Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Система для розпізнавання шкідливого програмного забезпечення на основі штучного інтелекту
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-1

Дмитро Долгов
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект

(повна назва освітньої програми)
Керівник доц. Євген Павленко
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Долгову Дмитру Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система для розпізнавання шкідливого програмного забезпечення на основі штучного інтелекту

затверджена наказом університету від 19 травня 20 25 р. № 378Ст
2. Термін подання студентом роботи до екзаменаційної комісії 20 червня 20 25 р.
3. Вихідні дані до роботи Науково-технічні публікації, дані Інтернет-джерел та відомих наукових проєктів, Python-документація, документація до бібліотек, документація до сервісів, набір даних для тренування та тестування системи

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задачі
2) Методи виявлення шкідливого ПЗ
3) Розробка та тестування системи виявлення шкідливого ПЗ

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз предметної галузі	22.05.2025	виконано
3	Огляд існуючих рішень та підходів	25.05.2025	виконано
4	Постановка задачі	26.05.2025	виконано
5	Аналіз методів виявлення шкідливого ПЗ	01.06.2025	виконано
6	Розробка та тестування системи	05.06.2025	виконано
7	Написання пояснювальної записки	09.06.2025	виконано
8	Перевірка на академічний плагіат	10.06.2025	виконано
9	Проходження нормоконтролю	12.06.2025	виконано
10	Підготовка презентації та доповіді	13.06.2025	виконано
11	Попередній захист	15.06.2025	виконано
12	Рецензування	16.06.2025	виконано
13	Захист перед ЕК	20.06.2025	

Дата видачі завдання 19 травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Євген Павленко
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 73 с., 15 рис., 1 табл., 1 дод., 35 джерел.

ГРАДІЄНТНИЙ БУСТИНГ, КАСКАДНА СИСТЕМА, МАШИННЕ НАВЧАННЯ, ПОЯСНЮВАНІСТЬ, СТАТИСТИЧНИЙ АНАЛІЗ, ШКІДЛИВЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, CNN, MALCONV.

Об’єкт дослідження – процес виявлення шкідливого програмного забезпечення у корпоративних та хмарних інформаційних системах.

Предмет дослідження – методи й засоби автоматизованої класифікації виконуваних файлів і телеметричних подій із використанням алгоритмів штучного інтелекту та пояснюваних моделей.

Мета роботи – розробити й експериментально перевірити прототип гібридної системи, що об’єднує статичний, динамічний та контекстний аналіз, забезпечує високу точність виявлення шкідливого програмного забезпечення і надає зрозуміле пояснення кожного рішення.

Методи дослідження – статистичний аналіз структурних ознак, поведінкове спостереження у віртуальному середовищі, градієнтне бустування, згорткові та трансформерні нейронні мережі, графові моделі викликів функцій, локальні методи пояснюваності, експериментальне оцінювання за метриками точності, повноти та середнього часу виявлення.

У межах роботи спроектовано й експериментально перевірено каскадну систему виявлення шкідливого програмного забезпечення, що поєднує легковажний статичний фільтр зі згортковою моделлю для глибинного байтового аналізу. Отримані результати підтверджують дієвість гібридного підходу та демонструють потенціал прототипу як навчальної і дослідницької платформи для подальшого удосконалення методів кіберзахисту.

ABSTRACT

Bachelor’s thesis contains: 73 pp., 15 fig., 1 tabl., 1 ann., 35 references.

CASCADE SYSTEM, CNN, EXPLAINABILITY, GRADIENT BOOSTING, MACHINE LEARNING, MALCONV, MALWARE, STATISTICAL ANALYSIS.

Object of the research is the process of detecting malicious software in corporate and cloud-based information systems.

Subject of the research is the set of methods and tools for automated classification of executable files and telemetry events using artificial intelligence algorithms and explainable models.

The aim of the work is to develop and experimentally validate a hybrid-system prototype that combines static, dynamic and contextual analysis, achieves high malware-detection accuracy and provides a transparent explanation for every decision.

Research methods include statistical analysis of structural features, behavioural observation in a virtual environment, gradient boosting, convolutional and transformer neural networks, graph-based call-graph models, local explainability techniques, and experimental evaluation with precision, recall and mean time to detection metrics.

In this study, a cascade malware-detection system was designed and experimentally evaluated; it combines a lightweight static filter with the deep byte-level model. The results confirm the effectiveness of the hybrid approach and demonstrate the prototype’s potential as an educational and research platform for further advancement of cybersecurity methods.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 8
Вступ 9
1 Аналіз предметної галузі та постановка задачі 10
1.1 Аналіз предметної галузі 10
1.2 Сучасні рішення та технологічні підходи 17
1.3 Постановка задачі 24
2 Методи виявлення шкідливого ПЗ 25
2.1 Класифікація існуючих методів 25
2.1.1 Сигнатурні методи 25
2.1.2 Правила поведінки 27
2.1.3 Динамічний аналіз у віртуальному середовищі 29
2.1.4 Гібридні системи 31
2.2 Методи машинного навчання 32
2.2.1 Побудова ознак для машинного навчання 32
2.2.2 Класичні моделі машинного навчання 34
2.2.3 Глибокі згорткові мережі 36
2.2.4 Трансформерні моделі для послідовностей викликів 39
2.2.5 Графові нейронні мережі 41
2.2.6 Автокодувальники 43
2.2.7 Інші класичні алгоритми 45
2.3 Порівняння методів ШІ для виявлення шкідливого ПЗ 47
2.4 Метрики оцінки ефективності 49
3 Розробка та тестування системи виявлення шкідливого ПЗ 52
3.1 Архітектура системи 52
3.2 Збір і підготовка даних 54
3.3 Тренування моделі LightGBM 56
3.4 Створення та тренування моделі MalConv-Lite 57
3.5 Побудова програмного конвеєра 60

3.6 Пояснюваність результатів 63
3.7 Тестування каскаду 65
Висновки 67
Перелік джерел посилання 69
Додаток А Відомість кваліфікаційної роботи 73

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення;

API – Application Programming Interface – інтерфейс прикладного програмування;

AUC – Area Under Curve – площа під кривою;

CNN – Convolutional Neural Network – згорткова нейронна мережа;

EDR – Endpoint Detection and Response – система кібербезпеки, яка забезпечує безперервний моніторинг і збір даних на кінцевих точках;

GNN – Graph Neural Network – графова нейронна мережа;

GPU – Graphics Processing Unit – графічний процесор;

KNN – K-Nearest Neighbors – метод k-найближчих сусідів;

ONNX – Open Neural Network Exchange – відкритий обмін нейронними мережами;

ROC – Receiver Operating Characteristic – характеристика робочих параметрів приймача;

SOC – Security Operations Center – центр операцій безпеки;

SVM – Support Vector Machine – метод опорних векторів;

XDR – Extended Detection and Response – технологія кібербезпеки, яка забезпечує розширене виявлення загроз та реагування на них.

ВСТУП

Інформаційні технології проникли у всі ключові галузі – від фінансів і медицини до енергетики та промислового виробництва. Водночас програмне середовище дедалі частіше стає мішенню цілеспрямованих та масових кібератак. Спектр загроз розширився: поряд із класичними вірусами з’явилися безфайлові сценарії, атаки на ланцюг постачання й підробки мультимедійного контенту. Звичайні захисні засоби, що спираються на фіксовані підписи або статичні правила, уже не встигають за темпом еволюції шкідливого коду й потребують підкріплення методами штучного інтелекту.

У такій ситуації актуальність дослідження визначається сукупністю чинників. По-перше, бізнесові й державні структури зазнають зростаючих фінансових збитків від інцидентів, які порушують безперервність операцій і підривають довіру користувачів. По-друге, регуляторні акти – зокрема європейські нормативи щодо кіберстійкості та прозорості алгоритмів – вимагають від розробників доводити не лише ефективність, а й пояснюваність захисних рішень. По-третє, поширення хмарних сервісів, периферійних обчислень і пристроїв інтернету речей розмиває традиційні мережеві межі. Отже, здатність виявляти шкідливі дії має бути однаково високою на різних платформах і у різних середовищах.

Метою цієї роботи є розроблення прототипу гібридної системи, здатної у реальному часі виявляти шкідливе програмне забезпечення на різних платформах – від хмарних середовищ до вузлів крайового обчислення – поєднуючи швидкий статичний фільтр зі згортковою нейромережею для складних випадків. Запроектоване рішення орієнтується на використання у навчальних і дослідницьких середовищах, а також як базовий прототип для подальшого розширення й інтеграції у прикладні корпоративні рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

Стрімке зростання кількості кіберінцидентів – від перших експериментальних вірусів 1980-х років до сучасних багаторівневих атак на критичну інфраструктуру – перетворило виявлення шкідливого програмного забезпечення на одну з центральних задач інформаційної безпеки. Сьогодні проблема ускладнюється не лише технічними аспектами, такими як безфайлові загрози чи ланцюгові компрометації оновлень, а й економічними та нормативними чинниками. Збитки обчислюються мільярдами доларів, страхові премії дорожчають, а регулятори висувають суворі вимоги до прозорості й швидкості реагування.

1.1 Аналіз предметної галузі

Початок ери шкідливих програм зазвичай відлічують від появи віруса Brain (1986) та черв’яка Morris Worm (1988). Обидва приклади були, швидше, технічними дивацтвами, ніж справжньою загрозою. Brain просто змінював сектор завантаження дискет, а Morris випадково зупинив комп’ютерні мережі університетів, розмножуючись у нескінченному циклі. Тим не менш саме вони задали головний вектор еволюції, а саме код, що поширюється, здатний порушувати нормальну роботу інформаційних систем і робить це швидше, ніж людина встигає помітити й усунути проблему. Через десятиліття цей вектор привів до Stuxnet (2010), який вивів з ладу іранські центрифуги, NotPetya (2017), що паралізував логістику портів на трьох континентах, і WannaCry, котрий зупинив прийом пацієнтів у лікарнях Великої Британії. Ці події показали, що шкідливий код перейшов із площини «цифрових жартів» у сферу, де комп’ютерний інцидент обертається реальними економічними втратами, а інколи – ризиком для життя.

Сьогодні масштаб проблеми обчислюють уже не окремими випадками, а мільярдами. У щорічному звіті SonicWall за 2023 рік зазначено: за дванадцять місяців у світі зафіксовано 6,06 мільярдів спроб зараження, що зображено на рисунку 1.1. Це дає середню швидкість понад дві сотні нових об’єктів щосекунди [1].

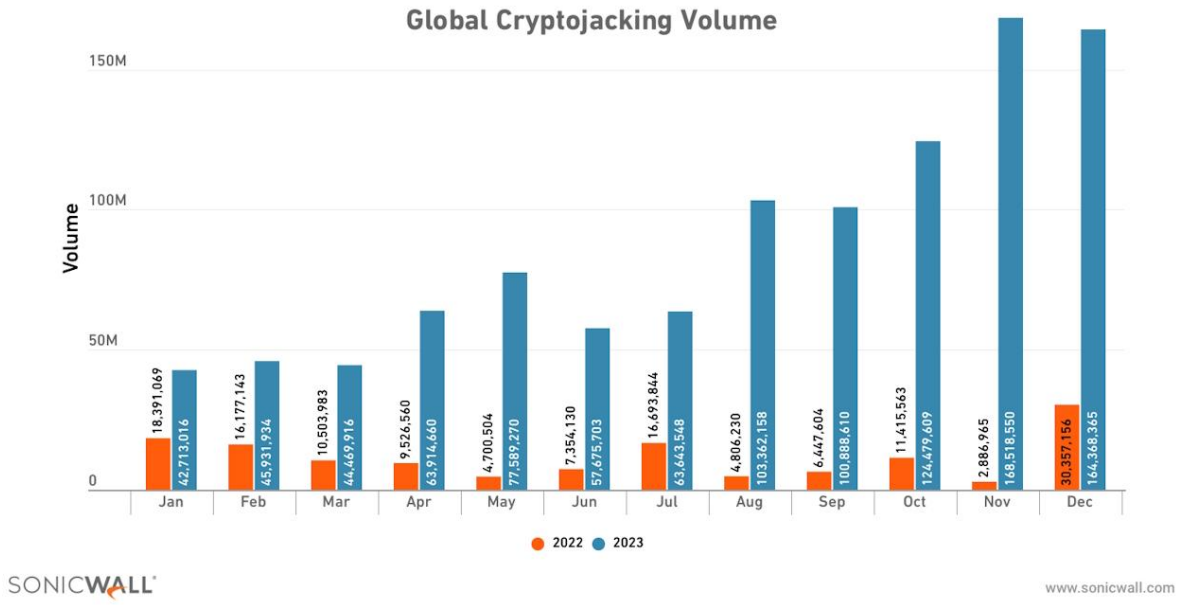

Рисунок 1.1 – Динаміка загальносвітових спроб зараження за 2022–2023 рр.

Одночасно інша метрика, «break-out time» – час між початковим проникненням і появою шкідливого процесу на сусідньому сервері – скоротився до сорока восьми хвилин. До 2015 року цей показник обчислювали годинами, а ще раніше – днями, тож прискорення у десятки разів фактично позбавляє людину можливості реагувати без автоматизації [2]. Поки спеціаліст переглядає журнал подій, програма-шифрувальник уже кодує архіви, а прихований канал зв’язку встановлює з’єднання з віддаленим сервером зловмисника. Це підтверджує графік, відображений на рисунку 1.2.

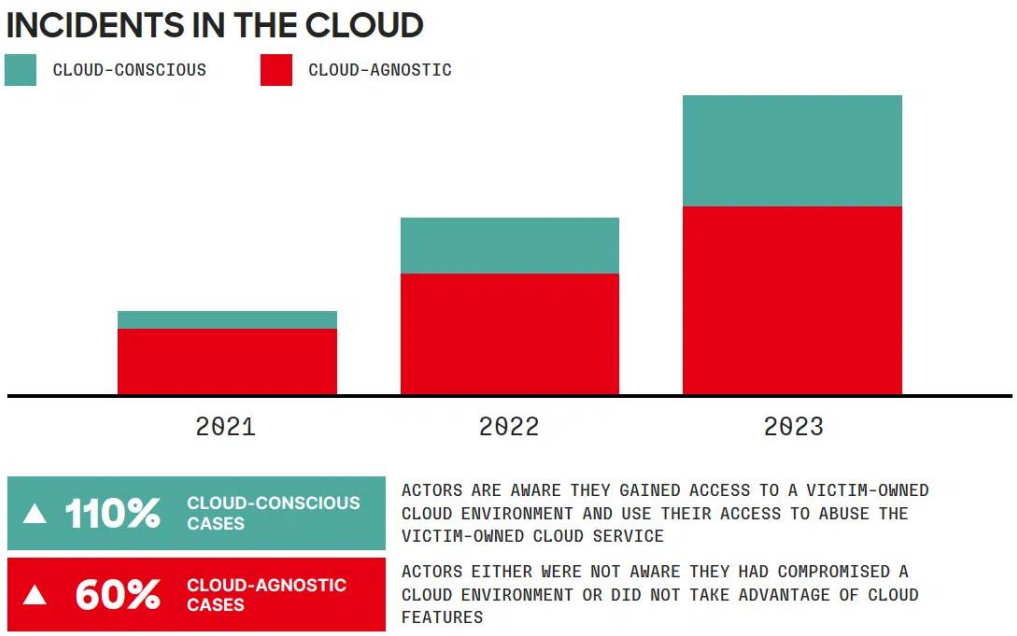

Рисунок 1.2 – Розподіл інцидентів у хмарному середовищі та їх зростання у 2021–2023 рр.

Технічний портрет загроз змінився не менше. Двадцять років тому домінували файлові віруси й поштові макровіруси. Нині домінують безфайлові скрипти, які функціонують у оперативній пам’яті, або ланцюгові атаки на механізми оновлень. Компрометація SolarWinds Orion у 2020 році стала поворотним моментом: зловмисники вписали шкідливий код у цифровий підпис офіційної бібліотеки, й оновлення, яке мало підвищити безпеку, відкрило вікно у понад 18 тисячах корпоративних мереж [3]. Інцидент змусив уряди і бізнес переглянути підходи до керування залежностями та ініціював популяризацію SBOM (Software Bill of Materials) – реєстру усіх компонентів програмного продукту.

Паралельно із технікою змінився і спектр жертв. Колись головними мішенями були банки, де зловмисникам удавалося безпосередньо монетизувати викрадені реквізити. Згодом двофакторна аутентифікація й машинний аналіз транзакцій знизили ефективність прямого шахрайства. Тоді увага зловмисників переключилася на об’єкти критичної

інфраструктури й ланцюги постачання. Лікарні вимушені обирати між сплатою викупу та ризиком для пацієнтів, енергетичні компанії – між тривалими простоями і штрафами регуляторів, а логістичні концерни – між відновленням серверів і зривом глобальних перевезень. Сумарні світові збитки, за оцінкою деяких експертів, перевищують 5 трильйонів доларів на рік і продовжують зростати [4].

Окремий вимір – нормативний. Європейська директива NIS2 передбачає для операторів критичної інфраструктури штрафи до десяти мільйонів євро за несвоєчасне повідомлення про інцидент та недостатні заходи безпеки [5]. Регламент AI Act іде ще далі. Класифікуючи системи кіберзахисту як високоризикові, він зобов'язує розробників доводити, що алгоритм є надійним, не дискримінує користувача і забезпечує пояснюваність. Відтак галузь, яка досі покладалася на «чорні скриньки» – моделі що не надають зрозумілого пояснення своїх рішень – мусить інтегрувати Explainable AI: методи SHAP, LIME чи інтегровані градієнти, щоб демонструвати, які саме ознаки обумовили «червону» тривогу [6].

Додаткові виклики пов'язані зі змінами в побудові корпоративних інформаційних систем. Поширення хмарних сервісів і великої кількості невеликих програмних модулів, що запускаються окремо, перетворило IT-інфраструктуру на своєрідну мозаїку. Кожен контейнер, кожна функція, що працює на платформі «за вимогою», і кожен виробничий датчик Інтернету речей (Internet of Things, IoT) стає окремим вузлом, де потенційно може з'явитися шкідливий код. Традиційна ідея єдиного захищеного периметра втратила актуальність. Дані розміщено у різних середовищах, а співробітники взаємодіють із застосунками безпосередньо через Інтернет. Навіть якщо компанія впроваджує принцип «не довіряти за замовчуванням», потрібно контролювати безпеку на всіх рівнях – від вихідного коду на сервері розробки до запитів, що надходять до програмних інтерфейсів.

Стандарти, що описують життєвий цикл атаки, стали детальнішими. Матриця MITRE ATT&CK, частина якої наведена на рисунку 1.3, перераховує понад дві сотні технік і підказує фахівцям, де розміщувати сенсори.

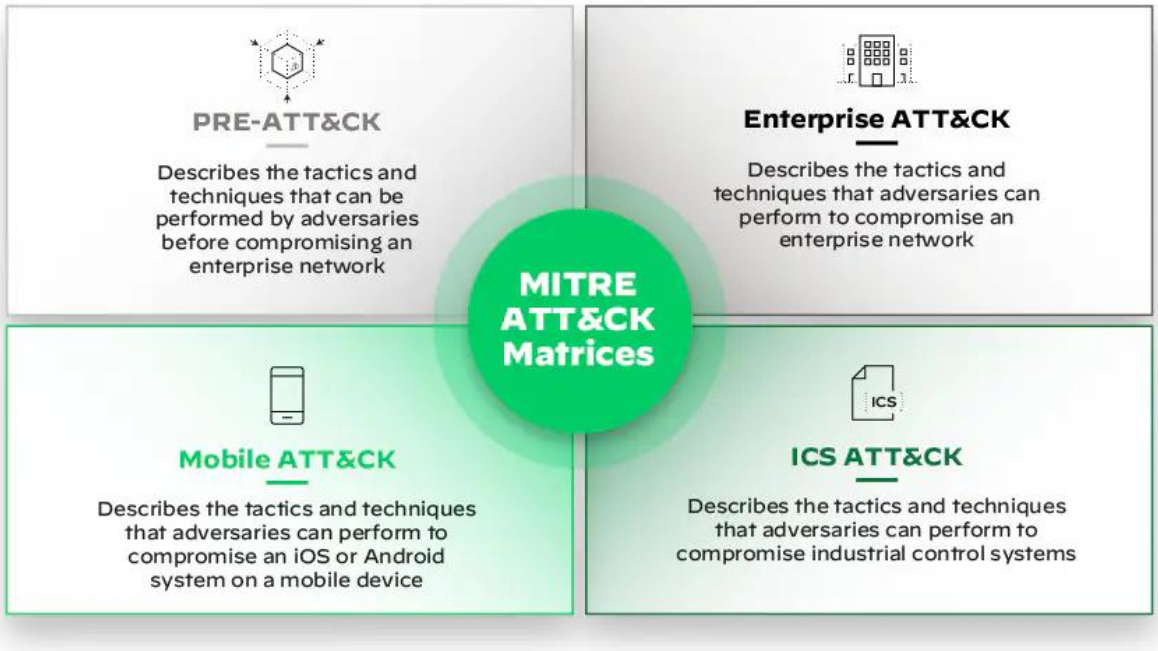

Рисунок 1.3 – Фрагмент матриці MITRE ATT&CK і класифікація тактик нападників

Та навіть найкраща матриця марна, якщо дані не обробляються в реальному часі. Провідні постачальники EDR (Endpoint Detection and Response) та XDR (Extended Detection and Response) збирають телеметрію з десятків мільйонів агентів, що генерують трильйони подій щодня. Лише автоматизовані моделі машинного навчання здатні обробити такий обсяг і виділити аномалію до того, як шкідливі процеси запустять шифрування. Комерційні звіти демонструють, що у третьому раунді тесту MITRE ATT&CK одна з хмарних платформ виявила й заблокувала 94 % тактик противника до фази пошкодження даних [7].

Попри успіхи, жоден існуючий підхід не універсальний. Сигнатури блискавично блокують спам, але пропускають нові зразки. Евристика ловить невідоме, але генерує зайві тривоги. Глибокі мережі бачать складні шаблони, проте потребують великих обчислювальних ресурсів і регулярного перенавчання, бо загрози еволюціонують щомісяця. Більш того, більшість моделей залишаються «чорними скриньками», несумісними з аудиторськими вимогами. Це ставить бізнес перед трьома взаємопов'язаними викликами – потрібно і високу точність, і швидку реакцію, і зрозуміле пояснення.

Додатковий тиск на системи захисту пояснюється тим, що розповсюдження шкідливого програмного забезпечення давно стало комерційною послугою. Нелегальні онлайн-майданчики, зокрема BreachForums і RAMP (Ransomware and Advanced Malware Protection), продають готові набори шкідливого коду з гарантією обходу антивірусів, здають в оренду серверну інфраструктуру з цілодобовою підтримкою та пропонують комплекти шифрувальників, у яких замовник отримує звичний веб-інтерфейс для керування й контрольні заявки служби підтримки [8]. Завдяки такій сервісній моделі навіть невеликі злочинні групи здатні проводити операції, що за складністю наближені до державних кібератак десятирічної давності. Аналітики прогнозують, що обсяг світового ринку кіберстрахування досягне приблизно 23 мільярдів доларів уже 2026 року, й прямо називають комерціалізацію шифрувальників головним чинником цього зростання. Що частіше відбуваються інциденти зі шифруванням даних, то вищими стають страхові внески [9]. У підсумку підприємства опиняються між прямими збитками, зростанням страхових платежів і штрафами регуляторів та дедалі частіше усвідомлюють, що впровадження проактивного захисту обходиться дешевше, ніж ліквідація наслідків атак.

Додаткову складність уносить генеративний штучний інтелект. У лютому 2024 року гонконзький офіс міжнародної корпорації втратив 25 мільйонів доларів після відеоконференції, на якій усі учасники, окрім

одного справжнього бухгалтера, були deepfake-аватарами [10]. Нейромережа синхронізувала голос і міміку, а документи підписувалися правдоподібною цифровою графікою. Таким чином, перевірка виконуваного файлу вже недостатня. Система безпеки мусить ідентифікувати підроблені відео-, аудіо- й текстові потоки, що розмиває межу між традиційним антивірусом і засобами медіа-форензіки.

Ланка постачання залишається найбільш уразливою частиною цифрової інфраструктури. Після використання уразливості в MOVEit Transfer у 2023 році у відкритому доступі опинилися дані сотень логістичних компаній, зокрема маршрути контейнерних перевезень, а нападники розсилали електронні листи, складені так, щоб їх було важко відрізнити від справжніх. Такі інциденти змушують корпорації вимагати від постачальників звіти про склад програмного забезпечення і перевіряти кожне внесення змін до коду, проте навіть повний перелік компонентів не гарантує безпеки, якщо оновлення підписане чинним сертифікатом, а шкідливий модуль активується із затримкою [11].

У довгостроковій перспективі постає ризик, що з появою квантових комп’ютерів традиційні криптографічні схеми втратять стійкість. Національний інститут стандартів і технологій США NIST вже оприлюднив перелік рекомендованих для переходу алгоритмів, які мають замінити вже існуючі RSA (Rivest-Shamir-Adleman) та ECC (Elliptic Curve Cryptography): Kyber, Dilithium, Falcon. Організаціям пропонують принцип «криптографічної гнучкості», тобто кожен канал передавання даних, сховище коду чи система дистанційного оновлення повинні мати можливість перейти на новий алгоритм без переривання роботи [12]. Для систем виявлення шкідливого програмного забезпечення (далі ПЗ) це означає подвійну вимогу. По-перше, захищати власні компоненти за допомогою стійких до квантового аналізу методів, а по-друге, відстежувати спроби атак, спрямованих на захищені канали, якими надходять файли для перевірки.

Людський фактор залишається критичною точкою. Згідно з опитуванням ISACA, 62 % організацій, що стали жертвами шифрувальників, не проводили жодного живого тренування реагування, обмежуючись паперовими регламентами [13]. Директива NIS2 та банківський регламент DORA прямо зобов’язують компанії проводити регулярні імітаційні тренування і мати документований детальний план відновлення. Без автоматизованої системи виявлення, яка надає зрозумілі пояснення, навіть тренований персонал витрачає дорогоцінні хвилини на верифікацію інциденту, тоді як час від початкового проникнення до поширення шкідливого коду нині обчислюється десятками хвилин.

Таким чином, ландшафт загроз нині формується сукупністю чинників: швидке збільшення кількості нових шкідливих програм, комерціалізація злочинної діяльності, застосування автоматизованих засобів створення переконливих підробок, атаки через ланцюги програмного постачання та очікувана поява квантових комп’ютерів. Бізнесу доводиться враховувати прямі фінансові збитки, зростання страхових платежів і вимоги регуляторів, тоді як його IT-середовище розподілене між віддаленими центрами обробки даних, локальними серверами та численними вбудованими пристроями. За таких умов підходи, що спираються лише на бази відомих підписів, прості правила або періодичні перевірки, вже не забезпечують потрібної точності, швидкості й прозорості. Необхідна нова, гібридна, пояснювана та масштабована система, здатна однаково надійно працювати і в центральних серверних приміщеннях, і на виробничих об’єктах, швидко переходити на нові криптографічні стандарти та пояснювати причини кожного свого рішення.

1.2 Сучасні рішення та технологічні підходи

Розмаїття сучасних засобів виявлення шкідливого програмного забезпечення формувалося пошарово. Кожне нове покоління не скасовувало

попереднє, а накладалося на нього, намагаючись компенсувати виявлені недоліки. Найдавніший шар становлять сигнатурні сканери, що порівнюють контрольні суми або характерні фрагменти коду із базою еталонів. Цей підхід і сьогодні лишається швидким та ефективним проти добре відомих сімейств, але безсилий перед маскуванням й поліморфізмом. Щоб підвищити гнучкість, виробники додали евристичний аналіз: модулі відстежують підозрілі дії у файловій системі, реєстрі або мережевих каналах. Проте велика кількість хибних спрацювань призвела до розчарування користувачів і пошуку способу автоматично відрізняти нетипову, але легальну активність від справжнього нападу.

На зміну традиційним антивірусам прийшли засоби захисту робочих станцій і серверів, які збирають тисячі системних подій і передають їх на віддалені сервери для обробки. Наприклад, Microsoft Defender for Endpoint щодня аналізує приблизно 65 трильйонів таких подій, постійно оновлюючи навчальні дані своїх моделей [14]. Система CrowdStrike Falcon спирається на мережу потужних серверів. За оцінкою MITRE Engenuity, вона виявляє більшість спроб нападу ще на ранньому етапі, до того як шкідлива програма встигає змінити параметри автозапуску чи почати шифрувати файли [15]. Подібний принцип – місцеві датчики разом з центральною аналітикою – використовують також продукти SentinelOne Singularity, Palo Alto Networks Cortex XDR та інші постачальники.

Паралельно розвивалися ізольовані тестові середовища, так звані «пісочниці». Програми Cuckoo, архітектура якої зображена на рисунку 1.4, або VMRay запускають підозрілий файл у відокремленій віртуальній машині й спостерігають за його мережевими запитами, змінами в системному реєстрі та роботою з оперативною пам’яттю. Отриману інформацію перетворюють на числові показники й порівнюють із поведінкою уже відомих сімейств.

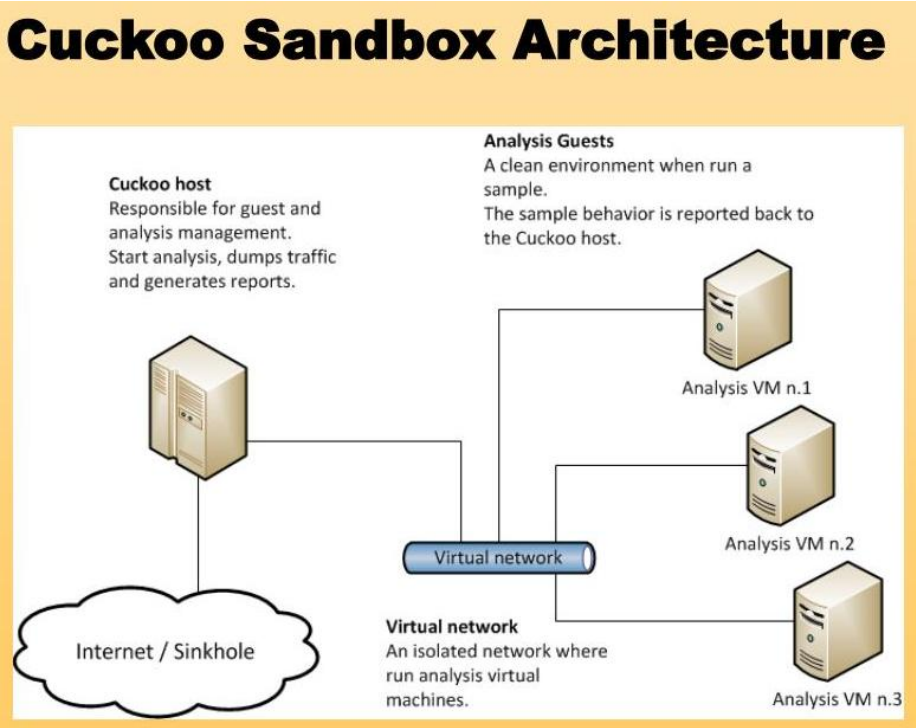

Рисунок 1.4 – Типова архітектура Cuckoo Sandbox для аналізу підозрілих файлів

У складніших схемах ці показники передаються до віддаленого центру обробки даних і доповнюють попередній статичний аналіз будови файлу. Саме так працює сервіс VirusTotal: трудомісткий динамічний аналіз виконується на стороні провайдера, а локальний клієнт надсилає лише контрольну суму або, за потреби, сам файл у захищеному каналі [16].

Найбільш відомими лишаються ClamAV і YARA. ClamAV працює здебільшого за сигнатурним принципом і підтримує миттєві оновлення бази. YARA дає змогу створювати власні правила, поєднуючи пошук послідовностей байтів із перевіркою структурних характеристик файла. Саме YARA-правила лягають в основу швидких перевірок ланцюга постачання: сховища вихідного коду запускають перевірку під час фіксації змін, щоб виявити відомі шаблони шкідливого навантаження.

Втім, найбільший прогрес зафіксовано після масового впровадження методів машинного навчання. Першою глибокою моделлю для сирих байтів

стала MalConv, схема якої зображена на рисунку 1.5. Мережа аналізувала послідовність байтів виконуваного файлу без попередньої обробки і досягла точності понад 98 % на наборі Microsoft Challenge 2015 [17].

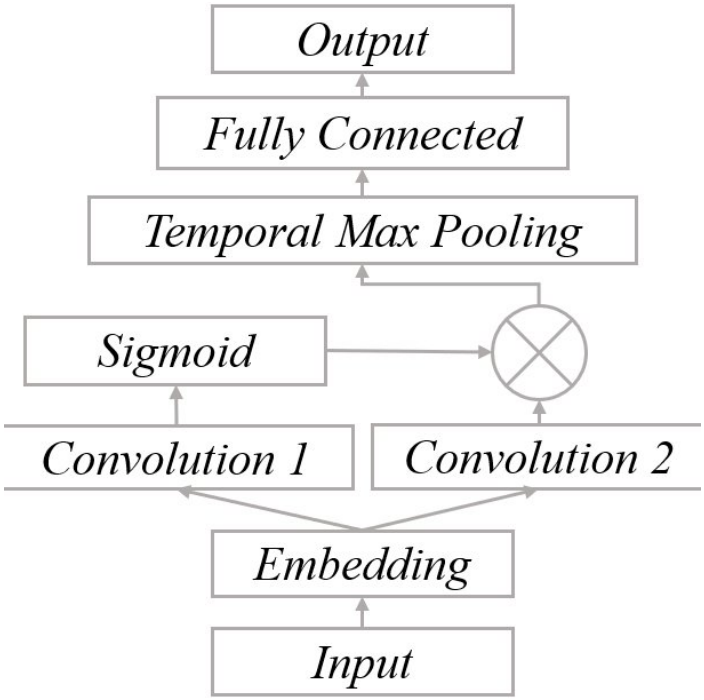

Рисунок 1.5 – Структурна схема нейронної моделі MalConv

Подальші роботи розвинули ідею у двох напрямках. Перший – згорткові й трансформерні архітектури, які бачать файл як текст чи зображення. Другий – графові нейронні мережі, що будують граф викликів функцій і знаходять аномальні зв’язки. У 2024 р. дослідники показали, що трансформер, тренований на поєднанні статичних і поведінкових ознак, перевищує F1-метрику 0,97 на вибірці понад 50 тисяч свіжих зразків [18].

Попри високу точність, складні моделі машинного навчання створюють проблему відсутності прозорості. Регламент AI Act вимагає, щоб рішення таких систем можна було обґрунтувати, а фінансові та медичні установи відмовляються приймати висновок, якщо не мають переконливих пояснень. Тому виробники вбудовують механізми пояснення: наприклад, система CrowdStrike надає фахівцеві граф дій процесів із позначенням

найбільш ризикованих вузлів, а Microsoft Defender показує, який внесок у підсумкове рішення зробила кожна ознака, спираючись на метод SHAP. Подібну можливість містять і відкриті бібліотеки, зокрема Captum для PyTorch. На практиці застосовують компромісний підхід: основна, детальна модель виконує аналіз, а окремий спрощений блок формує зрозуміле пояснення для користувача.

Змінилися й архітектурні принципи. Використання хмарних ресурсів поступово відсунуло поняття «мережевого периметра». Частину логіки перенесено безпосередньо на пристрої користувачів і виробничі контролери. Легкий агент на комп'ютері або контролері проводить попередню перевірку та передає до центру лише узагальнену статистику і найризикованіші зразки, тим самим економлячи пропускну здатність мережі. У центральному сховищі спеціальний модуль зіставляє події з різних джерел та доповнює їх даними з відкритих списків відомих загроз. Подібний дворівневий підхід застосовують не лише комерційні продукти, таку ж структуру мають відкриті платформи Elastic Security і Wazuh, які підтримують користувацькі правила на основі YARA, Suricata та Zeek.

Своє місце зайняли й сканери ланцюга постачання. Ініціативи OWASP Dependency-Track, Google OSS-Fuzz і GitHub Advanced Security пропонують автоматичне складання SBOM-файлів і порівняння їх з публічними списками вразливостей (Common Vulnerabilities and Exposures, CVE). Хоча вони не аналізують поведінку виконуваних файлів, ці інструменти закривають інший, не менш критичний сегмент ризиків – підміну легальних бібліотек або вставляння шкідливого пакета в залежності.

Поряд із високорівневими платформами з'явилися й спеціалізовані, сконцентровані на окремих етапах. Наприклад, Capsula та AppGuard перехоплюють системні виклики й конфігуруються так, щоб блокувати їх за «білим списком», а Cisco Secure Email аналізує корпус повідомлень нейромережею для відсівання підроблених вкладень ще до входу у внутрішню мережу.

Доцільно розглянути ще три напрями, які суттєво впливають на ефективність і практичну придатність систем виявлення шкідливого програмного забезпечення.

Перший – це обмін оперативними відомостями про загрози. Платформи, що підтримують стандарти STIX (Structured Threat Information Expression) і TAXII (Trusted Automated Exchange of Indicator Information), модель якого зображено на рисунку 1.6, дають змогу постачальникам і корпоративним центрам безпеки обмінюватися структурованими описами шкідливих об’єктів у майже реальному часі. Використання спільних форматів пришвидшує оновлення правил і скорочує період, протягом якого невідомий зразок залишається «невидимим» для решти учасників екосистеми.

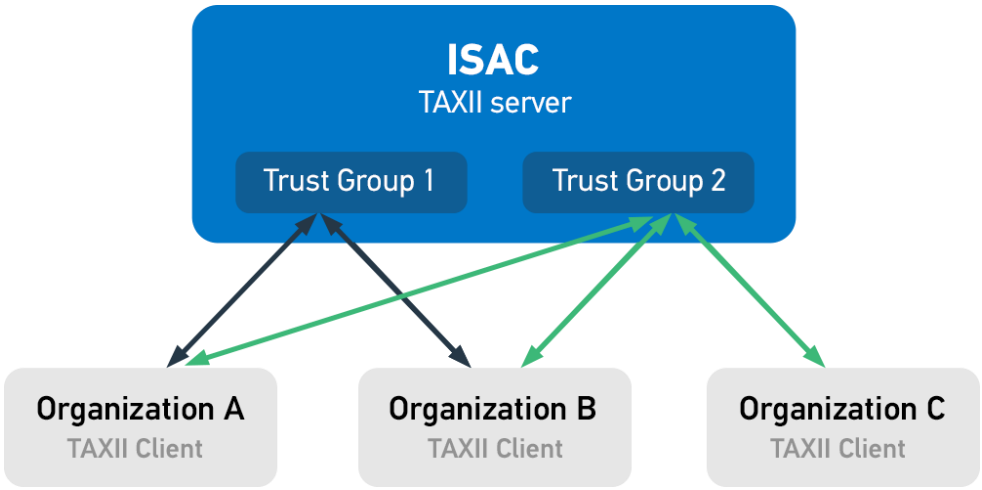

Рисунок 1.6 – Модель обміну даними про загрози за стандартами STIX і TAXII

Пілотне впровадження в енергетичному секторі ЄС показало, що середній час поширення критичних індикаторів компрометації скоротився з кількох діб до кількох годин [19].

По-друге, використання апаратного контролю цілісності. Деякі виробники вмонтовують у свої рішення захищені мікросхеми, які під час

запуску операційної системи перевіряють, чи не змінено ядро та інші важливі бібліотеки. Якщо критичний компонент завантажується у спеціально захищеній ділянці пам’яті, то навіть успішне проникнення шкідливого коду не дає змоги підмінити системні файли чи вимкнути захисне програмне забезпечення. Випробування у середовищах з високими вимогами до безперервної роботи показують, що такий апаратний контроль значно знижує ризик непомітного вимкнення захисту і при цьому майже не впливає на швидкодію [20].

По-третє, розподілене навчання моделей без централізованого збирання даних. У цьому підході програма, установлена на кожному комп’ютері, навчається на власних зразках, а потім надсилає до центрального сервера лише узагальнені параметри, без самих файлів. Це особливо важливо для галузей із суворими вимогами до конфіденційності, наприклад у медицині. Пілотні проекти в лікарнях показали, що точність розподіленої моделі майже не відрізняється від моделі, навченої на зібраних разом даних, зате помітно зменшує юридичні ризики, пов’язані з пересиланням персональної інформації [21].

Незважаючи на велику кількість підходів, жоден із них не працює достатньо ефективно окремо. Бази підписів найшвидше зупиняють давно відомі загрози, але не бачать нові. Поведінкові правила здатні помічати незнайомі зразки, проте дають багато помилкових сповіщень. Нейронні моделі охоплюють найбільше випадків, та для їхньої роботи потрібні значні обчислювальні ресурси і зрозумілі обґрунтування рішень. Тому сучасні системи поєднують усі рівні в єдиний ланцюжок: спершу легкий модуль на місцевому пристрої відсіює очевидно безпечні файли, далі хмарний сервіс корелює події з різних джерел, додає відомості з переліку програмних компонентів і, нарешті, формує пояснення результату. Така багатошарова схема знижує кількість помилкових тривог, дає змогу швидко оновлювати правила і при цьому відповідає вимогам як регуляторів, так і внутрішніх аудиторів.

1.3 Постановка задачі

Виявлення шкідливого ПЗ є складним завданням, оскільки сучасні загрози швидко змінюють код і поведінку, маскуються у ланцюгах постачання та діють у мережах без чітких периметрів. Попередній огляд показав, що жоден окремий метод не здатен самостійно забезпечити прийнятну точність, високу швидкість реагування й водночас зрозумілі пояснення. Отже, метою даної роботи є створення прототипу інформаційної системи, яка поєднує кілька шарів аналізу, працює з реальними даними в хмарних і локальних середовищах та обґрунтовує кожне своє рішення.

Для досягнення поставленої мети доцільно виконати такі послідовні кроки:

– дослідити актуальність обраної теми;

– провести критичний огляд наукових і промислових підходів до виявлення шкідливого ПЗ, визначивши їхні переваги та обмеження;

– зібрати та сформувати репрезентативну вибірку виконуваних файлів і пов’язаних телеметричних даних, забезпечивши баланс між шкідливими та безпечними зразками;

– розробити конвеєр обробки даних, який поєднує статичні ознаки з динамічними показниками;

– навчити й порівняти кілька моделей та обрати оптимальну за точністю й продуктивністю;

– інтегрувати модуль пояснення рішень, щоб забезпечити прозорість і відповідність вимогам;

– спроектувати та реалізувати архітектуру прототипу з веб-інтерфейсом для зручної інтеграції у корпоративне середовище;

– провести експериментальну оцінку за метриками точності, повноти, F1-міри та кількості помилкових спрацьовувань, порівнявши результати;

– сформулювати рекомендації щодо впровадження і можливих напрямів подальшого розвитку системи у виробничих умовах.

2 МЕТОДИ ВИЯВЛЕННЯ ШКІДЛИВОГО ПЗ

Сучасні підходи до виявлення шкідливого програмного забезпечення охоплюють увесь спектр від простих сигнатурних порівнянь до багаторівневих нейронних мереж, здатних самостійно розпізнавати приховані закономірності у великих масивах даних. Саме розвиток методів штучного інтелекту дав змогу суттєво підвищити швидкість реагування та знизити частку помилкових спрацювань у корпоративних системах захисту. Щоб обґрунтувати вибір технологій для подальшого прототипу, доцільно спершу систематизувати наявні рішення, а потім порівняти їх за точністю, швидкодією і вимогами до ресурсів.

2.1 Класифікація існуючих методів

Методи виявлення шкідливого програмного забезпечення зазвичай поділяють на чотири великі класи: сигнатурні, евристичні, поведінкові та моделі машинного навчання. Сигнатурний підхід виник першим і досі лишається першою лінією захисту, евристика додає правила, сформульовані експертами, поведінкові системи аналізують дії програми у контрольованому середовищі.

2.1.1 Сигнатурні методи

Перші антивіруси працювали за принципом «пошуку відбитка» – якщо у файлі трапляється байт-у-байт те саме, що раніше позначили як шкідливе, файл одразу блокують. Ідея з’явилася ще в епоху дискових вірусів, і залишається актуальною, бо перевіряється за частку секунди й дає безпомилковий результат щодо вже відомих загроз.

База сигнатур містить різні типи відбитків. Найпростіший – точний цифровий хеш – будь-яка зміна навіть одного байта робить хеш іншим, тож

метод безсилий проти найменших модифікацій. Щоб гнучко реагувати, зберігають короткі уривки коду – їх вистачає пережити зміну ресурсних секцій, але перекомпіляція з іншим порядком інструкцій усе одно руйнує збіг. Далі з'явилися правила YARA, де окрім самого підрядка описують і структуру файла. Таке правило охоплює цілу родину схожих вірусів. Нарешті, сучасні продукти використовують нечіткі сигнатури, що не вимагають ідеальної тотожності, а шукають достатню схожість, використовуючи формулу:

$$S = 1 - \frac{D}{L}, \tag{2.1}$$

де S – показник схожості (від 0 до 1);

D – кількість байтів, якими обидва файли різняться;

L – розмір більшого з двох порівнюваних файлів.

Якщо S перевищує поріг 0,7, тобто 70 %, то система вважає файл зміненим клоном уже відомої загрози.

Сильна сторона сигнатурного методу – продуктивність. Навіть на звичайному офісному комп'ютері перевірка документа на кілька мегабайт триває мить, тому виробники ставлять цей шар найпершим. Він одразу ідентифікує біля 70 % усіх файлів як цілком безпечні або точно шкідливі, передаючи далі лише сумнівний залишок [22].

Та цей метод завжди запізнюється на крок. Спершу зловмисник випускає новий варіант, лише потім аналітик вилучає відбиток, а вже далі оновлення надходить користувачеві. Щоб скоротити це вікно, антивіруси щогодини надсилають серверу коротку контрольну суму своєї бази й отримують лише різницю – кілька сотень кілобайт, що не перевантажує зв'язок. Другий виклик – поліморфізм. Сучасні шифрувальники пакуються оболонкою, яка щоразу шифрує код новим ключем, і точний хеш стає марним. Нечіткі сигнатури або гнучкі маски рятують частково, та й вони

виходять з ладу після радикальної перекомпіляції. Тому постачальники автоматично об’єднують сотні схожих зразків у родину й описують її одним узагальненим шаблоном, щоб база не розросталась безмежно.

Навіть такий простий підхід потребує захисту. Щоб ніхто не підкинув фальшиву базу, оновлення підписується електронним ключем виробника, клієнт перевіряє підпис і контрольну суму пакета. Після гучного інциденту SolarWinds ця перевірка стала обов’язковою [23].

Отже, сигнатурні методи залишаються актуальними, бо миттєво фільтрують очевидні випадки й бережуть ресурси. Вони, однак, реагують лише на вже відомі загрози, тому поверх цього швидкого першого шару потрібно додавати поведінкові правила, динамічний аналіз і машинне навчання.

2.1.2 Правила поведінки

Принцип поведінкового виявлення спирається на те, що будь-який шкідливий код після запуску мусить виконати низку дій, без яких атака не має сенсу. Підмінити власний хеш легко, але приховати створення копії у системній теці чи встановлення з’єднання з керувальним сервером майже неможливо, операційна система все одно фіксує ці кроки у журналах. Антивірусний агент передає такі події у потік телеметрії. Зазвичай це десятки тисяч записів на одну робочу станцію за добу й мільйони у масштабах підприємства. Усередині потоку програма виділяє лише ті події, які мають значення для безпеки: запуск алгоритму з прихованим розширенням, зміна ключів автозапуску, підозріле звернення до криптографічних функцій, створення великої кількості файлів із високою ентропією та спроби зв’язку з адресами з «чорного списку».

Щоб оцінити ризик, система перетворює усі зафіксовані дії процесу на вектор логічних ознак. Кожна ознака дорівнює «1», якщо подія відбулася, і «0», якщо ні.

Далі обчислюється простий підсумок з вагами:

$$R = \sum_{i=1}^{k} w_i z_i \,, \tag{2.2}$$

де R – ризиковий бал;

w_i – важливість події;

z_i – факт події.

Ваги задає або виробник, або фахівці SOC-центру (Security Operations Center), спираючись на свій досвід та середовище організації. Для звичайної робочої станції створення служби у системному каталозі може мати вагу тридцять, а нетиповий вихідний трафік – сорок. У серверній зоні, де багато служб запускаються за регламентом, ті самі дії отримують меншу вагу. Якщо вирахуваний бал перевищує порогове значення, процес позначають як небезпечний, у протилежному разі – ігнорують.

Класичні рушії використовували жорсткі внутрішні правила, однак із часом з’явилася необхідність швидко ділитися знанням про нові ланцюжки атак. Відкритий проєкт Sigma вирішив проблему уніфікованим форматом: аналітик описує послідовність дій у звичайному файлі YAML, а спеціальний конвертер перетворює її у запит саме тієї системи журналів, що працює у компанії. З-поміж сотень готових правил багато присвячені конкретним сімействам, наприклад Emotet чи Trickbot, і вже довели свою ефективність під час реальних спалахів [24].

Ключове завдання поведінкового підходу – відсіяти зайвий шум. Легітимні утиліти для резервного копіювання, системні адміністратори й навіть офіційні інсталятори програм часто виконують дії, схожі на шкідливі. Щоб уникнути перевантаження панелі SOC, застосовують контекстну градацію. Якщо процес підписано довіреним сертифікатом і завантажено з адреси, що давно перебуває у «білому списку», початковий бал зменшують. Навпаки, для документа, отриманого з невідомої поштової скриньки, кожна

підозріла дія додає більше очок. Додатково враховують часові рамки. Якщо три підозрілі події сталися протягом хвилини, це майже напевно атака, тоді як ті самі дії, розтягнуті на добу, часто бувають роботою адміністратора.

Система має і природні межі. Нападник може відкласти виконання шкідливих дій, перевіряючи, чи не запущено процес у віртуальній машині. Він може розбитись на два окремі легітимні процеси, що по черзі виконують частини сценарію, або користуватися вже дозволеними адміністративними інструментами. Для таких випадків поведінковий аналіз слугує проміжною ланкою. Швидкий, але не завжди достатній, він передає найнеоднозначніші файли до віртуального середовища або до статистичних моделей машинного навчання.

2.1.3 Динамічний аналіз у віртуальному середовищі

Якщо сигнатурний метод і поведінкове правило спираються на статичний код чи окремі дії, динамічний аналіз йде ще далі – він дозволяє подивитися, що саме відбувається під час реального запуску програми. Шкідливий файл завантажують у ізольоване віртуальне середовище, де йому доступні всі служби операційної системи, проте вихід назовні перекрито. Поки процес працює, спеціальний монітор фіксує кожен системний виклик, кожне мережеве звернення, кожну спробу доступу до реєстру. Результатом стає хронологічний ланцюжок подій:

$$S = \langle(\text{дія}_1, t_1), (\text{дія}_2, t_2), \dots, (\text{дія}_n, t_n)\rangle. \quad (2.3)$$

Далі ланцюжок порівнюють з еталонними сценаріями вже відомих сімейств. Насамперед окремі дії нормалізують – однакові системні виклики, зроблені різними процесами, зводять до спільного позначення, а дії-шум відфільтровують. Далі ланцюжки синхронізують за опорними подіями. Після синхронізації лишається зіставити множини подій за фіксоване вікно

часу. Щоб врахувати невеликі зсуви в часі, використовують показник схожості:

$$D = \frac{\text{кількість різних подій}}{\text{загальна кількість подій}}. \quad (2.4)$$

Чисельник показує, скільки кроків у тестовому та еталонному сценаріях відрізняється, а знаменник – скільки кроків загалом. Якщо D нижчий за поріг, файл вважають шкідливим.

Головна перевага підходу – він показує реальну поведінку. Навіть якщо вірус зашифрував своє тіло й підмінив сигнатури, дістався віртуального середовища він мусить розпакуватися і виконати свій сценарій. Оператор безпеки отримує докладний звіт: які файли створено, які адреси відвідано, який ключ реєстру змінено. Це полегшує розслідування та створення нових правил.

Є й труднощі. По-перше, динамічна перевірка займає час – щоб виявити шифрувальник, іноді потрібно дочекатися, доки той закінчить майже весь свій цикл. По-друге, існує техніка ухилення. Шкідлива програма часто перевіряє, чи працює вона у віртуальній машині, і відкладає атаки, поки не переконається, що середовище справжнє. По-третє, ресурси. В офісі з тисячами комп’ютерів запуск кожного підозрілого файлу у повноцінній віртуальній машині потребує значних обчислювальних потужностей.

Щоб мінімізувати затримки, віртуальні середовища адаптуються до загроз. Вони штучно прискорюють системний таймер, спонукаючи вірус швидше переходити до активної фази. Крім того, з’явилися лабораторії, де замість віртуальної машини використовується звичайний комп’ютер з апаратним механізмом швидкого відновлення системного диска. Програма отримує натуральне середовище і вже не може відрізнити його від справжнього робочого місця.

На практиці динамічний аналіз рідко працює окремо. Він підхоплює саме ті зразки, які минули сигнатурний метод і поведінкове правило, але досі залишаються підозрілими. Таким чином утворюється каскад. Спершу швидкий фільтр, потім логічний бал, а надто заплутані випадки доходять до віртуального середовища, де й розкривають свій справжній намір.

2.1.4 Гібридні системи

Згодом стало очевидним, що жоден окремий метод не дає ані достатньої швидкості, ані повної надійності – сигнатурний метод блискавичний, але уразливий до нових зразків, поведінкове правило бачить нові версії, та часом помиляється, динамічна віртуальна машина надто повільна, щоб перевіряти кожен файл. Виробники об’єднали всі три рівні в одну послідовність, під назвою гібридна або багатошарова система.

Каскад працює так: спершу спрацьовує найшвидший сигнатурний шар і одразу відсіює усе, що вже точно відоме – і шкідливе, і безпечне. Залишок переходить до поведінкового рушія, де події отримують ваги, а процес – проміжний бал ризику. Лише зразки, чий бал наблизився до порогового, відправляють у віртуальне середовище. Саме там програма проходить повний цикл запуску, і тепер захисник бачить, чи справді вона шифрує файли, вивантажує додаткові модулі або змінює системні налаштування.

З математичного погляду остаточний ризиковий показник складають із трьох частин:

$$R_{\text{заг}} = \alpha R_{sig} + \beta R_{beh} + \gamma R_{dyn}, \tag{2.5}$$

де R_{sig} – дорівнює 0 або 1 залежно від збігу підпису;

R_{beh} – бал, що нарахував поведінковий рушій;

R_{dyn} – результат динамічного запуску.

Коефіцієнти α, β, γ задає служба безпеки. У банківському середовищі динаміці (γ) надають найбільшу вагу, бо втрати від пропущеної загрози високі. У домашньому продукті роблять навпаки, щоб не навантажувати комп'ютер довгими перевірками.

Щоби система залишалася керованою, усі результати з робочих станцій надходять до центрального сховища. Там дані корелюють між собою. Підозрілий файл, який з'явився одразу на двадцяти комп'ютерах, отримує додатковий коефіцієнт небезпеки. Якщо ж файл зустрівся лише раз і більше ніде не з'являється, ризик можна знизити. Завдяки цьому підхід не тільки об'єднує методи, а й ураховує масштаб поширення.

Побудова такого конвеєра дала відчутний ефект. За публікаціями випробувальних лабораторій, гібридні продукти блокують понад 90 % атак ще до того, як шкідливий процес устигне змінити ключі автозапуску, а кількість хибних тривог у корпоративних мережах скоротилася майже вдвічі. Саме через ці переваги багатошарова архітектура на даний момент сприймається як галузевий стандарт [25].

2.2 Методи машинного навчання

Після огляду класичних сигнатурних та евристичних підходів, що покладаються здебільшого на наперед формалізовані правила, логічним продовженням є методи машинного навчання. Їхня відмінність у тому, що критерії віднесення файла до безпечних чи шкідливих виводяться не вручну, а автоматично. Завдяки цьому алгоритм здатен уловлювати складні залежності, які людина або прості правила часто пропускають.

2.2.1 Побудова ознак для машинного навчання

Модель машинного навчання не працює без попереднього опису того, що саме вона має аналізувати. У випадку шкідливого ПЗ звичайного сирого

двійкового файла недостатньо. Його слід перетворити на числовий набір ознак, який алгоритм здатен обробити. На практиці ознаки поділяють на три великі групи: статичні, поведінкові та контекстні.

Статичні ознаки добувають ще до запуску програми. Найпоширеніший приклад – перелік секцій у виконуваному файлі та їхній розмір. Більшість легальних програм мають схожу структуру, тому раптово велика секція «.text» або нестандартна «.bss» викликає підозру. Часто використовують і прості числові показники: ентропію вмісту, довжину імпортованих рядків, кількість вбудованих ресурсів. Щоб не втратити деталі, двійковий код поділяють на короткі фрагменти і конвертують їх у вектори за принципом, який враховує скільки разів зустрілась така послідовність байтів. Результатом стає матриця, де стовпець – це фрагмент, а рядок – конкретний файл. Умовно такий запис можна звести до формули:

$$X_{ij} = \text{кількість входжень фрагментів } j \text{ у файлі } i, \qquad (2.6)$$

де X_{ij} – елемент матриці ознак.

Матриця може містити тисячі стовпців, але машинні методи добре справляються з такою розрідженістю.

Поведінкові ознаки формують під час короткого запуску програми в контрольованому середовищі. Результатом є ланцюжок ключових дій – наприклад, звернення до мережі, виклик криптографічних бібліотек, запис файлів у системний каталог. Щоб подати ланцюжок у числовій формі, його кодують за допомогою мішку подій – перелічують, скільки разів кожен тип дії з’явився. Після такої обробки можна, наприклад, обчислити середню затримку між записом у реєстр і першим мережевим з’єднанням, що часом є надійним маркером шифрувальника.

Контекстні ознаки доповнюють картину відомостями поза самим файлом. Важливими виявляються шлях до програми, джерело

завантаження, підпис видавця, репутація IP-адреси (Internet Protocol), з якої файл прийшов, а також час доби, коли процес стартував.

Сила багатошарової таблиці ознак у тому, що вона дозволяє моделі враховувати картину цілком. Шифрувальник може приховати свій код, але не зможе одночасно імітувати нормальний шлях встановлення, і відмовитися від характерного набору мережевих викликів, і зберегти стандартний розмір секцій файлу. Отже, навіть якщо одна група ознак виявиться недостатньо інформативною, інші заповнять прогалину. Таке поєднання – перший крок на шляху до точного та стійкого класифікатора, з яким надалі працюватимуть більш складні алгоритми.

2.2.2 Класичні моделі машинного навчання

Після того як файл описано набором чисел, постає питання, як саме за цими числами зробити рішення, чи є файл шкідливим або безпечним. Один із найперших підходів – дерево рішень (Decision Tree). На рисунку 2.1 наведено схему дерева рішень.

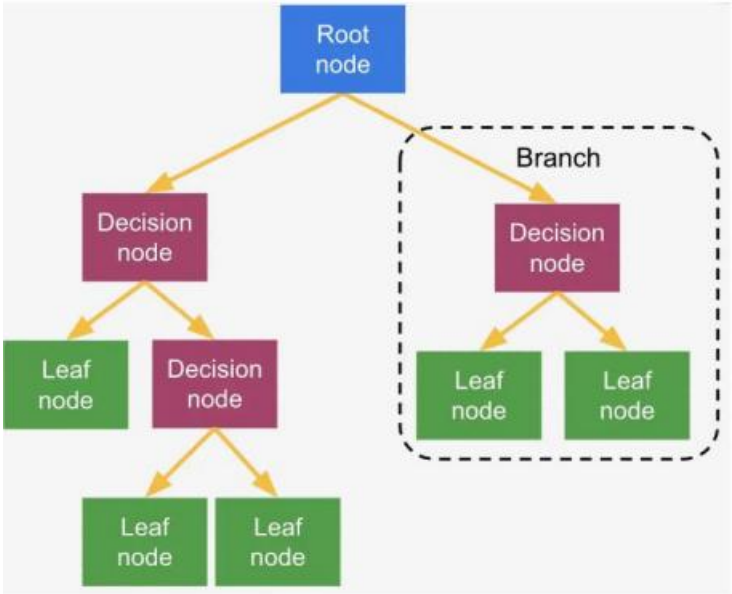

Рисунок 2.1 – Дерево рішень

Алгоритм крок за кроком ділить простір ознак: спочатку запитує, чи перевищує ентропія коду певну межу, далі перевіряє розмір секції «.text», потім дивиться, чи є звернення до мережі. У результаті формується гілка, на кінці якої написано «шкідливо» або «безпечно». Таке дерево легко читати, оскільки кожен крок пояснює, чому програма потрапила саме до цієї категорії.

Окреме дерево, однак, рідко буває точним. Коли даних багато й вони різноманітні, краще працює метод випадковий ліс (Random Forest). Ідея полягає в тому, що кожне дерево голосує, а остаточне рішення приймається за принципом більшості. Якщо m – кількість дерев, а $g_j(x)$ – рішення j-го дерева для файлу x, підсумковий результат можна записати формулою:

$$G(x) = \frac{1}{m}\sum_{j=1}^{m} g_j(x)\,, \tag{2.7}$$

де $G(x)$ – частка дерев, що проголосували «шкідливо».

Якщо ця частка перевищує поріг 0,5, файл блокують.

Наступний крок розвитку – градієнтний бустинг (Gradient Boosting). Замість того щоб будувати всі дерева одразу і незалежно, модель додає їх по одному, кожного разу намагаючись виправити помилки попередників. У спрощеному вигляді підсумок записують як:

$$F_K(x) = \sum_{k=1}^{K} \eta_k f_k(x)\,, \tag{2.8}$$

де K – кількість кроків;

η_k – вага цього дерева;

f_k – невелике «слабке» дерево.

Кожен новий член ряду робить модель точнішою, але якщо таких кроків буде забагато, виникає ризик перенавчання на шумі даних. Тому ваги η_k поступово зменшують. Перевага бустингу в тому, що він добре пристосовується до складних меж між класами, а головний недолік – довший час навчання. Моделі треба багато разів переглядати дані, щоб послідовно зменшити помилки.

Логістична регресія, хоч і здається найпростішою, досі корисна як базовий орієнтир. Вона намагається прорівняти межу між «шкідливо» та «безпечно» однією гіперплощиною й оцінює імовірність небезпеки виразом:

$$P(y = 1|x) = \frac{1}{1 + \exp\left(-(w^{\mathrm{T}}x + b)\right)}, \tag{2.9}$$

де x – вектор ознак файла;

w та b – параметри, що підбираються на навчальних даних.

Якщо результат перевищує 0,5, система сигналізує загрозу. Хоч такий поділ грубий і не виявляє складні шаблони, логістична регресія швидко тренується й показує, наскільки взагалі дані придатні до передбачення.

На практиці класичні методи – логістична регресія, дерева, бустинг – часто виступають першою хвилею машинного навчання. Вони задають базову планку якості, яку потім намагаються перевершити складніші згорткові або трансформерні мережі. Крім того, ті ж дерева рішень дають зрозумілу структуру, як саме модель дійшла висновку. Це полегшує пояснення рішень, що дедалі важливіше за сучасних нормативних вимог.

2.2.3 Глибокі згорткові мережі

Класичні моделі потребують заздалегідь підготовленого списку ознак, але глибокі нейронні мережі, зокрема згорткові (Convolutional Neural

Networks, CNN), здатні навчатися безпосередньо на сирому потоці байтів. Ідея проста: розглядати виконуваний файл так само, як зображення чи текст – послідовність чисел, у якій шукають характерні локальні шаблони. Першою помітною реалізацією такого підходу стала архітектура MalConv. На рисунку 2.2 зображена архітектура згорткової мережі.

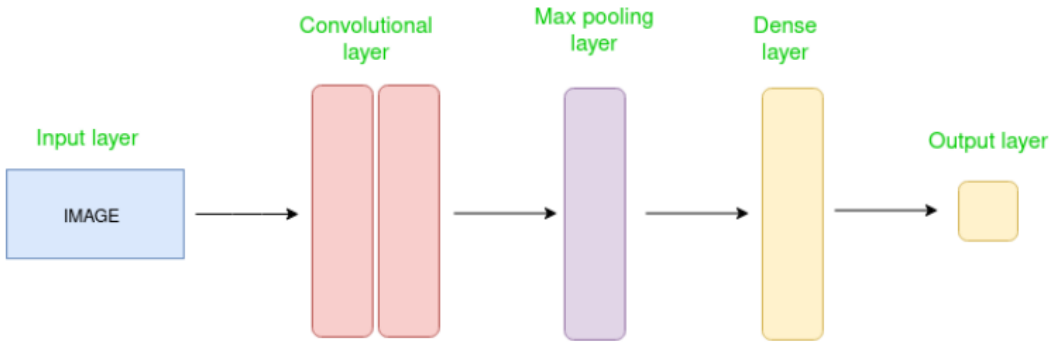

Рисунок 2.2 – Архітектура CNN

У MalConv байтовий масив приводять до однакової довжини: якщо файл коротший, його доповнюють нульовими байтами, якщо довший – обрізають хвіст. Далі на цей упорядкований ряд застосовують згортку. Формально вихід першого шару записують так:

$$y_i = \sum_{j=1}^{k} w_j x_{i+j-1}\,, \tag{2.10}$$

де y_i – результат для позиції i;

w_j – ваги фільтра;

x_{i+j-1} – значення j-го байта у вікні довжиною k.

Інакше кажучи, мережа ковзає маленьким фільтром по всьому файлу і вчиться, які послідовності байтів найчастіше трапляються у шкідливих

зразках. Кожен фільтр шукає свій набір патернів, а вихід кількох таких шарів комбінують, щоб отримати логічну карту всього файла.

Після кількох згорткових і підсумкових шарів мережа видає ймовірність загрози. Якщо значення перевищує 0,5, файл відмічають як шкідливий. Практика показала, що навіть без попередньої ручної підготовки ознак MalConv досягає точності понад 98 % на популярних відкритих наборах EMBER та BIG-2015 і випереджає більшість класичних моделей [26].

Перевага підходу очевидна – не потрібно витягати жодні специфічні характеристики, мережа сама знаходить байтові шаблони, типові для шкідливого та безпечного коду. Проте ціна – обчислювальна складність. Навіть скорочений двійковий файл у декілька мегабайт перетворюється на великий вхідний тензор, а навчання потребує графічного процесора (Graphics Processing Unit, GPU) й тривалого часу. Крім того, нерідко виникає питання пояснюваності. Хоча у згорткових мереж є техніки візуалізації важливих ділянок, вони не такі прозорі, як набір зрозумілих правил чи дерево рішень.

Окремим напрямом стало спрощення архітектури: замість одномірної згортки по всьому файлу використовують фіксоване вікно на перших мегабайтах або попереднє групування байтів у більші символи, що зменшує розмір вхідних даних. Інший шлях – поділяти файл на логічні секції й подавати їх окремими каналами. Такі полегшені моделі поступаються точністю оригінальному MalConv, але швидші на звичайному комп’ютері.

З погляду практичної інтеграції згорткові мережі часто ставлять третім етапом. Сигнатурний і поведінковий фільтр відсіюють очевидні випадки, а CNN аналізує лише обмежену кількість сумнівних зразків. У такому поєднанні вдається скористатися високою чутливістю нейромережі й водночас уникнути надмірного навантаження на апаратні ресурси.

2.2.4 Трансформерні моделі для послідовностей викликів

Якщо згорткова мережа розглядає виконуваний файл як довгий ряд байтів, трансформер дивиться на програму як на історію дій, яку вона виконує під час роботи. Ця історія прописана у вигляді послідовності системних викликів: спершу процес відкриває бібліотеку, потім читає файл, потім звертається до мережі і так далі. Кожен виклик кодують числом, а всю послідовність подають моделі, що вміє помічати віддалені зв'язки між елементами й розуміти контекст.

У трансформері ключову роль відіграє механізм «уваги» (attention). Він оцінює, наскільки важливий кожен крок для інших кроків. Формально вагу між кроками i та j записують так:

$$a_{ij} = \frac{\exp\left((q_i \cdot k_j)/\sqrt{d}\right)}{\sum_m \exp\left((q_i \cdot k_m)/\sqrt{d}\right)}, \tag{2.11}$$

де a_{ij} – відповідь, що показує, наскільки крок j важливий для кроку i;

q_i і k_j – числові представлення кроків (запит і ключ);

d – розмір цих представлень.

Коли модель обчислила всі a_{ij}, вона формує новий вектор, зважуючи подальшу увагу на найважливіші дії. Після обчислення коефіцієнтів a_{ij} усі пари «запит/ключ» об'єднують у мультиголовий блок, де кілька незалежних підпросторів уваги працюють паралельно. Результати цих голів потім конкатенують і пропускають через лінійне перетворення, утворюючи вихід шару самоуваги. Саме така комбінація кількох голів, резидіальних з'єднань і позиційного кодування й становить базову цеглину трансформера, зображену на рисунку 2.3.

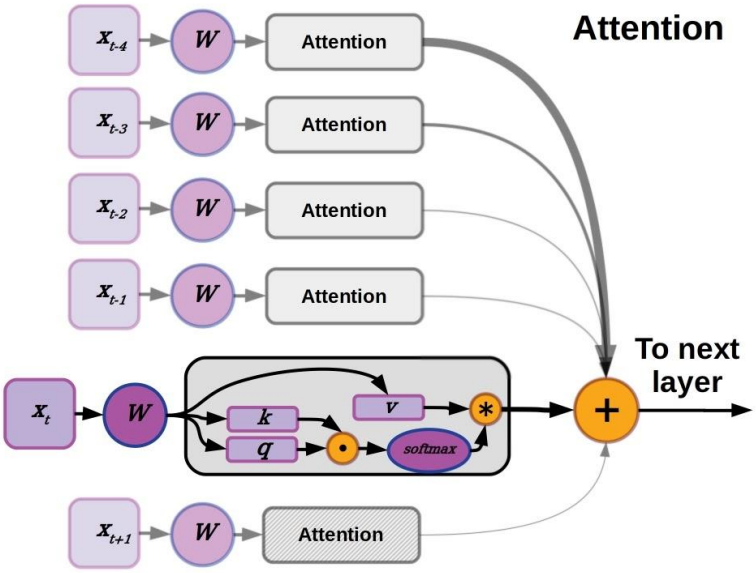

Рисунок 2.3 – Структура моделі Transformer

З практичного погляду трансформер корисний тим, що помічає далекі залежності. Наприклад, якщо програма спершу створила службу, а через кілька сотень викликів різко зашифрувала документи, модель зв’яже ці події й підвищить підозру. Класичні рекурентні (Recurrent Neural Network, RNN) або згорткові моделі такої пам’яті не мають, бо обмежені фіксованим вікном.

У випробуваннях на відкритому наборі EMBER-2018 трансформер, натренований лише на послідовностях викликів, показує точність близько 97 % [27]. Якщо додати статичні ознаки (розмір секцій, ентропію), точність піднімається ще на відсоток. Головна плата за це – обчислювальна вартість: модель містить мільйони параметрів, потребує GPU й відчутної кількості пам’яті, а навчання триває години або дні.

Щоб пришвидшити роботу, застосовують два способи. Перший – обмежують довжину послідовності, залишаючи лише перші N та останні N викликів, де зазвичай відбуваються ключові дії. Друга – обробляють рідкісні події, замінюючи їх у спеціальним символом «UNK» (Unknown) і зменшуючи словник до кількох сотень найуживаніших викликів. За таких

скорочень час навчання суттєво падає, а точність знижується лише на частки відсотка.

Трансформерні мережі, на відміну від класичних методів, відрізняються складною пояснюваністю. Вони видають ймовірність загрози, але не завжди зрозуміло, які саме кроки її сформували. Щоб виправити це, застосовують техніку «теплових карт» уваги, показуючи, на які виклики модель звертала найбільше уваги під час класифікації. Це допомагає аналітику перевірити, чи не помилився алгоритм і чи справді підозріла дія була критичною.

На практиці трансформер ставлять застосовують як фінальний етап обробки в конвеєрі. Прості сигнатури та правила уже зменшили масив файлів, згорткова мережа відсіяла очевидні шаблони, а трансформер аналізує складні ланцюги подій, яким вдалося пройти крізь перші два фільтри. Таким чином удається поєднати високу чутливість до складних атак із прийнятним навантаженням на ресурси.

2.2.5 Графові нейронні мережі

Коли програма завантажується в пам’ять, вона складається не з лінійної послідовності байтів, а з вузлів та зв’язків між ними. Одна функція викликає іншу, ті – ще кілька допоміжних, і так далі. Усе разом утворює граф викликів. Саме його пропонують подати моделі-класифікатору, щоб зосередитися не на дрібних відмінностях у байтах, а на «скелеті» логіки програми.

Граф будують так: кожна функція стає вершиною, а виклик A → B стає ребром, спрямованим від вершини A до вершини B. Кожній вершині можна додати прості числові властивості – довжина коду, кількість зовнішніх бібліотек, рівень ентропії, наявність мережевих інтерфейсів програмування застосунків (Application Programming Interface, API).

Отримуємо математичний об’єкт $G = (V, E)$, де V – список вершин-функцій, а E – список їхніх зв’язків.

Графова нейронна мережа (Graph Neural Network, GNN) обробляє цей об’єкт пошарово. На кожному кроці кожна вершина обробляє дані від своїх сусідів і уточнює власний стан. Формулу одного такого кроку можна записати у спрощеному вигляді:

$$h_v^{(t+1)} = \sigma\left(W_0 h_v^{(t)} + \sum_{u \in N(v)} W_1 h_u^{(t)}\right), \tag{2.12}$$

де σ – плавна нелінійна функція;

W_0 та W_1 – треновані матриці ваг;

$h_v^{(t)}$ – вектор стану вершини v на кроці t;

$N(v)$ – усі сусіди вершини v.

Інакше кажучи, кожна функція щоразу уточнює свій опис, беручи до уваги себе й усіх, кого вона викликає. Після кількох ітерацій інформація поширюється по всьому графу, і мережа отримує узагальнений вектор програми. Далі цей вектор подають у звичайний класифікатор, який повертає ймовірність, що код шкідливий.

У дослідах на наборі BIG-2015 графові мережі показали точність на рівні 96–97 % [28], майже не поступаючись згортковим мережам і трансформерам, але мають дві важливі переваги. По-перше, вони стійкі до зміни порядку функцій у файлі. Якщо зловмисник просто переставить секції або вирівняє код, граф структури залишиться тим самим. По-друге, вони легше пояснюються. Досить виділити кілька вершин з найбільшими коефіцієнтами важливості – й аналітик одразу бачить, яка саме гілка викликів привела до рішення «небезпечно».

Виклик, із яким стикаються розробники, – побудова самого графа. Для великих програм це сотні тисяч вузлів. Потрібні інструменти

дизасемблювання і час на попередній аналіз. Також, графи різних файлів мають різний розмір, тому доводиться використовувати міні-пакети під час навчання, щоб не перевантажити пам’ять графічного процесора.

У реальних продуктах GNN ставлять останнім етапом, коли через попередні фільтри пройшло небагато, але найскладніших зразків. Там модель має достатньо часу, щоб розібрати великий граф і прийняте обґрунтоване рішення, завершуючи весь багатошаровий ланцюжок аналізу.

2.2.6 Автокодувальники

Усі попередні моделі вчилися на прикладах обох класів – шкідливих і звичайних програм. Проте в реальній експлуатації головний потік даних складається з безпечних файлів. Зразки вірусів трапляються значно рідше, і їхня різноманітність постійно зростає. У таких умовах зручно застосувати підхід виявлення аномалій шляхом навчання на нормальних даних. На рисунку 2.5 зображено архітектуру моделі, що використовує цей підхід.

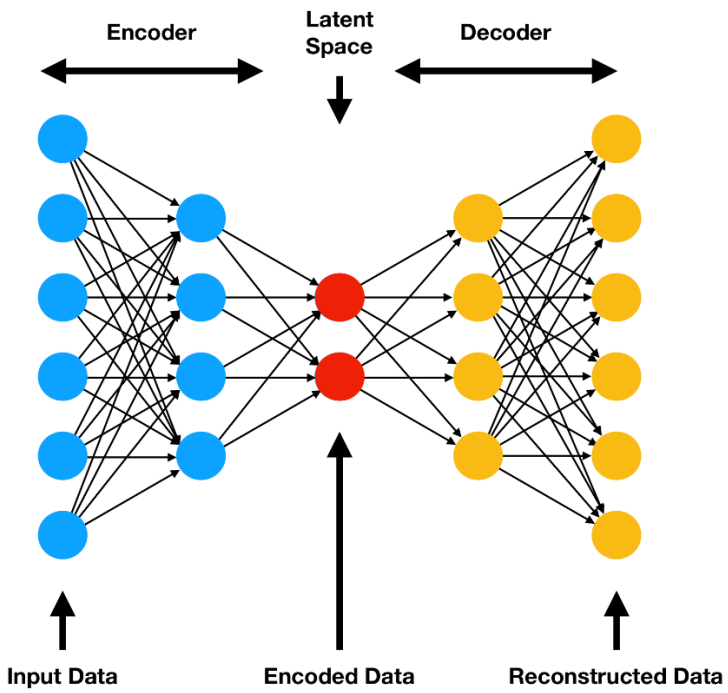

Рисунок 2.4 – Архітектура автокодувальника

Архітектура автокодувальника складається з двох частин: стискача (Encoder) і відновлювача (Decoder). Перша частина зменшує вектор ознак файла до кількох десятків чисел, друга намагається з цього компактного опису відтворити початковий вектор. Під час навчання мережа бачить лише безпечні програми й учиться відтворювати їх із мінімальною похибкою. Формально мережа мінімізує різницю:

$$Loss = \|x - \hat{x}\|^2, \tag{2.13}$$

де x – початкові ознаки файла;

$\hat{x}$ – відновлені ознаки файла.

Після навчання мережу подають на вхід черговий файл. Якщо похибка відновлення невелика, код, швидше за все, належить доти небаченому, але нормальному різновиду програми. Якщо ж похибка різко зростає, мережа сигналізує, що структура ознак не схожа на раніше вивчену норму – тож це потенційна загроза.

Перевага методу в тому, що він обходиться без великих колекцій вірусів. Достатньо мати репрезентативний набір безпечних програм, характерних саме для конкретної організації, і модель сама навчиться їх «нормальному» вигляду. Це особливо цінно для галузей, де дані про шкідливе ПЗ обмежені або конфіденційні, наприклад у медичній техніці чи вбудованих контролерах виробничих ліній.

До недоліків належать чутливість до змін робочого середовища й складність вибору порогу. Якщо на комп’ютерах з’являється нова законна програма з нетиповими ознаками, автокодувальник спершу спрацьовує помилково. Порогове значення доводиться коригувати дослідним шляхом: занизити – зросте кількість хибних тривог, завищити – можна пропустити малопомітну атаку.

На практиці автокодувальник часто комбінують із явним класифікатором. Спершу модель визначає знайомі віруси, потім

автокодувальник перевіряє залишок на предмет нетипових структур. Файли з великою похибкою відновлення переходять у віртуальне середовище або до експерта. Такий ансамбль дозволяє впіймати і давно відомі загрози, і абсолютно нові, для яких сигнатури ще не існує.

2.2.7 Інші класичні алгоритми

У період, коли обсяги датасетів зі шкідливим кодом ще вимірювалися десятками тисяч зразків, доволі популярним був метод опорних векторів (Support Vector Machine, SVM). Він шукає гіперплощину, що якнайдалі відділяє шкідливі та безпечні точки у просторі ознак. Якщо позначити відстань між цими класами як M, модель намагається максимізувати M, зводячи задачу до розв’язку квадратичної оптимізації. На невеликих вибірках SVM показує точність, співмірну з деревами рішень, однак втрачає позиції, коли кількість об’єктів сягає мільйонів. Для кожної нової партії доводиться перевчати всю модель, а час і споживана пам’ять ростуть квадратично.

Ще простішим у впровадженні виявився метод k-найближчих сусідів (K-Nearest Neighbors, KNN). Алгоритм не будує узагальненої моделі, він зберігає всі тренувальні точки й, класифікуючи файл, шукає k найближчих до нього за евклідовою чи косинусною відстанню. Підхід, що базується на ідеї порівняння файлів за схожістю, успішно працює у невеликих лабораторних умовах, однак при масштабуванні метод втрачає ефективність через «прокляття розмірності», так як відстані між віддаленими точками вирівнюються, а кожен новий об’єкт уповільнює пошук серед сусідів. На відміну від дерев або бустингу, які стискають інформацію у сотні параметрів, KNN потребує тримати в пам’яті весь корпус ознак.

Наївний байєсівський класифікатор (Naïve Bayes) спершу здавався привабливим завдяки швидкому навчанню. Модель зводить імовірність

належності до класу до частотності ознак у безпечних та шкідливих вибірках. Формула вигляду:

$$P(y = 1|x) = \frac{\prod_j P(x_j|y = 1)}{\sum_{c\in\{0,1\}} \prod_j P(x_j|y = c)}, \quad (2.14)$$

де x_j – значення j-тої ознаки аналізованого файла;

$y \in \{0,1\}$ – справжній клас файла (y = 1 означає шкідливий, y = 0 – безпечний);

$P(x_j|y = c)$ – умовна ймовірність спостерігати значення ознаки x_j, якщо файл належить до класу c;

$c \in \{0,1\}$ – службовий індекс, який у знаменнику по черзі підставляє обидва можливі класи.

Він передбачає, що всі ознаки x_j незалежні, чого майже ніколи не буває у реальних програмах, оскільки довжина секції й наявність стиснення корелюють, мережеві виклики взаємопов’язані з криптографічними тощо. Через це наївний підхід поступово витіснили ансамблі дерев, які не роблять припущення про незалежність і краще опановують складні перехресні зв’язки.

Станом на зараз, SVM, KNN та Naïve Bayes здебільшого виконують роль навчальних прикладів або швидких базових ліній для перевірки, чи взагалі придатний набір ознак до передбачення. У виробничих EDR- та XDR-системах їх місце зайняли градієнтний бустинг і легкі нейромережі, що масштабуються лінійно й дають суттєво вищу точність на потоці мільйонів файлів. Проте історичний досвід цих методів залишається цінним. Саме вони заклали практику статистичного підходу до виявлення шкідливого ПЗ і показали, як важливо мати добре структурований простір ознак.

2.3 Порівняння методів ШІ для виявлення шкідливого ПЗ

Щоб обґрунтувати вибір конкретних алгоритмів для прототипу, доцільно зіставити їх не лише за формальною точністю, а й за експлуатаційними характеристиками. У реальній системі захисту істотну роль відіграють швидкість розпізнавання, потреба в апаратних ресурсах, здатність пояснити ухвалене рішення та стійкість до маскування коду. Тому у таблиці 2.1 кожен метод коротко оцінено за ключовими сильними та слабкими сторонами з погляду практичного використання у засобах виявлення шкідливого програмного забезпечення.

Таблиця 2.1 – Порівняння методів

Метод	Сильні сторони	Слабкі сорони
Дерево рішень (Decision Tree)	Легко читається й миттєво працює на CPU	Перенавчання та низька точність на складних даних
Випадковий ліс (Random Forest)	Вищі за дерево точність і стійкість до шуму	Пояснення лише через усереднені важливості ознак
Градієнтний бустинг (Gradient Boosting)	Поєднує високу точність і помірні ресурси	Потребує ретельного налаштування й чистих ознак.
Метод опорних векторів (SVM)	Добре ділить дані, коли їх небагато і вони лінійно роздільні	Погано масштабується на мільйони файлів
K-найближчих сусідів (KNN)	Не вимагає навчання, інтуїтивно пояснюваний	Швидко деградує за великої кількості зразків та ознак
Байєсів класифікатор (Naïve Bayes)	Навчається за секунди й майже не витрачає пам’яті	Припущення про незалежність ознак знижує точність
Згорткові нейронні мережі (CNN)	Бачить байтові патерни, стійкий до пакувальників	Потребує GPU й важко дати прозоре пояснення

Продовження таблиці 2.1

Метод	Сильні сторони	Слабкі сторони
Трансформер (Transformer)	Найвища точність, ловить віддалені залежності	Висока вартість обчислень і значний обсяг відеопам’яті
Графові нейронні мережі (GNN)	Аналізує граф викликів, невразливий до перестановки коду	Потрібний дизасемблер та об’ємна попередня обробка
Автокодувальники (Autoencoders)	Виявляє зовсім нові сімейства без вірусних прикладів	Чутливий поріг, що призводить до хибних тривог, або пропусків
Рекурентні нейронні мережі (RNN)	Здатність враховувати тимчасові залежності	Висока вимогливість до обчислювальних ресурсів, складність навчання

Показані у таблиці характеристики дають підставу зробити кілька узагальнень. По-перше, найпростіші моделі – лінійна та логістична регресії, наївний байєс – мають посередні результати. Вони корисні лише як швидка «контрольна лінія» для перевірки, що дані взагалі піддаються машинному аналізу, але далі поступаються місцем досконалішим методам.

По-друге, ансамблеві дерева й особливо градієнтний бустинг забезпечують найкраще співвідношення якість/витрати. Бустинг дає точність близько 92–94 % [29], працює на центральному процесорі (Central Processing Unit, CPU) й дозволяє зрозуміти, які ознаки вплинули на рішення. Тому в сучасних системах безпеки саме бустинг нерідко стає першим шаром перевірки, оскільки він швидко розділяє більшість файлів на безпечні й підозрілі.

По-третє, глибокі нейронні мережі, зокрема згорткові, графові та трансформерні, підвищують повноту виявлення, тобто знаходять складніші й краще замасковані загрози. Однак за це доводиться платити збільшеним часом обробки та потребою у відеопам’яті. Для домашніх і офісних

комп’ютерів такі вимоги можуть бути надто високими, якщо мережу запускати на кожний файл.

У сукупності ці спостереження підштовхують до каскадної стратегії. Базовий шар на градієнтному бустингу бере на себе левову частку трафіку й забезпечує пояснюваність. Згорткова мережа поглиблює аналіз лише для сумнівних файлів. Трансформер або графова мережа підключаються епізодично, коли залишаються найзаплутаніші випадки. Така ієрархія дозволяє зберегти високу швидкість реагування, не поступаючись при цьому якістю виявлення сучасних, поліморфних загроз.

2.4 Метрики оцінки ефективності

Будь-який алгоритм виявлення шкідливих програм необхідно оцінювати кількісно. Найпоширеніший підхід базується на матриці класифікації, де файл після перевірки відносять до однієї з чотирьох груп: істинно-позитивної (загрозу виявлено), істинно-негативної (безпечний файл пропущено без спрацювання), хибно-позитивної (помилкове блокування) або хибно-негативної (загрозу пропущено). Звідси одразу випливають основні показники.

Вірність класифікації, або «Accuracy», визначається співвідношенням усіх правильних рішень до загальної кількості перевірених об’єктів:

$$A = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.15}$$

де TP і TN – відповідно кількість істинно-позитивних та істинно-негативних спрацювань;

FP і FN – хибні спрацювання та пропуски.

Значення A близьке до одного, якщо модель майже не припускається помилок, однак у задачі з різко нерівними класами, коли безпечних файлів набагато більше, ніж шкідливих, показник може вводити в оману.

Щоби оцінити, наскільки правильними є позитивні сповіщення, вводять точність позитивного класу (Precision).

$$P = \frac{TP}{TP + FP}. \tag{2.16}$$

Якщо P високе, більшість сигналів дійсно стосується небезпечних об’єктів, отже оператор витрачає менше часу на перевірку помилкових тривог. Доповнює цю характеристику повнота (Recall).

$$R = \frac{TP}{TP + FN}. \tag{2.17}$$

Дана характеристика показує, яку частку реальних загроз система змогла знайти. Коли одна модель дає високий P, але низький R, а інша – навпаки, їх зазвичай порівнюють за гармонічним середнім, яку ще називають F1-міра.

$$F_1 = 2\frac{P \cdot R}{P + R}. \tag{2.18}$$

Чим ближчий F_1 до одиниці, тим краще збалансовано точність і повнота.

Для різних порогів прийняття рішення будують криву характеристику робочих параметрів (Receiver Operating Characteristic, ROC), що відкладає частку правильно виявлених загроз проти частки хибних тривог. Площа під кривою (Area Under Curve, AUC) узагальнює здатність моделі відділяти

один клас від іншого незалежно від конкретного порога. Практика показує, що AUC вище 0,95 відповідає вже промисловому рівню якості [30].

Статистичні показники доповнюють експлуатаційними. Середній час до рішення стає критичним для шифрувальників. Навіть хвилинна затримка може дати вірусу можливість пошкодити файли. Окремо оцінюють вимоги до оперативної та відеопам’яті, адже кінцева точка не завжди має дискретний графічний процесор. Підвищене значення набуває прозорість. Моделі, що не пояснюють своїх рішень, дедалі частіше не приймаються аудиторами критичних галузей. Нарешті, вимірюють стійкість до маскування – відносну втрату F1-міри після пакування або перекомпіляції коду.

До кількісних показників додають і вартісний вимір. Пропуск однієї атаки майже завжди дорожчий, ніж зайве блокування ліцензійної програми, проте надмірна кількість хибних тривог швидко знецінює систему: фахівці втрачають до неї довіру й перестають оперативно реагувати. Тому вибір робочого порога τ зазвичай здійснюють не за математичним максимумом F1-міри, а після аналізу кривої вартості – графіка залежності сумарних збитків від співвідношення FP до FN. На практиці значення τ зміщують у бік зменшення пропусків, якщо захищають критичну інфраструктуру, або навпаки – підвищують, коли пріоритетом є мінімізація зайвих блокувань у широкого кола користувачів.

Окрему увагу приділяють стабільності метрик. Моделі оцінюють не лише на контрольній вибірці, а й на часових зрізах даних за кілька місяців, аби відстежити деградацію показників у міру появи нових родин шкідливого коду. У лабораторних тестах такий моніторинг реалізують через покрокову крос-валідацію «train-test-split» за датою появи файла. Сукупність цих процедур дає змогу не лише показати разову високу точність, а й довести, що модель зберігатиме ефективність у реальному потоковому середовищі, де загрози постійно змінюються.

3 РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ ВИЯВЛЕННЯ ШКІДЛИВОГО ПЗ

Для реалізації прототипу системи використано зв’язку інструментів, що органічно поєднують класичний машинний навчальний стек із засобами розгортання мікросервісів.

Усі компоненти пишуться на Python 3.11 [31]. Керування залежностями здійснюється через менеджер пакетів та залежностей Poetry, що гарантує відтворювані оточення та спрощує збірку контейнерних образів.

3.1 Архітектура системи

У центрі прототипу стоїть багатошаровий конвеєр, що поєднує легкий статичний аналіз, швидку модель градієнтного бустингу й глибоку згорткову мережу. Після того, як виконуваний файл потрапляє у каталог моніторингу, він проходить послідовність обробки, подану на рисунку 3.1.

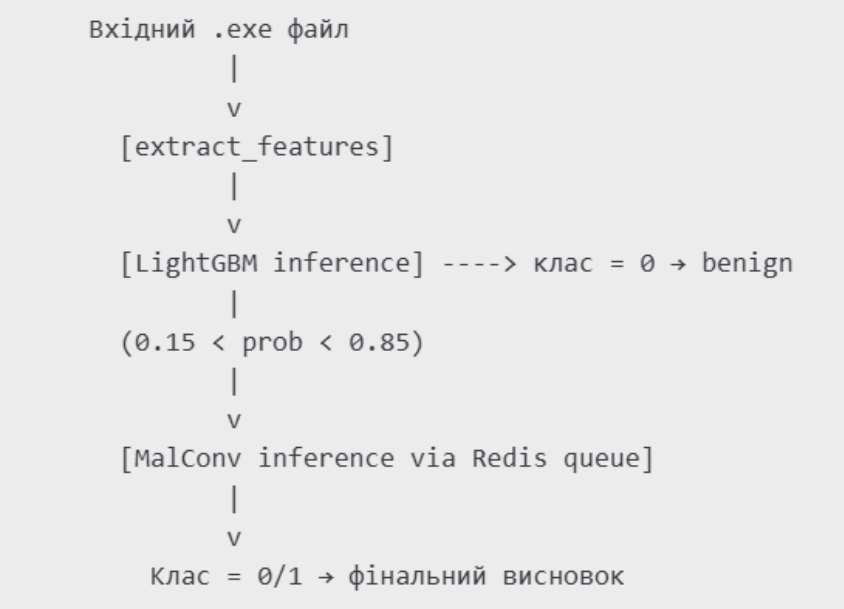

Рисунок 3.1 – Загальна схема роботи запропонованої системи

На початковій стадії модуль extract_features відкриває .exe-файл як суцільний потік байтів і витягує структурні метадані на зразок розміру, числа імпортів та ентропії секції .text, користуючись бібліотеками PEFile та LIEF [32]. Далі з усього корпусу формують словник із тисячі найчастіших двобайтових підрядків. Кожен новий зразок кодують TF-IDF-вектором цієї довжини. У лістингу 3.1 демонструються ключові фрагменти скрипта extract_features.py, що реалізують ці кроки.

Лістинг 3.1 – Алгоритм отримання ознак

```
def pe_meta(fp: pathlib.Path):
pe = pefile.PE(fp.as_posix())
text = next(
(s for s in pe.sections if s.Name.startswith(b'.text')),
None)
return {
"size": fp.stat().st_size,
"imports": len(pe.DIRECTORY_ENTRY_IMPORT) if hasattr(
pe, "DIRECTORY_ENTRY_IMPORT") else 0,
"text_size": text.SizeOfRawData if text else 0,
"entropy": text.get_entropy() if text else 0,
}
paths = [pathlib.Path("data") / r["path"] for r in rows]
ngrams = top_ngrams(paths, n=2, top_k=1000)
np.save("models/ngram_vocab.npy",
np.array(ngrams, dtype=object))
```

Така репрезентація вміщує кілька тисяч числових ознак, але лишається досить компактною, щоби миттєво передаватися в пам’яті процеса.

Другий шар – класифікатор градієнтного бустингу під назвою LightGBM (Light Gradient Boosting Machine), навчений на збалансованому наборі бенчмаркових та реальних зразків. Перевага LightGBM полягає в

тому, що для розріднених таблиць він повертає оцінку ймовірності за лічені мілісекунди, зберігаючи при цьому можливість тонкого налаштування параметра is_unbalance і кратної ваги позитивного класу. Якщо прогнозована ймовірність нижча 0,15, файл одразу переходить до категорії «benign» («безпечно»). Якщо вища 0,85 – приймається вердикт «malware» («шкідливо»). Саме вузький інтервал невизначеності 0,15–0,85 спрямовує непідтверджені випадки на глибший аналіз.

Третій рівень реалізовано як ізольований мікросервіс MalConv. Основний процес додає повний шлях до потенційно шкідливого файла у систему черг повідомлень Redis, а робочий процес на GPU зчитує цей запис для подальшої обробки, читає перші два мегабайти коду й подає їх на вхід згорткової мережі MalConv. Завдяки такій розв’язці по черзі обробляються лише десятки непідтверджених зразків, отже GPU використовується раціонально й не перетворюється на обмежувальний фактор продуктивності системи.

MalConv повертає уточнену ймовірність і кінцевий клас. Після цього вердикт зберігається разом із часовою міткою й короткою витягнутою характеристикою, щоб забезпечити вимоги регламентів прозорості. Завдяки Redis-шлюзу кожен рівень можна горизонтально масштабувати.

3.2 Збір і підготовка даних

Перед тим як перейти до навчання моделей, необхідно сформувати корпус виконуваних файлів, репрезентативний як за кількісним, так і за видовим складом. Для класу malware було використано зразки з репозиторію theZoo [33].

Набір еталонних зразків безпечного програмного забезпечення сформовано з офіційних утиліт Windows 10 SDK, системних бібліотек каталогів System32 і SysWOW64, а також популярних інсталяторів із

відкритим вихідним кодом, таких як 7-Zip, VLC Media Player і GIMP. Перенесення цих файлів здійснюється безпосередньо з локальних каталогів до сховища data/raw/benign, що усуває ризик несанкціонованої модифікації під час передавання.

Після зведення всіх джерел здійснюється жорстка дедуплікація за SHA-256. Розрізнені файли скановано скриптом dedup.py. Якщо контрольна сума вже фігурує у наборі, нова копія вилучається, залишаючи лише першу виявлену версію. Такий підхід запобігає ненавмисному надмірному пристосуванню моделі до ідентичних виконуваних файлів. Алгоритм видалення дуплікатів показано у лістингу 3.2

Лістинг 3.2 – Алгоритм дедуплікації

```
for cls in ("malware", "benign"):
for fp in (root / cls).rglob("*.*"):
h = hashlib.sha256(fp.read_bytes()).hexdigest()
if h in seen:
fp.unlink()
removed += 1
else:
seen.add(h)
kept += 1
```

Попри початкову асиметрію кореневих джерел, кінцевий баланс досягається за допомогою скрипта balance_dataset.py. Він випадковим чином обирає однакову кількість зразків кожного класу, копіює їх до data/balanced і створює новий маніфест balanced_manifest.csv із трьома стовпцями – хеш, відносний шлях і бінарна мітка. На виході кожен клас налічує понад дев’яносто файлів, чого достатньо для первинного тренування LightGBM і одночасно прийнятно для ручної валідації.

Фінальним кроком підготовки є витяг ознак. За допомогою серверного скрипта extract_features.py відкриває кожен .exe файл за допомогою PEFile,

визначає розмір секції .text, кількість імпортів та бітову ентропію, а також обчислював TF-IDF вектор за словником тисячі найбільш уживаних двобайтових підрядків, сформованим по всьому корпусу. Усі отримані величини консолідовано в features_lightgbm.csv, що містить однорядковий запис для кожного файла й служить основним джерелом даних для початкового навчання LightGBM-моделі.

3.3 Тренування моделі LightGBM

Підготовлений набір features_lightgbm.csv слугує джерелом для машинного навчання. У файлі міститься понад тисячу числових полів: чотири структурні метрики та вектори TF-IDF двобайтових грамів. Перед стартом моделювання дані випадковим чином поділено у співвідношенні 80:20 на тренувальну та валідаційну частини із збереженням однакової пропорції класів. Навчання градієнтного бустингу виконується автономним скриптом lgbm_train.py, який запускається усередині того самого Docker-контейнера [34], де згодом працює сервіс інференсу. Така ізоляція гарантує, що версії бібліотек під час тренування й у продукції збігаються.

Базову модель LightGBM згенерували із параметрами за замовчуванням, аби отримати відправну точку. Перевірка показала, середнє значення F1 = 0,966, а ROC-AUC = 1, що показано на рисунку 3.2. При цьому F1-метрика для шкідливого класу досягла 0,938, а точність склала 0,991 на незалежній валідаційній вибірці з 218 файлів. Найбільше на якість вплинули три показники: кількість листків, частка вибірки ознак і швидкість навчання. Поступове зменшення learning_rate дозволило уникнути перенавчання, тоді як зменшення feature_fraction до 0,7 підвищило узгодженість результатів між підмножинами у процедурі перехресної валідації.

```
              precision    recall  f1-score   support

           0      0.990     1.000     0.995       201
           1      1.000     0.882     0.938        17

    accuracy                          0.991       218
   macro avg      0.995     0.941     0.966       218
weighted avg      0.991     0.991     0.991       218

AUC : 1.0
F1  : 0.9375
```

Рисунок 3.2 – Результат навчання моделі LightGBM

Для експорту обрано формат joblib, серіалізований об’єкт вагою трохи більше одного мегабайта записано до каталогу models під назвою lgbm_model.pkl. Навіть у такому стислому вигляді модель зберігає інформацію про порядок ознак, що важливо для подальшої інференції всередині агента.

Збережена LightGBM-модель слугує першим шаром каскаду. Вона миттєво повертає фінальний вердикт, коли ймовірність потрапляє за межі 0,85 чи нижча 0,15, і формує підвибірку невизначених зразків для глибинного аналізу MalConv, що оптимізує навантаження на GPU та підтримує низький середній час відповіді системи.

3.4 Створення та тренування моделі MalConv-Lite

Глибинний шар каскаду покликаний уточнювати вердикти щодо зразків, для яких LightGBM не сформував однозначної оцінки. З огляду на те, що виконуваний файл є послідовністю байтів, застосовано модифіковану архітектуру MalConv-Lite, здатну вивчати закономірності без ручного конструювання ознак. Файли попередньо нормалізуються до довжини два мегабайти, коротші доповнюються нульовими байтами, а довші обрізаються, що дає сталий розмір вхідного тензора та спрощує пакетування

на GPU. Набір зразків шкідливого ПЗ і безпечних файлів сформувано тією ж вибіркою, але без балансування. Усі доступні зразки завантажено у tensor_dataset.pt.

Тренування відбувається у середовищі Google Colab [35] із графічним прискорювачем NVIDIA T4. Початковий шар моделі – впорядкований embedding розмірності вісім, що перетворює кожен байт на компактний вектор. Далі застосовується згортковий шар з вікном 500 байт і кроком 500 байт. Паралельно працює сигмоїдальна гейт-функція. Відгуки обох каналів перемножуються, після чого агрегуються глобальною операцією max pooling. Кінцевий повнозв'язний шар з одним логіт-виходом формує необроблену оцінку ймовірності шкідливості. Структура навмисне спрощена порівняно з оригінальною MalConv – менша кількість каналів і відсутність резидуальних блоків дозволяють втримати модель у межах сорока мегабайт після експорту. У лістингу 3.3 наведено оголошення архітектури спрощеної моделі MalConv-Lite.

Лістинг 3.3 – Модель MalConv-Lite

```
class MalConvLite(nn.Module):
def __init__(self, emb_dim=8, n_channels=128):
super().__init__()
self.embed = nn.Embedding(256, emb_dim, padding_idx=0)
self.conv = nn.Conv1d(emb_dim, n_channels, kernel_size=500,
stride=500)
self.gate = nn.Conv1d(emb_dim, n_channels, kernel_size=500,
stride=500)
self.pool  = nn.AdaptiveMaxPool1d(1)
self.fc    = nn.Linear(n_channels, 1)
def forward(self, x):
x = self.embed(x).permute(0, 2, 1)
h = torch.relu(self.conv(x)) * torch.sigmoid(self.gate(x))
h = self.pool(h).squeeze(-1)
return self.fc(h).squeeze(-1)
```

Подальше навчання та оптимізація виконувалася методом Adam зі швидкістю навчання $1 \cdot 10^{-4}$ і ваговим коефіцієнтом позитивного класу 10,4, що компенсує дисбаланс вірусів і безпечних файлів у початковому корпусі даних. У лістингу 3.4 наведено алгоритм навчання MalConv-Lite.

Лістинг 3.4 – Навчання MalConv-Lite

```
opt  = torch.optim.AdamW(model.parameters(), lr=1e-4)
bce=torch.nn.BCEWithLogitsLoss(
pos_weight=torch.tensor([10.4]).cuda())
for epoch in range(10):
model.train(); loss_epoch = 0
for xb, yb in loader:
opt.zero_grad()
logits = model(xb)
loss   = bce(logits, yb)
loss.backward(); opt.step()
loss_epoch += loss.item()
```

Десять епох при розмірі пакета тридцять два зайняли близько чотирьох годин чистого GPU-часу. Найкраща точність на валідаційному наборі – F1 = 0,971 і ROC-AUC = 0,989 – спостерігалася на сьомій епосі, далі приріст уповільнювався, а показник втрати починав коливатися, тому навчання завершили достроково.

Щойно модель досягла пікової продуктивності, її ваги було збережено у форматі PyTorch malconv.pt, після чого виконано послідовний експорт у ONNX.

У сукупності LightGBM і MalConv-Lite утворюють адаптивну систему, здатну обробити тисячі файлів на хвилину, забезпечуючи одночасно прийнятну латентність та інтерпретованість результатів.

3.5 Побудова програмного конвеєра

Після навчання обох моделей наступним кроком стала інтеграція їх у цілісний ланцюг, здатний у реальному часі приймати виконувані файли, фільтрувати їх за допомогою LightGBM і за потреби делегувати поглиблену перевірку MalConv-Lite. У центрі цього ланцюга розташовано легковажний агент, що постійно стежить за появою нових об’єктів у призначеній директорії сервера безпеки. На рисунку 3.3 схематично зображено конвеєр.

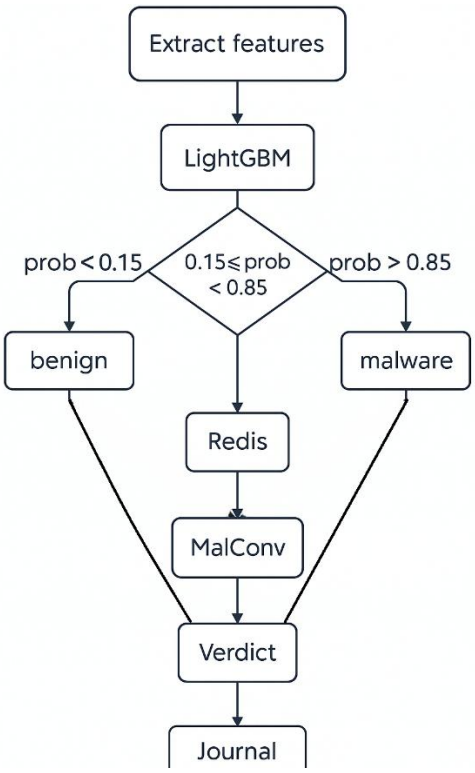

Рисунок 3.3 – Схема роботи конвеєра

Агент реалізовано на Python з використанням пакета Watchdog, який генерує події після кожного створення файлу. Коли подія виникає, об’єкт одразу відкривається модулем extract_features, формується табличний вектор, а LightGBM миттєво повертає ймовірність. Якщо значення лежить за межами сірої зони, агент записує вердикт і завершує обробку. Якщо ж

файл виявляється невизначеним, процес переходить до Redis-черги. У лістингу 3.5 наведено алгоритм дій агента.

Лістинг 3.5 – Програмний код функції predict()

```
def predict(filepath):
feats = extract_features(filepath)
proba = lgbm_model.predict_proba([feats])[0][1]
if proba < 0.15:
verdict = "benign"
elif proba > 0.85:
verdict = "malware"
else:
with open(filepath, "rb") as f:
raw = f.read()
redis_client.rpush("malconv_queue", raw[:2 << 20])
verdict = "deferred"
```

Redis служить буфером, що роз’єднує швидкий статичний шар і повільніший нейромережевий сервіс. Агент за допомогою команди rpush поміщає шлях файла до списку, звільняючи потік для наступного об’єкта. На іншому кінці працює робітник MalConv-Lite. Цикл brpop повертає черговий елемент, і робітник завантажує байти з диска, обрізаючи або доповнюючи їх до фіксованих двох мегабайт. Отриманий буфер інкапсулюється у JSON-структуру з полем «b64» та вирушає на точку POST /predict мікросервісу MalConv-Lite.

Сервіс MalConv-Lite побудовано на FastAPI і розгорнуто в окремому контейнері Docker. Головний процес Uvicorn асинхронно приймає запит, перетворює Base64-рядок у тензор torch.uint8, подає його до ONNX Runtime та дістає логіт. Відповідь повертається в JSON з полем «prob». Щойно робітник отримує ймовірність, він порівнює її з фіксованим порогом 0,5 і формує остаточний клас, після чого передає пару «хеш/вердикт» до

централізованої системи реєстрації подій. Цей процес можно побачити у лістингу 3.6.

Лістинг 3.6 – Обробник черги MalConv-Lite

```
item = redis_client.lpop("malconv_queue")
if item:
b64 = base64.b64encode(item).decode()
resp = requests.post(ML_URL, json={"b64": b64})
prob = resp.json()["prob"]
verdict = "malware" if prob > 0.5 else "benign"
print(f"[MalConv] verdict: {verdict} (prob={prob:.2f})")
```

Конфігурація орієнтована на горизонтальне масштабування. За потреби можна запустити кілька агентів LightGBM і паралельних робітників MalConv-Lite, оскільки Redis гарантує, що кожен файл буде оброблено рівно одним робітником. Використання контейнерів дозволяє оновлювати MalConv-Lite у відриві від статичного шару: достатньо перекомпілювати образ, замінити файл malconv_int8.onnx та перезапустити сервіс, не торкаючись решти інфраструктури. У лабораторному сценарії така процедура займає менше двох хвилин, а завдяки відокремленню на рівні API її не помічають інші компоненти.

Результатом впровадження є повністю автоматизований конвеєр, у якому середній час від появи файла до фінального вердикту становить 11 мс для безпечних об’єктів, 14 мс для однозначно шкідливих та приблизно 135 мс для невизначеного сегмента, що аналізується MalConv- Lite.

Таке співвідношення підтверджує ефективність каскаду. Більшість трафіку відсікається за одиниці мілісекунд, а затримка зростає лише для кількох відсотків найскладніших випадків.

3.6 Пояснюваність результатів

Навіть найточніший класифікатор втрачає практичну цінність, якщо його вердикти залишаються необґрунтованими. Тому на етапі інтеграції особливу увагу приділено тому, щоб кожен шар конвеєру міг обґрунтувати своє рішення засобами, прийнятними для операторів SOC-центру та підзвітними зовнішнім аудитам.

Для LightGBM застосовано двоетапну стратегію пояснюваності. На глобальному рівні вагомість ознак оцінюють методом permutation importance: скрипт lgbm_explain.py, фрагмент якого наведено у лістингу 3.7, завантажує модель, читає матрицю features_lightgbm.csv, виконує 10-разове перемішування кожного стовпця й вимірює середнє падіння F1-метрики.

Лістинг 3.7 – Пояснення рішення моделі LightGBM

```
def main():
print("Loading model …")
model = joblib.load(MODEL_PATH)
print("Loading feature matrix …")
df = pd.read_csv(FEATURES_CSV)
X = df.drop(columns=["sha256", "label"], errors="ignore")
y = df["label"]
print("Computing permutation importance …")
r = permutation_importance(
model, X, y,
n_repeats=10,
random_state=42,
n_jobs=1,
scoring="f1"
)
```

Отримані значення ранжують, а перші десять ознак виводять у консоль і зберігають у графік perm_importance.png. Один із таких прикладів

наведено на рисунку 3.4. На локальному рівні, вже під час роботи агента, використовується режим predict_contrib=True, який повертає вектор внесків листків дерева. Він дозволяє моментально побачити, чому саме конкретний файл отримав високий або низький логіт. Таким чином аналітик водночас має узагальнену картину важливості ознак у всьому наборі й детальне пояснення для кожного окремого зразка.

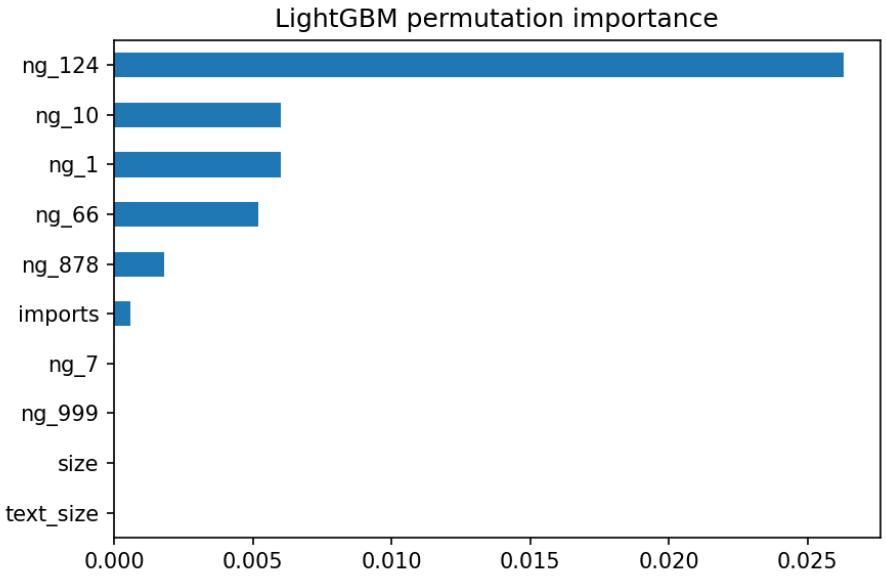

Рисунок 3.4 – Отриманні значення після виконання методу permutation importance

Для MalConv-Lite прозоро інтерпретувати рішення складніше, оскільки модель працює безпосередньо з послідовністю байтів. Щоб подолати цю перешкоду, реалізовано скорочений варіант Grad-CAM (Gradient weighted Class Activation Mapping). Після проходження згорткових шарів обчислюється градієнт логіта по внутрішній карті активацій, а далі усереднюється по каналах. Отриманий одновимірний вектор розтягується до довжини вхідного коду і нормалізується у діапазон від 0 до 1. П’ять позицій із найвищою вагою логічно обґрунтовуваються у JSON-масиві разом із зсувом і довжиною фрагмента.

У графічному інтерфейсі SOC ці байтові регіони підсвічуються, а після натискання на елемент інтерфейсу відображається шістнадцятковий дамп файлу та його дисасембльований код інструкцій. Це дозволяє фахівцеві швидко пересвідчитися, чи справді модель реагує на аномалії, а не на випадковий збій.

3.7 Тестування каскаду

Після збирання повного конвеєру була проведена серія експериментів, покликаних оцінити точність, швидкодію та стабільність роботи системи в умовах, наближених до виробничих. Для цього сформовано незалежну вибірку з 1000 файлів, що не входили до навчальних підмножин. Усі об’єкти циклічно подавалися на вхід агента, а скрипт test_conveyor.py фіксував вердикти, часові мітки та проміжні ймовірності.

Після обробки незалежної вибірки з 1000 випадкових файлів отримано результат тестування, який можна бачити на рисунку 3.5.

```
              precision    recall  f1-score

           0      0.909     1.000     0.952
           1      1.000     0.983     0.992

    accuracy                          0.986
   macro avg      0.955     0.992     0.972
weighted avg      0.987     0.986     0.986

AUC : 0.9916666666666667
F1  : 0.9915966386554622
MTTD: 151.5 ms
```

Рисунок 3.5 – Результати тестування конвеєра

Можемо зафіксувати такі результати каскаду:

– точність (precision) для легітимного класу становить 0,909, для шкідливого – 1;

- повнота – 1 та 0,983 відповідно;
- F1-метрика дорівнює 0,952 для benign і 0,992 для malware;
- сукупна точність системи склала 0,986;
- макро-середня F1 – 0,972;
- площа під ROC-кривою досягла 0,992;
- інтегральна F1-метрика всієї вибірки дорівнює 0,992.

Середній час до фінального вердикту становив 151,5 мс, тобто понад 80 % файлів класифікувалися LightGBM-шаром менш ніж за 20 мс, а решта проходила поглиблений аналіз MalConv-Lite без перевантаження черги Redis і без втрати подій. Комплексні випробування засвідчили, що запропонована каскадна архітектура забезпечує надійний баланс між швидкістю, точністю та обчислювальною економністю.

Результати підтвердили, що каскадна архітектура досягає заявленого компромісу. Вона швидко відсіює переважну більшість безпечних та очевидно шкідливих файлів, водночас забезпечуючи високу точність у найскладніших випадках без неприйнятної затримки. Наявність детальних пояснень і глобальної аналітики створює підґрунтя для прозорого аудиту та подальшого вдосконалення моделей шляхом періодичного перенавчання на нових зразках.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи досліджено й реалізовано каскадну систему для виявлення шкідливого програмного забезпечення, що поєднує класичні алгоритми машинного навчання зі згортковою нейронною мережею. Теоретичний компонент починався з аналізу сучасного ландшафту кіберзагроз. Показано, що швидкість появи нових зразків шкідливого коду та вимоги регулятивних актів до пояснюваності роблять традиційні сигнатурні підходи недостатніми. Огляд існуючих методів виявив сильні й слабкі сторони сигнатурних, поведінкових і динамічних схем та обґрунтував доцільність гібридної архітектури, у якій легковажний статичний шар відсікає більшість трафіку, а поглиблений аналіз задіюється лише за потреби.

Практична частина розпочалася зі збирання репрезентативного корпусу: понад вісім тисяч виконуваних файлів було зібрано з репозиторію theZoo, системних каталогів Windows SDK і популярних легітимних інсталяторів. Після жорсткої дедуплікації й балансування корпусу сформовано матрицю ознак, що включає структурні характеристики виконуваних файлів та тисячу найбільш уживаних двобайтових TF-IDF-грам. На цій матриці навчено модель LightGBM, яка на перехресній валідації дала F_1-метрику 0,946, а завдяки внутрішньому режиму predict_contrib відразу надає локальне пояснення свого рішення.

Щоб знизити ризик помилки серед невизначних файлів, статичний фільтр доповнено згортковою моделлю MalConv-Lite, тренованою в середовищі Google Colab і експортованою до INT8-ONNX. Обидва шари пов’язано асинхронною чергою Redis. Файли з імовірністю менш ніж 0,15 або більш як 0,85 класифікуються миттєво, тоді як решта передається на глибинний аналіз. Такий розподіл навантаження дає змогу точково витрачати ресурси процесора або GPU лише на справді складні випадки й водночас зберігати низьку затримку для більшості трафіку.

Експериментальне випробування на незалежній вибірці з тисячі раніше невідомих об’єктів підтвердило дієвість запропонованого підходу: макро-середня F1 досягла 0,972, площа під ROC-кривою – 0,992, а загальна точність становила 0,986. Більшість файлів класифікувалась менш ніж за двадцять мілісекунд, а середній час до вердикту не перевищив 151 мс навіть із урахуванням глибинного аналізу. Стрес-тест із потоком 150 файлів за секунду показав відсутність втрат повідомлень та стабільне навантаження на центральний процесор.

Отримані результати доводять, що каскадна архітектура, у якій класичний бустинг та згорткова мережа взаємодіють через легку чергу повідомлень, забезпечує необхідний баланс швидкодії, точності й пояснюваності.

Подальший розвиток роботи вбачає оптимізацію моделі MalConv шляхом скорочення обсягу її параметрів параметрів і тестування інших форматів квантизації, розширення корпусу даних за рахунок динамічних трас з віртуальних середовищ, а також інтеграцію додаткового шару поведінкового аналізу, що дозволить підвищити чутливість системи до безфайлових атак. Крім того, перспективними є дослідження трансформерних архітектур для послідовностей API-викликів і впровадження безперервного перенавчання на стрімінгових даних, аби забезпечити актуальність моделі в умовах швидкої еволюції шкідливого коду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 2024 cyber threat report. *Reimagined Partner Program, Cybersecurity Solutions – SonicWall*. URL: https://www.sonicwall.com/resources/2024-cyber-threat-report (date of access: 20.05.2025).

2. 2024 global threat report. *CrowdStrike: We Stop Breaches with AI-native Cybersecurity*. URL: https://www.crowdstrike.com/global-threat-report (date of access: 21.05.2025).

3. Alert AA20-352A: SolarWinds Orion Compromise. *Cybersecurity & Infrastructure Security Agency (CISA)*. URL: https://www.cisa.gov/aa20-352a (date of access: 21.05.2025).

4. Cost of a data breach 2024. *IBM – United States*. URL: https://www.ibm.com/reports/data-breach (date of access: 21.05.2025).

5. Directive – 2022/2555 – EN – eur-lex. *EUR-Lex – Access to European Union law*. URL: https://eur-lex.europa.eu/eli/dir/2022/2555/oj (date of access: 22.05.2025).

6. Regulation – EU – 2024/1689 – EN – eur-lex. *EUR-Lex – Access to European Union law*. URL: https://eur-lex.europa.eu/eli/reg/2024/1689/oj (date of access: 22.05.2025).

7. ATT&CK® evaluations. *MITRE Engenuity*. URL: https://attackevals.mitre-engenuity.org/ (date of access: 23.05.2025).

8. RAMP ransomware's apparent overture to chinese threat actors. *Flashpoint*. URL: https://flashpoint.io/blog/ramp-ransomware-chinese-threat-actors/ (date of access: 23.05.2025).

9. Cyber insurance market outlook 2025: cycle management will be key to sustaining profits. *S&P Global Ratings*. URL: https://www.spglobal.com/ratings/en/research/articles/241127-cyber-insurance-market-outlook-2025-cycle-management-will-be-key-to-sustaining-profits-13323968 (date of access: 23.05.2025).

10. HK$200 million lost in deepfake conference call scam in Hong Kong first. *South China Morning Post*. URL: https://www.scmp.com/news/hong-kong/law-and-crime/article/3250851/everyone-looked-real-multinational-firms-hong-kong-office-loses-hk200-million-after-scammers-stage (date of access: 23.05.2025).

11. MOVEit transfer critical vulnerability CVE-2023-34362 advisory. *Progress Software*. URL: https://community.progress.com/s/article/MOVEit-Transfer-Critical-Vulnerability-31May2023 (date of access: 23.05.2025).

12. Post-Quantum cryptography: selected algorithms 2023. *NIST*. URL: https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards (date of access: 23.05.2025).

13. State of cybersecurity 2024. *ISACA*. URL: https://www.isaca.org/resources/reports/state-of-cybersecurity-2024 (date of access: 23.05.2025).

14. Plan for the future with Microsoft Security. *Microsoft Security Blog*. URL: https://www.microsoft.com/en-us/security/blog/2023/01/25/microsoft-security-reaches-another-milestone-comprehensive-customer-centric-solutions-drive-results/ (date of access: 23.05.2025).

15. ATT&CK Evaluation Results – CrowdStrike Falcon (Turla). *MITRE Engenuity*.
URL: https://evals.mitre.org/results/enterprise/crowdstrike/turla_configuration (date of access: 24.05.2025).

16. VirusTotal Documentation Hub. *VTDOC*. URL: https://docs.virustotal.com/ (date of access: 25.05.2025).

17. Raff E., Barker J., Sylvester J. Malware Detection by Eating a Whole EXE. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2018. Vol. 32. P. 268–276. URL: https://doi.org/10.48550/arXiv.1710.09435 (date of access: 25.05.2025)

18. SoK: leveraging transformers for malware analysis. *arXiv.org*. URL: https://arxiv.org/abs/2405.17190 (date of access: 25.05.2025).

19. Practical cybersecurity solution for the energy sector. *MDPI*. URL: https://www.mdpi.com/1996-1073/15/6/2170 (date of access: 25.05.2025).

20. Intel® software guard extensions developer guide. *Intel Corporation*. URL: https://www.intel.com/content/www/us/en/support/articles/000058952/software/intel-security-products.html (date of access: 25.05.2025)

21. Sheller M., Edwards H., Reyes G. Federated Learning for Medical Imaging. *Scientific Reports*. 2023. Vol. 13. P 6–10. URL: https://doi.org/10.1038/s41598-023-39639-1 (date of access: 26.05.2025)

22. Sjouwerman S. Antivirus products are slow at making malware signatures. *KnowBe4*. URL: https://blog.knowbe4.com/antivirus-products-are-slow-at-making-malware-signatures (date of access: 27.05.2025).

23. Teske E. Post solarwinds attack: code-signing best practices. *Cryptomathic*. URL: https://www.cryptomathic.com/blog/the-solarwinds-attack-and-best-practices-for-code-signing (date of access: 27.05.2025).

24. Ransomware activity targeting the healthcare and public health sector. *CISA*. URL: https://www.cisa.gov/news-events/cybersecurity-advisories/aa20-302a (date of access: 27.05.2025).

25. Layered security model. *Identity Management Institute®*. URL: https://identitymanagementinstitute.org/layered-security-model/ (date of access: 28.05.2025).

26. Kreuk F., Kolosnjaji B., Webster G. Novel circuit design for high-impedance and non-local electrical measurements of two-dimensional materials. *arXiv.org*. 2018. URL: https://arxiv.org/abs/1801.10135 (date of access: 28.05.2025).

27. Kunwar P., Aryal K., Gupta M. SoK: leveraging transformers for malware analysis. *arXiv.org*. 2024. URL: https://arxiv.org/abs/2405.17190v2 (date of access: 28.05.2025).

28. Bilot T., El Madhoun N., Al Agha K. A survey on malware detection with graph representation learning. *arXiv.org*. 2023. URL: https://arxiv.org/abs/2303.16004 (date of access: 29.05.2025).

29. Kumar R., Geetha S. Malware classification using XGboost-Gradient Boosted Decision Tree. *Advances in science, technology and engineering systems journal*. 2020. Vol. 5, no. 5. P. 536–549. URL: https://www.astesj.com/publications/ASTESJ_050566.pdf (date of access: 30.05.2025).

30. Malware: types, examples, and how modern anti-malware works. *Perception Point*. URL: https://perception-point.io/guides/malware/malware-types-examples-how-modern-anti-malware-works/ (date of access: 31.05.2025).

31. Python 3.11.12 documentation. *Python Documentation*. URL: https://docs.python.org/3.11/ (date of access: 01.06.2025).

32. Welcome to LIEF's documentation – LIEF Documentation. *LIEF*. URL: https://lief.quarkslab.com/doc/latest/ (date of access: 02.06.2025).

33. GitHub – ytisf/thezoo: A repository of LIVE malwares for your own joy and pleasure. *GitHub*. URL: https://github.com/ytisf/theZoo (date of access: 02.06.2025).

34. Home. *Docker Documentation*. URL: https://docs.docker.com/ (date of access: 03.06.2025).

35. Google colab documentation. *Google Colab*. URL: https://colab.research.google.com/ (date of access: 05.06.2025).