 2023. – P. 772–777. – DOI: 10.1109/ICRTAC59277.2023.10480859.*

37. *Kumar M. Oversee effective DDoS detection based on dual frameworks // Computer Networks. – 2025. – Vol. 257. – Article 111863. URL*: https://doi.org/10.1016/j.comnet.2025.111863 (дата звернення: 04.12.2025).

38. *Mahmood A., Avcı İ. Evaluation and benchmarking of cybersecurity DDoS attacks detection models through the integration of FWZIC and MABAC methods // Computer Systems Science and Engineering. – 2025. – Vol. 49, № 1. – P. 401–417. URL*: https://doi.org/10.32604/csse.2025.062413 (дата звернення: 04.12.2025).

39. *What are the pros and cons of signature-based vs. anomaly-based detection?* [Електронний ресурс]. – LinkedIn Advice. *URL*: https://www.linkedin.com/advice/0/what-pros-cons-signature-based-vs-anomaly-based (дата звернення: 04.12.2025).

40. *Intrusion Detection System (IDS): Signature vs. Anomaly-Based* [Електронний ресурс]. – N-able. *URL*: https://www.n-able.com/blog/intrusion-detection-system (дата звернення: 04.12.2025).
41. *P. R., Kamalakkannan S. A hybrid machine learning model with improved feature set for DDoS attack detection under bigdata perspective // Information Security Journal: A Global Perspective. – 2025. – P. 1–30. URL*: https://doi.org/10.1080/19393555.2025.2589766 (дата звернення: 04.12.2025).
42. *Distributed denial of service attacks against cloud computing environment: Survey, issues, challenges and coherent taxonomy / Z. R. Alashhab et al. // Applied Sciences. – 2022. – Vol. 12, № 23. – Article 12441. URL*: https://doi.org/10.3390/app122312441 (дата звернення: 04.12.2025).
43. *DDoS attack detection techniques in IoT networks: a survey / A. Pakmehr et al. // Cluster Computing. – 2024. – Vol. 27. – P. 14637–14668. URL*: https://doi.org/10.1007/s10586-024-04662-6 (дата звернення: 04.12.2025).
44. *Intrusion Detection System (IDS) [Електронний ресурс]. – GeeksforGeeks. URL*: https://www.geeksforgeeks.org/ethical-hacking/intrusion-detection-system-ids/ (дата звернення: 04.12.2025).
45. *Mahdi Z. S., Zaki R. M., Alzubaidi L. Advanced hybrid techniques for cyberattack detection and defense in IoT networks // Security and Privacy. – 2024. – Vol. 7, № 5. – e471. URL*: https://doi.org/10.1002/spy2.471 (дата звернення: 04.12.2025).
46. *Mittal M., Kumar K., Behal S. Deep learning approaches for detecting DDoS attacks: a systematic review // Soft Computing. – 2022. – P. 1–37. URL*: https://doi.org/10.1007/s00500-021-06608-1 (дата звернення: 04.12.2025).
47. *Batool S., Aslam M., Akpokodje E., Jilani S. F. A Comprehensive Review of DDoS Detection and Mitigation in SDN Environments: Machine Learning, Deep Learning, and Federated Learning Perspectives // Electronics. – 2025. – Vol. 14, № 21. – 4222. URL*: https://doi.org/10.3390/electronics14214222 (дата звернення: 04.12.2025).

48. *Detecting and mitigating DDoS attacks with AI: A survey / A. Apostu et al. // arXiv preprint arXiv:2503.17867. – 2025. URL*: https://doi.org/10.48550/arXiv.2503.17867 (дата звернення: 04.12.2025).
49. *Najafimehr M., Zarifzadeh S., Mostafavi S. DDoS attacks and machine-learning-based detection methods: A survey and taxonomy // Engineering Reports. – 2023. – Vol. 5, № 5. – e12697. URL*: https://doi.org/10.1002/eng2.12697 (дата звернення: 04.12.2025).
50. *Chen S.-R., Chen S.-J., Hsieh W.-B. Enhancing Machine Learning-Based DDoS Detection Through Hyperparameter Optimization // Electronics. – 2025. – Vol. 14, № 16. – 3319. URL*: https://doi.org/10.3390/electronics14163319 (дата звернення: 04.12.2025).
51. *Advancing DDoS attack detection: A synergistic approach using deep residual neural networks and synthetic oversampling / A. Alfatemi et al. // arXiv preprint arXiv:2401.03116. – 2024. URL*: https://doi.org/10.48550/arXiv.2401.03116 (дата звернення: 04.12.2025).
52. *A DDoS Detection Method Based on Feature Engineering and Machine Learning in Software-Defined Networks / Z. Liu et al. // Sensors. – Basel, 2023. – Vol. 23, № 13. – 6176. URL*: https://doi.org/10.3390/s23136176 (дата звернення: 04.12.2025).

ДОДАТОК А

Перелік необхідних бібліотек (requirements.txt)

```
matplotlib>=3.5.0
scapy>=2.4.5
numpy>=1.21.0
scikit-learn>=1.0.0
pandas>=1.3.0
psutil>=5.8.0
```

ДОДАТОК Б

Лістинг main.py

```
"""
Головний файл системи виявлення DDoS-атак
"""
import sys
import os
import tkinter as tk
from tkinter import messagebox
import traceback
import platform
import time
import warnings

# Додати поточну директорію до шляху Python
sys.path.append(os.path.dirname(os.path.abspath(__file__)))

# Ігнорувати попередження
warnings.filterwarnings('ignore')

def check_dependencies():
    """Перевірити наявність необхідних бібліотек"""
    print("\n[CHECK] ПЕРЕВІРКА ЗАЛЕЖНОСТЕЙ")
    print("=" * 40)

    required_packages = [
        ('matplotlib', 'matplotlib'),
        ('scapy', 'scapy'),
        ('numpy', 'numpy'),
        ('scikit-learn', 'sklearn'),  # pip install scikit-learn, import sklearn
        ('pandas', 'pandas'),
        ('psutil', 'psutil'),
    ]

    missing_packages = []

    for package_name, import_name in required_packages:
        try:
            __import__(import_name)
```

```
            print(f"  [OK] {package_name}")
        except ImportError:
            missing_packages.append(package_name)
            print(f"  [FAIL] {package_name}")

    return missing_packages

def install_missing_packages(packages):
    """Спробувати встановити відсутні пакети"""
    import subprocess
    import sys

    print("\n[INSTALL] ВСТАНОВЛЕННЯ ВІДСУТНІХ ПАКЕТІВ")
    print("=" * 40)

    # Спеціальна обробка для scikit-learn
    if 'scikit-learn' in packages:
        packages.remove('scikit-learn')
        packages.append('scikit-learn')

    for package in packages:
        try:
            print(f"[INSTALL] Встановлення {package}...")
            result = subprocess.run(
                [sys.executable, "-m", "pip", "install", package],
                capture_output=True,
                text=True,
                timeout=60
            )

            if result.returncode == 0:
                print(f"  [OK] {package} встановлено")
            else:
                # Спеціальна спроба для scikit-learn
                if 'scikit-learn' in result.stdout or 'scikit-learn' in
result.stderr:
                    print(f"  [INFO] Спеціальна спроба для scikit-learn...")
                    subprocess.run([sys.executable, "-m", "pip", "install",
"scikit-learn", "--no-deps"])
                    print(f"  [OK] scikit-learn встановлено")
                else:
                    print(f"  [ERROR] Помилка: {result.stderr[:100]}")
```

```
                    return False

            except subprocess.TimeoutExpired:
                print(f"  [TIMEOUT] Таймаут при встановленні {package}")
            except Exception as e:
                print(f"  [ERROR] Помилка: {e}")
                return False

    return True

def setup_directories():
    """Створити необхідні директорії"""
    directories = ['reports', 'logs', 'models', 'data']

    print("\n[SETUP] СТВОРЕННЯ ДИРЕКТОРІЙ")
    print("=" * 40)

    for directory in directories:
        try:
            os.makedirs(directory, exist_ok=True)
            print(f"  [DIR] {directory}")
        except:
            print(f"  [WARNING] Не вдалося створити {directory}")

def display_welcome():
    """Показати вітальне повідомлення"""
    print("\n" + "=" * 60)
    print("[START] СИСТЕМА ВИЯВЛЕННЯ DDoS-АТАК v3.0")
    print("=" * 60)
    print(f"[INFO] ОС: {platform.system()} {platform.release()}")
    print(f"[INFO] Python: {platform.python_version()}")
    print("=" * 60)

def load_modules():
    """Завантажити модулі з обробкою помилок"""
    modules = {}

    try:
        print("\n[LOAD] ЗАВАНТАЖЕННЯ МОДУЛІВ")
        print("=" * 40)

        try:
```

```
            from config import NETWORK_CONFIG, ML_CONFIG
            modules['config'] = {
                'NETWORK_CONFIG': NETWORK_CONFIG,
                'ML_CONFIG': ML_CONFIG
            }
            print("  [OK] config")
        except ImportError:
            print("      [WARNING]   config:   використовуються   значення   за
замовчуванням")
            modules['config'] = {
                'NETWORK_CONFIG': {'sample_duration': 2.0, 'thresholds': {}},
                'ML_CONFIG': {'enabled': False}
            }

        # network_monitor
        try:
            from network_monitor import NetworkMonitor
            modules['NetworkMonitor'] = NetworkMonitor
            print("  [OK] network_monitor")
        except ImportError as e:
            print(f"  [FAIL] network_monitor: {e}")
            print(f"  [ERROR] Встановіть scapy: pip install scapy")
            return None

        # analyzer або ml_detector
        config = modules.get('config', {})
        ml_config = config.get('ML_CONFIG', {})
        use_ml = ml_config.get('enabled', False)

        if use_ml:
            try:
                from ml_detector import MLDDoSDetector
                modules['DDOSAnalyzer'] = MLDDoSDetector
                print("  [OK] ml_detector (ML режим)")
            except ImportError as e:
                print(f"  [FAIL] ml_detector: {e}")
                print("  [INFO] Перемикання на простий аналізатор")
                use_ml = False

        if not use_ml:
            try:
                from analyzer import DDOSAnalyzer
```

```
                modules['DDOSAnalyzer'] = DDOSAnalyzer
                print("  [OK] analyzer (Simple режим)")
            except ImportError as e:
                print(f"  [FAIL] analyzer: {e}")
                print(f"  [ERROR] Відсутній модуль аналізатора")
                return None

        # gui
        try:
            from gui import DDOSDetectorGUI
            modules['DDOSDetectorGUI'] = DDOSDetectorGUI
            print("  [OK] gui")
        except ImportError as e:
            print(f"  [FAIL] gui: {e}")
            print("  [ERROR] GUI не знайдено! Перевірте наявність файлу gui.py")
            return None

        print("\n[OK] Всі модулі завантажено")
        return modules

    except Exception as e:
        print(f"\n[ERROR] Критична помилка завантаження: {e}")
        return None

def select_interface_simple():
    """Простий вибір інтерфейсу через консоль"""
    print("\n[SELECT] ВИБІР МЕРЕЖЕВОГО ІНТЕРФЕЙСУ")
    print("=" * 40)

    try:
        # Створити тимчасовий монітор для отримання списку
        network_monitor_module = sys.modules.get('network_monitor', None)
        if network_monitor_module and hasattr(network_monitor_module,
'NetworkMonitor'):
            monitor = network_monitor_module.NetworkMonitor()
        else:
            # Створити демо-монітор
            class TempMonitor:
                def get_available_interfaces_list(self):
                    return [
                        {'name': 'eth0', 'type': 'Ethernet', 'status':
'активний'},
```

```
                        {'name': 'wlan0', 'type': 'Wi-Fi', 'status': 'активний'},
                        {'name': 'lo', 'type': 'Loopback', 'status': 'активний'},
                        {'name':   'demo',   'type':   'Демо-режим',   'status':
'активний'}
                    ]
            monitor = TempMonitor()

        interfaces = monitor.get_available_interfaces_list()

        if not interfaces:
            print("  [ERROR] Не знайдено мережевих інтерфейсів")
            return "demo"

        print("\nДоступні інтерфейси:")
        for i, iface in enumerate(interfaces, 1):
            status_icon = "[OK]" if 'актив' in iface.get('status', '').lower()
else "[--]"
            print(f"   {i}. {status_icon} {iface['name']} - {iface.get('type',
'Невідомо')}")

        print(f"  {len(interfaces)+1}. Демо-режим (без реального трафіку)")

        try:
            choice = input("\nВиберіть номер (Enter для авто-вибору): ").strip()

            if choice == "":
                # Автоматичний вибір - шукаємо Ethernet, потім Wi-Fi
                for iface in interfaces:
                    if   iface.get('type')   ==   'Ethernet'   and   'актив'   in
iface.get('status', '').lower():
                        print(f"\n[OK] Обрано Ethernet: {iface['name']}")
                        return iface['name']

                for iface in interfaces:
                    if   iface.get('type')   ==   'Wi-Fi'   and   'актив'   in
iface.get('status', '').lower():
                        print(f"\n[OK] Обрано Wi-Fi: {iface['name']}")
                        return iface['name']

                # Якщо нічого не знайдено
                print("\n[WARNING] Обрано перший доступний інтерфейс")
                return interfaces[0]['name']
```

```
            elif choice.isdigit():
                idx = int(choice) - 1
                if 0 <= idx < len(interfaces):
                    print(f"\n[OK] Обрано: {interfaces[idx]['name']}")
                    return interfaces[idx]['name']
                elif idx == len(interfaces):
                    print("\n[OK] Режим: Демо")
                    return "demo"

            print("\n[WARNING] Невірний вибір, використовую авто-вибір")
            return interfaces[0]['name'] if interfaces else "demo"

        except (KeyboardInterrupt, EOFError):
            print("\n[WARNING] Використовую авто-вибір")
            return interfaces[0]['name'] if interfaces else "demo"

    except Exception as e:
        print(f"\n[WARNING] Помилка вибору: {e}")
        return "demo"

def main():
    """Головна функція"""
    try:
        # Показати вітання
        display_welcome()

        # Створити директорії
        setup_directories()

        # Перевірити залежності
        missing = check_dependencies()

        if missing:
            print(f"\n[WARNING] Відсутні пакети: {', '.join(missing)}")

            # Не запитуємо встановлення, щоб уникнути проблем
            print("   Запускаю в демо-режимі...")
            print("   Для повної функціональності встановіть:")
            print(f"   pip install {' '.join(missing)}")

        # Завантажити модулі
```

```
        modules = load_modules()

        if not modules or 'DDOSDetectorGUI' not in modules:
            messagebox.showerror(
                "Критична помилка",
                "Не вдалося завантажити графічний інтерфейс.\n"
                "Переконайтеся, що файл gui.py існує в поточній директорії."
            )
            return

        # Вибрати інтерфейс
        selected_iface = select_interface_simple()

        # Створити монітор
        try:
            monitor = modules['NetworkMonitor'](interface=selected_iface)
            print(f"\n[OK] Ініціалізовано монітор для: {monitor.interface}")
        except Exception as e:
            print(f"\n[WARNING] Помилка монітора: {e}")
            monitor = modules['NetworkMonitor']()

        # Створити аналізатор
        try:
            config = modules.get('config', {})
            ml_config = config.get('ML_CONFIG', {})

            # Перевірити чи це ML клас чи простий
            analyzer_class = modules['DDOSAnalyzer']
            is_ml_class = analyzer_class.__name__ == 'MLDDoSDetector'

            if is_ml_class:
                # ML режим
                analyzer = analyzer_class(models_dir=ml_config.get('models_dir',
'models'))
                print(f"[OK] Ініціалізовано ML аналізатор")

                # Спроба завантажити моделі
                if not analyzer.load_models():
                    print("\n[WARNING] ML моделі не знайдено!")
                    print("  Запустіть: python training.py")
                    print("    Або встановіть ML_CONFIG['enabled'] = False в
config.py\n")
```

```
            else:
                # Простий режим
                thresholds = config.get('NETWORK_CONFIG', {}).get('thresholds',
{})
                analyzer = analyzer_class(thresholds=thresholds)
                print(f"[OK] Ініціалізовано простий аналізатор")
        except Exception as e:
            print(f"[WARNING] Помилка аналізатора: {e}")
            thresholds = config.get('NETWORK_CONFIG', {}).get('thresholds', {})
            analyzer = modules['DDOSAnalyzer'](thresholds=thresholds)

        # Запустити GUI
        print("\n[GUI] ЗАПУСК ГРАФІЧНОГО ІНТЕРФЕЙСУ")
        print("=" * 40)

        try:
            root = tk.Tk()
            root.title("Система виявлення DDoS-атак v3.0")
            root.geometry("1400x900")
            root.minsize(1000, 700)

            # Створити GUI
            gui_class = modules['DDOSDetectorGUI']
            app = gui_class(root, monitor, analyzer)

            # Додати обробник закриття
            if hasattr(app, 'on_closing'):
                root.protocol("WM_DELETE_WINDOW", app.on_closing)

            print("\n[OK] СИСТЕМА УСПІШНО ЗАПУЩЕНА!")
            print("=" * 60)
            print("\n[INFO] ІНСТРУКЦІЯ:")
            print("1. Натисніть 'ПОЧАТИ МОНІТОРИНГ'")
            print("2. Використовуйте тестові атаки для перевірки")
            print("3. Спостерігайте за статистикою у реальному часі")
            print("=" * 60)

            root.mainloop()

        except Exception as e:
            print(f"[ERROR] Помилка GUI: {e}")
            traceback.print_exc()
```

```
            messagebox.showerror(
                "Помилка запуску",
                f"Не вдалося запустити графічний інтерфейс:\n{str(e)[:200]}\n\n"
                "Детальна інформація у консолі."
            )

    except KeyboardInterrupt:
        print("\n\n[EXIT] Програму зупинено користувачем")
    except Exception as e:
        print(f"\n[ERROR] Критична помилка: {e}")
        traceback.print_exc()
        messagebox.showerror(
            "Критична помилка",
            f"Сталася критична помилка:\n{str(e)[:200]}\n\n"
            "Перевірте консоль для детальної інформації."
        )

if __name__ == "__main__":
    # Перевірити права
    if os.name == 'posix' and os.geteuid() != 0:
        print("[WARNING] Для повної функціональності запустіть з sudo")

    # Запустити
    main()
```

ДОДАТОК В

Лістинг ml_detector.py

```
"""
ML детектор DDoS-атак з використанням Isolation Forest та Random Forest
"""
import numpy as np
import pickle
import os
from sklearn.ensemble import IsolationForest, RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from datetime import datetime


class MLDDoSDetector:
    """ML детектор з двома моделями"""

    def __init__(self, models_dir="models"):
        self.models_dir = models_dir
        self.isolation_forest = None
        self.random_forest = None
        self.scaler = None
        self.feature_names = [
            "packet_rate", "byte_rate", "syn_count", "udp_count",
            "icmp_count",          "tcp_count",          "src_ip_entropy",
"dst_port_entropy",
            "unique_src_ips", "unique_dst_ports", "avg_packet_size"
        ]
        self.attack_types = ["Normal", "SYN Flood", "UDP Flood", "ICMP
Flood", "HTTP Flood"]

        # Статистика
        self.analysis_count = 0
        self.attack_count = 0
        self.total_confidence = 0.0
        self.attack_history = []

        print("[OK] ML Детектор ініціалізовано")

    def train_isolation_forest(self, X_train, contamination=0.1):
```

```
            """Тренування Isolation Forest (виявлення аномалій)"""
            print(f"\n[TRAIN] Isolation Forest (contamination={contamination})...")

            self.isolation_forest = IsolationForest(
                contamination=contamination,
                n_estimators=100,
                max_samples=256,
                random_state=42,
                n_jobs=-1
            )

            self.isolation_forest.fit(X_train)

            # Оцінка на тренувальних даних
            predictions = self.isolation_forest.predict(X_train)
            anomalies = np.sum(predictions == -1)

            print(f"[OK] Isolation Forest навчено")
            print(f"     Виявлено аномалій: {anomalies}/{len(X_train)} ({anomalies/len(X_train):.1%})")

            return self.isolation_forest

        def train_random_forest(self, X_train, y_train):
            """Тренування Random Forest (класифікація)"""
            print(f"\n[TRAIN] Random Forest Classifier...")

            self.random_forest = RandomForestClassifier(
                n_estimators=100,
                max_depth=20,
                min_samples_split=5,
                min_samples_leaf=2,
                random_state=42,
                n_jobs=-1
            )

            self.random_forest.fit(X_train, y_train)

            # Оцінка точності
            score = self.random_forest.score(X_train, y_train)
```

```
        print(f"[OK] Random Forest навчено")
        print(f"     Точність на тренуванні: {score:.1%}")

        return self.random_forest

    def train_scaler(self, X_train):
        """Тренування нормалізатора"""
        print(f"\n[TRAIN] StandardScaler...")

        self.scaler = StandardScaler()
        self.scaler.fit(X_train)

        print(f"[OK] Scaler навчено")

        return self.scaler

    def save_models(self):
        """Зберегти моделі у файли"""
        os.makedirs(self.models_dir, exist_ok=True)

        if self.isolation_forest:
            path = os.path.join(self.models_dir, "isolation_forest.pkl")
            with open(path, "wb") as f:
                pickle.dump(self.isolation_forest, f)
            print(f"[SAVE] {path}")

        if self.random_forest:
            path = os.path.join(self.models_dir, "ddos_model.pkl")
            with open(path, "wb") as f:
                pickle.dump(self.random_forest, f)
            print(f"[SAVE] {path}")

        if self.scaler:
            path = os.path.join(self.models_dir, "scaler.pkl")
            with open(path, "wb") as f:
                pickle.dump(self.scaler, f)
            print(f"[SAVE] {path}")

        print("[OK] Моделі збережено")

    def load_models(self):
        """Завантажити моделі з файлів"""
```

```
            try:
                iso_path = os.path.join(self.models_dir,
"isolation_forest.pkl")
                if os.path.exists(iso_path):
                    with open(iso_path, "rb") as f:
                        self.isolation_forest = pickle.load(f)
                    print(f"[LOAD] {iso_path}")

                rf_path = os.path.join(self.models_dir, "ddos_model.pkl")
                if os.path.exists(rf_path):
                    with open(rf_path, "rb") as f:
                        self.random_forest = pickle.load(f)
                    print(f"[LOAD] {rf_path}")

                scaler_path = os.path.join(self.models_dir, "scaler.pkl")
                if os.path.exists(scaler_path):
                    with open(scaler_path, "rb") as f:
                        self.scaler = pickle.load(f)
                    print(f"[LOAD] {scaler_path}")

                if self.isolation_forest and self.random_forest and
self.scaler:
                    print("[OK] Моделі завантажено успішно")
                    return True
                else:
                    print("[WARNING] Не всі моделі завантажено")
                    return False

            except Exception as e:
                print(f"[ERROR] Помилка завантаження моделей: {e}")
                return False

        def extract_features(self, stats):
            """Витягнути ознаки з статистики трафіку"""
            features = []

            features.append(stats.get("пакетів/с", stats.get("packet_rate",
0)))
            features.append(stats.get("байт/с", stats.get("byte_rate", 0)))
            features.append(stats.get("SYN/с", stats.get("syn_count", 0)))
            features.append(stats.get("UDP", stats.get("udp_count", 0)))
            features.append(stats.get("ICMP", stats.get("icmp_count", 0)))
```

```
            features.append(stats.get("TCP", stats.get("tcp_count", 0)))
            features.append(stats.get("ентропія                    IP",
stats.get("src_ip_entropy", 0)))
            features.append(stats.get("ентропія                  портів",
stats.get("dst_port_entropy", 0)))
            features.append(stats.get("унік.  IP",  stats.get("unique_src_ips",
0)))
            features.append(stats.get("унік.                     порти",
stats.get("unique_dst_ports", stats.get("унік. IP", 0))))
            features.append(stats.get("середній        розмір        пакета",
stats.get("avg_packet_size", 500)))

            return np.array(features).reshape(1, -1)

        def analyze_traffic(self, stats):
            """Проаналізувати трафік за допомогою ML"""
            if  not  stats  or  stats.get("status")  in  ["неактивний",  "немає
даних"]:
                return {
                    "is_attack": False,
                    "attack_type": "Немає даних",
                    "confidence": 0.0,
                    "severity": "НИЗЬКА",
                    "indicators": [],
                    "ml_anomaly_score": 0.0,
                    "ml_prediction": "Normal"
                }

            if   not   (self.isolation_forest   and   self.random_forest   and
self.scaler):
                print("[WARNING] ML моделі не завантажено, використовується
простий аналіз")
                return self._simple_analysis(stats)

            try:
                self.analysis_count += 1

                # Витягнути ознаки
                features = self.extract_features(stats)

                # Нормалізувати
                features_scaled = self.scaler.transform(features)
```

```
                # 1. Isolation Forest (виявлення аномалій)
                anomaly_pred = self.isolation_forest.predict(features_scaled)[0]
                anomaly_score = -self.isolation_forest.score_samples(features_scaled)[0]
                is_anomaly = anomaly_pred == -1

                # 2. Random Forest (класифікація)
                attack_pred = self.random_forest.predict(features_scaled)[0]
                attack_proba = self.random_forest.predict_proba(features_scaled)[0]
                attack_confidence = np.max(attack_proba)

                # Комбінована логіка
                is_attack = is_anomaly or attack_pred == 1

                if is_attack:
                    # Визначити тип атаки
                    if attack_pred == 1:
                        # Random Forest передбачив атаку - використати його класифікацію
                        attack_type = self._determine_attack_type_ml(stats, features)
                    else:
                        # Тільки Isolation Forest - простий аналіз
                        attack_type = self._determine_attack_type_simple(stats)

                    # Впевненість: комбінація двох моделей
                    if is_anomaly and attack_pred == 1:
                        confidence = min(anomaly_score * 0.5 + attack_confidence * 0.5, 1.0)
                    elif is_anomaly:
                        confidence = min(anomaly_score * 0.8, 1.0)
                    else:
                        confidence = attack_confidence

                    severity = self._determine_severity(confidence, stats)
                    indicators = self._generate_indicators(stats, anomaly_score)
```

```
                self.attack_count += 1
            else:
                attack_type = "Normal"
                confidence = 0.0
                severity = "НИЗЬКА"
                indicators = []

            self.total_confidence += confidence

            return {
                "is_attack": is_attack,
                "attack_type": attack_type,
                "confidence": confidence,
                "severity": severity,
                "indicators": indicators,
                "ml_anomaly_score": float(anomaly_score),
                "ml_prediction": "Attack" if attack_pred == 1 else "Normal",
                "ml_confidence": float(attack_confidence)
            }

        except Exception as e:
            print(f"[ERROR] ML аналіз: {e}")
            return self._simple_analysis(stats)

    def _determine_attack_type_ml(self, stats, features):
        """Визначити тип атаки на основі ознак"""
        syn = stats.get("SYN/c", 0)
        udp = stats.get("UDP", 0)
        icmp = stats.get("ICMP", 0)
        tcp = stats.get("TCP", 0)

        if syn > 30:
            return "SYN Flood"
        elif udp > 150:
            return "UDP Flood"
        elif icmp > 80:
            return "ICMP Flood"
        elif tcp > 400:
            return "HTTP Flood"
        else:
            return "DDoS Attack"
```

```
def _determine_attack_type_simple(self, stats):
    """Простий метод визначення типу атаки"""
    syn = stats.get("SYN/с", 0)
    udp = stats.get("UDP", 0)
    icmp = stats.get("ICMP", 0)

    if syn > 30:
        return "SYN Flood"
    elif udp > 150:
        return "UDP Flood"
    elif icmp > 80:
        return "ICMP Flood"
    else:
        return "DDoS Attack"

def _determine_severity(self, confidence, stats):
    """Визначити серйозність"""
    pkt_rate = stats.get("пакетів/с", 0)

    if confidence > 0.85 and pkt_rate > 3000:
        return "КРИТИЧНА"
    elif confidence > 0.7 and pkt_rate > 1500:
        return "ВИСОКА"
    elif confidence > 0.5:
        return "ПОМІРНА"
    else:
        return "НИЗЬКА"

def _generate_indicators(self, stats, anomaly_score):
    """Генерувати індикатори атаки"""
    indicators = []
    indicators.append(f"ML аномалія: {anomaly_score:.3f}")

    pkt_rate = stats.get("пакетів/с", 0)
    if pkt_rate > 1000:
        indicators.append(f"Висока швидкість: {pkt_rate:.0f} пак/с")

    entropy = stats.get("ентропія IP", 0)
    if entropy < 2.0:
        indicators.append(f"Низька ентропія IP: {entropy:.2f}")

    return indicators
```

```
def _simple_analysis(self, stats):
    """Простий аналіз без ML"""
    pkt_rate = stats.get("пакетів/с", 0)
    syn_rate = stats.get("SYN/с", 0)

    is_attack = pkt_rate > 1000 or syn_rate > 50

    return {
        "is_attack": is_attack,
        "attack_type": "Suspected" if is_attack else "Normal",
        "confidence": 0.5 if is_attack else 0.0,
        "severity": "ПОМІРНА" if is_attack else "НИЗЬКА",
        "indicators": ["Простий аналіз (ML недоступний)"],
        "ml_anomaly_score": 0.0,
        "ml_prediction": "N/A"
    }

def get_analysis_summary(self):
    """Отримати зведену статистику"""
    avg_conf = self.total_confidence / max(self.analysis_count, 1)
    return {
        "total_analyses": self.analysis_count,
        "attack_count": self.attack_count,
        "avg_confidence": avg_conf
    }

def get_attack_history(self, limit=50):
    """Отримати історію атак"""
    return self.attack_history[-limit:] if self.attack_history else []
```

ДОДАТОК Г

Лістинг network_monitor.py

```
"""
network_monitor.py
Остаточна робоча версія (v3.1)
- Виправлено для сумісності з оновленим main.py
- Додано метод get_analysis_summary()
- Виправлено невеликі помилки
"""
import time
import threading
import socket
from collections import defaultdict, deque
import psutil
from scapy.all import sniff, IP, TCP, UDP, ICMP
import numpy as np
from datetime import datetime


class NetworkMonitor:
    def __init__(self, interface=None, sample_duration=10):
        self.sample_duration = sample_duration
        self.running = False
        self.interface = interface or self._auto_detect_interface()
        self.available_interfaces = self._get_available_interfaces()

        self.stats = {
            'packet_count': 0,
            'byte_count': 0,
            'packet_types': defaultdict(int),
            'src_ips': defaultdict(int),
            'dst_ports': defaultdict(int),
            'syn_count': 0,
            'connections': defaultdict(set),
            'timestamps': deque(maxlen=20000)
        }

        # Додано: історія для очищення старих даних
```

```
            self.packet_history = deque(maxlen=20000)  # (timestamp, type, size,
syn_flag)
            self.last_cleanup = time.time()
            self.cleanup_interval = 5  # Очищати кожні 5 секунд

            self.lock = threading.Lock()

            print(f"[OK] Ініціалізовано NetworkMonitor")
            print(f"[INFO] Інтерфейс: {self.interface}")

            # Додано для сумісності
            self.capture_thread = None
            self.monitor_thread = None

        def start_capture(self):
            """Почати захоплення трафіку"""
            if self.running:
                print("[WARNING] Моніторинг вже запущений")
                return

            if not self._is_interface_active(self.interface):
                print("[WARNING] Інтерфейс неактивний — шукаємо кращий...")
                if not self.auto_switch_best_interface():
                    print("[ERROR] Не вдалося знайти активний інтерфейс")
                    return

            self.running = True
            self.capture_thread                                              =
threading.Thread(target=self._capture_packets, daemon=True)
            self.capture_thread.start()

            self.monitor_thread                                              =
threading.Thread(target=self._monitor_status, daemon=True)
            self.monitor_thread.start()

            print(f"[START] Моніторинг запущено на {self.interface}")

        def stop_capture(self):
            """Зупинити захоплення трафіку"""
            print("\n[STOP] Зупинка моніторингу...")
            self.running = False
```

```
            # Чекаємо завершення потоків
            if self.capture_thread and self.capture_thread.is_alive():
                self.capture_thread.join(timeout=3)

            if self.monitor_thread and self.monitor_thread.is_alive():
                self.monitor_thread.join(timeout=3)

            print("[OK] Моніторинг зупинено")

        def get_current_stats(self):
            """Отримати поточну статистику"""
            with self.lock:
                # Очистити стару статистику
                self._cleanup_old_stats()

                if not self.running:
                    return {"status": "неактивний"}

                if not self.packet_history:
                    return {"status": "немає даних"}

                now = time.time()
                window = now - self.sample_duration

                # Підрахувати статистику тільки за останнє вікно часу
                recent_packets = [p for p in self.packet_history if p[0] >=
window]

                if not recent_packets:
                    return {"status": "немає даних"}

                # Підрахунок метрик
                total_packets = len(recent_packets)
                total_bytes = sum(p[2] for p in recent_packets)
                tcp_count = sum(1 for p in recent_packets if p[1] == 'TCP')
                udp_count = sum(1 for p in recent_packets if p[1] == 'UDP')
                icmp_count = sum(1 for p in recent_packets if p[1] == 'ICMP')
                syn_count = sum(1 for p in recent_packets if p[3])

                if recent_packets:
                    actual_duration = now - min(p[0] for p in recent_packets)
                    effective_duration = max(actual_duration, 1.0)
```

```
            else:
                effective_duration = self.sample_duration

            pkt_rate = total_packets / effective_duration
            byte_rate = total_bytes / effective_duration
            kb_rate = byte_rate / 1024
            syn_rate = syn_count / effective_duration
            avg_packet_size = total_bytes / max(total_packets, 1)

            recent_src_ips = defaultdict(int)
            recent_dst_ports = defaultdict(int)

            for p in recent_packets:
                if len(p) > 4:
                    recent_src_ips[p[4]] += 1
                    if len(p) > 5:
                        recent_dst_ports[p[5]] += 1

            src_entropy = self._entropy(list(recent_src_ips.values()))
            port_entropy = self._entropy(list(recent_dst_ports.values()))

            return {
                'timestamp': datetime.now().strftime("%H:%M:%S"),
                'interface': self.interface,
                'type': self._determine_interface_type(self.interface),
                'пакетів/с': round(pkt_rate, 2),
                'КБ/с': round(kb_rate, 2),
                'байт/с': round(byte_rate, 2),
                'середній розмір пакета': round(avg_packet_size, 0),
                'TCP': tcp_count,
                'UDP': udp_count,
                'ICMP': icmp_count,
                'SYN/с': round(syn_rate, 2),
                'унік. IP': len(recent_src_ips),
                'унік. порти': len(recent_dst_ports),
                'ентропія IP': round(src_entropy, 3),
                'ентропія портів': round(port_entropy, 3),
            }

    def get_analysis_summary(self):
        """Отримати зведену статистику (для сумісності)"""
        return {
```

```
            'total_analyses': self.stats['packet_count'],
            'attack_count': 0,
            'avg_confidence': 0,
            'interface': self.interface,
            'packet_count': self.stats['packet_count'],
            'byte_count': self.stats['byte_count']
        }

    # ========== Допоміжні методи ==========

    def _auto_detect_interface(self):
        """Автоматично вибрати інтерфейс"""
        return self._find_best_interface() or "demo"

    def _find_best_interface(self):
        """Знайти найкращий інтерфейс"""
        self.available_interfaces = self._get_available_interfaces()

        # 1. Ethernet — найвищий пріоритет
        for iface in self.available_interfaces:
            if (iface['type'] == 'Ethernet' and
                iface['status'] in ['активний', 'підключено'] and
                iface['ipv4'] and
                '127.0.0.1' not in iface['ipv4']):
                print(f"[OK] Обрано Ethernet: {iface['name']}")
                return iface['name']

        # 2. Wi-Fi
        for iface in self.available_interfaces:
            if (iface['type'] == 'Wi-Fi' and
                iface['status'] in ['активний', 'підключено'] and
                iface['ipv4'] and
                '127.0.0.1' not in iface['ipv4']):
                print(f"[OK] Обрано Wi-Fi: {iface['name']}")
                return iface['name']

        # 3. Будь-який інший
        for iface in self.available_interfaces:
            if iface['ipv4'] and '127.0.0.1' not in iface['ipv4']:
                return iface['name']

        return None
```

```
        def _get_available_interfaces(self):
            """Отримати список доступних інтерфейсів"""
            interfaces = []
            try:
                for name, addrs in psutil.net_if_addrs().items():
                    if name in ['lo', 'lo0']:
                        continue
                    ipv4 = [a.address for a in addrs if a.family ==
socket.AF_INET]
                    status = self._get_interface_status(name)
                    iface_type = self._determine_interface_type(name)
                    interfaces.append({
                        'name': name,
                        'type': iface_type,
                        'status': status,
                        'ipv4': ipv4,
                        'addresses': ipv4  # Для сумісності
                    })
            except Exception as e:
                print(f"[WARNING] Помилка сканування інтерфейсів: {e}")
                # Демо-дані
                interfaces = [
                    {'name': 'eth0', 'type': 'Ethernet', 'status': 'активний',
'ipv4': ['192.168.1.100'], 'addresses': ['192.168.1.100']},
                    {'name': 'wlan0', 'type': 'Wi-Fi', 'status': 'активний',
'ipv4': ['192.168.1.101'], 'addresses': ['192.168.1.101']},
                    {'name': 'demo', 'type': 'Демо', 'status': 'активний',
'ipv4': ['127.0.0.1'], 'addresses': ['127.0.0.1']},
                ]
            return interfaces

        def get_available_interfaces_list(self):
            """Отримати список інтерфейсів (для GUI)"""
            return self._get_available_interfaces()

        def _determine_interface_type(self, name):
            """Визначити тип інтерфейсу"""
            n = name.lower()
            if any(k in n for k in ['wi-fi', 'wifi', 'wlan', 'wlp', 'wlo']):
                return 'Wi-Fi'
```

```
            if any(k in n for k in ['ethernet', 'eth', 'enp', 'ens', 'eno',
'lan']):
                return 'Ethernet'
            if any(k in n for k in ['virtual', 'vmware', 'vbox', 'tun', 'tap',
'vpn']):
                return 'Virtual'
            if 'demo' in n:
                return 'Демо-режим'
            return 'Невідомо'

        def _get_interface_status(self, name):
            """Отримати статус інтерфейсу"""
            try:
                stats = psutil.net_io_counters(pernic=True).get(name)
                if stats and (stats.bytes_sent > 100 or stats.bytes_recv > 100):
                    return 'активний'
                for addr in psutil.net_if_addrs().get(name, []):
                    if addr.family == socket.AF_INET and addr.address !=
'127.0.0.1':
                        return 'підключено'
                return 'неактивний'
            except:
                return 'неактивний'

        def _is_interface_active(self, name):
            """Перевірити, чи активний інтерфейс"""
            return self._get_interface_status(name) in ['активний',
'підключено']

        def _monitor_status(self):
            """Моніторинг статусу інтерфейсу"""
            while self.running:
                time.sleep(5)
                if self.running and not
self._is_interface_active(self.interface):
                    print(f"[WARNING] Інтерфейс {self.interface} відпав!")
                    if self.auto_switch_best_interface():
                        self._restart_capture()

        def auto_switch_best_interface(self):
            """Автоматично перемкнути на кращий інтерфейс"""
            best = self._find_best_interface()
```

```
        if best and best != self.interface:
            print(f"[INFO] Автоматичне переключення → {best}")
            return self.switch_interface(best)
        return False

    def switch_interface(self, new_name):
        """Перемкнути на новий інтерфейс"""
        if self.running:
            print("[WARNING] Спочатку зупиніть моніторинг")
            return False
        if new_name in [i['name'] for i in self.available_interfaces]:
            self.interface = new_name
            print(f"[OK] Переключено на: {new_name}")
            self.reset_stats()
            return True
        print(f"[ERROR] Інтерфейс не знайдено: {new_name}")
        return False

    def _restart_capture(self):
        """Перезапустити захоплення"""
        print("[INFO] Перезапуск захоплення...")
        self.running = False

        if self.capture_thread and self.capture_thread.is_alive():
            self.capture_thread.join(timeout=3)

        time.sleep(1)
        self.reset_stats()

        # Запустити знову
        self.start_capture()

    def _capture_packets(self):
        """Основна функція захоплення пакетів"""
        try:
            sniff(
                iface=self.interface,
                prn=self._process_packet,
                store=False,
                stop_filter=lambda p: not self.running
            )
        except Exception as e:
```

```
            print(f"[ERROR] Помилка sniff: {e}")
            if self.running:
                time.sleep(2)
                self.auto_switch_best_interface()
                self._capture_packets()

    def _process_packet(self, packet):
        """Обробка кожного пакета"""
        with self.lock:
            t = time.time()
            size = len(packet)
            pkt_type = 'OTHER'
            syn_flag = False
            src_ip = None
            dst_port = None

            self.stats['packet_count'] += 1
            self.stats['byte_count'] += size
            self.stats['timestamps'].append(t)

            if IP in packet:
                ip = packet[IP]
                src_ip = ip.src
                self.stats['src_ips'][ip.src] += 1

                if TCP in packet:
                    tcp = packet[TCP]
                    pkt_type = 'TCP'
                    dst_port = tcp.dport
                    self.stats['packet_types']['TCP'] += 1
                    self.stats['dst_ports'][tcp.dport] += 1
                    if tcp.flags & 0x02:  # SYN flag
                        syn_flag = True
                        self.stats['syn_count'] += 1
                        self.stats['connections'][ip.src].add(ip.dst)

                elif UDP in packet:
                    udp = packet[UDP]
                    pkt_type = 'UDP'
                    dst_port = udp.dport
                    self.stats['packet_types']['UDP'] += 1
                    self.stats['dst_ports'][udp.dport] += 1
```

```
                    elif ICMP in packet:
                        pkt_type = 'ICMP'
                        self.stats['packet_types']['ICMP'] += 1

                self.packet_history.append((t, pkt_type, size, syn_flag,
src_ip, dst_port))

        def _cleanup_old_stats(self):
            """Очистити застарілу статистику"""
            now = time.time()

            # Очищати кожні cleanup_interval секунд
            if now - self.last_cleanup < self.cleanup_interval:
                return

            self.last_cleanup = now
            window = now - self.sample_duration * 2  # Зберігати дані за подвійне
вікно

            # Очистити старі timestamps
            while self.stats['timestamps'] and self.stats['timestamps'][0] <
window:
                self.stats['timestamps'].popleft()

            # Очистити старі пакети з історії - deque зробить це автоматично
завдяки maxlen
            # але для старих даних в stats потрібно обнулити
            if len(self.packet_history) > 0:
                # Обнулити лічильники для свіжого підрахунку
                recent_packets = [p for p in self.packet_history if p[0] >=
window]

                # Перерахувати src_ips та dst_ports тільки для останнього вікна
                new_src_ips = defaultdict(int)
                new_dst_ports = defaultdict(int)

                for p in recent_packets:
                    if len(p) > 4 and p[4]:
                        new_src_ips[p[4]] += 1
                    if len(p) > 5 and p[5]:
                        new_dst_ports[p[5]] += 1
```

```
                # Оновити словники
                self.stats['src_ips'] = new_src_ips
                self.stats['dst_ports'] = new_dst_ports

                # Обнулити лічильники
                self.stats['syn_count'] = sum(1 for p in recent_packets if p[3])
                self.stats['packet_types'] = defaultdict(int)
                for p in recent_packets:
                    if p[1] in ['TCP', 'UDP', 'ICMP']:
                        self.stats['packet_types'][p[1]] += 1
            else:
                # Якщо немає історії пакетів, обнулити лічильники
                self.stats['src_ips'].clear()
                self.stats['dst_ports'].clear()
                self.stats['syn_count'] = 0
                self.stats['packet_types'].clear()

        def _entropy(self, counts):
            """Обчислити ентропію"""
            if not counts:
                return 0.0
            total = sum(counts)
            return -sum((c / total) * np.log2(c / total) for c in counts if c >
0)

        def reset_stats(self):
            """Скинути статистику"""
            with self.lock:
                self.stats['packet_count'] = 0
                self.stats['byte_count'] = 0
                self.stats['packet_types'].clear()
                self.stats['src_ips'].clear()
                self.stats['dst_ports'].clear()
                self.stats['syn_count'] = 0
                self.stats['connections'].clear()
                self.stats['timestamps'].clear()
                self.packet_history.clear()  # Додано очищення історії пакетів
                self.last_cleanup = time.time()

        # ========== Методи для сумісності ==========
```

```
        def _warn_if_wifi(self):
            """Попередити, якщо використовується Wi-Fi"""
            if self._determine_interface_type(self.interface) == 'Wi-Fi':
                print("[INFO] Використовується Wi-Fi адаптер")
                print("   → Видно тільки трафік до вашого ПК")
                print("   → Для повного моніторингу підключіть Ethernet\n")

        def _print_interfaces_diagnostic(self):
            """Надрукувати діагностику інтерфейсів"""
            print("\n[INFO] ДОСТУПНІ ІНТЕРФЕЙСИ:")
            print(f"{'Назва':<28} {'Тип':<12} {'Статус':<12} {'IPv4'}")
            print("-" * 75)
            for iface in self.available_interfaces:
                ip = iface['ipv4'][0] if iface['ipv4'] else "-"
                print(f"{iface['name']:<28}                {iface['type']:<12}
{iface['status']:<12} {ip}")
            print()
```