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

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол № __ від __ грудня 2025р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)

«___»_____ 2025 р.

Кваліфікаційна робота

здобувача другого (магістерського) рівня вищої освіти

«Комп'ютерна модель виявлення зловмисного програмного забезпечення на підставі аналізу дампу пам'яті»

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

Виконавець _____ **Євген ЛАНІН**
(підпис)

Науковий керівник _____ **Ніна БАКУМЕНКО**
(підпис)

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до магістерської атестаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і трьох додатків. Загальний обсяг роботи складає 85 сторінки, із яких 63 сторінок основної частини з 13 рисунками, 4 таблицями, 27 найменувань списку використаних джерел та трьома додатками.

Метою кваліфікаційної роботи є підвищення точності класифікації шкідливого програмного забезпечення на основі аналізу дампів оперативної пам'яті з використанням методів машинного навчання.

Об'єкт дослідження – процес виявлення зловмисного програмного забезпечення в оперативній пам'яті комп'ютерних систем на основі аналізу дампів пам'яті за допомогою методів машинного навчання.

Предмет дослідження – методи та алгоритми побудови, навчання і оцінювання моделей машинного та глибокого навчання для класифікації шкідливого програмного забезпечення за даними дампів пам'яті.

Проблема, яка вирішується в кваліфікаційній роботі, полягає в підвищенні ефективності виявлення сучасного шкідливого ПЗ шляхом переходу від традиційних файлових і сигнатурних підходів до аналізу дампів оперативної пам'яті з використанням моделей машинного навчання. Це дає змогу автоматизувати обробку великої кількості пам'яттєвих ознак, зменшити вплив ручного аналізу, підвищити точність класифікації.

Область застосування – системи виявлення та реагування на кіберзагрози, Розроблений підхід може бути інтегрований у програмні комплекси класу IDS/IPS, EDR, а також використаний у дослідницьких і промислових рішеннях з виявлення шкідливого ПЗ.

Ключові слова: шкідливе програмне забезпечення, дамп оперативної пам'яті, машинне навчання, Random Forest, нейронна мережа, CNN.

ABSTRACT

The explanatory note to the master's thesis consists of an introduction, three chapters, conclusions, a list of references, and three appendices. The total volume of the work is 85 pages, of which 63 pages are the main part with 13 figures, 4 tables, 27 references, and three appendices.

The purpose of the qualification work is to improve the accuracy of malware classification based on the analysis of RAM dumps using machine learning methods.

The object of the research is the process of detecting malicious software in the RAM of computer systems based on the analysis of memory dumps using machine learning methods.

The subject of the research is the methods and algorithms for building, training, and evaluating machine learning and deep learning models for classifying malicious software based on memory dump data.

The problem addressed in the thesis is to improve the effectiveness of detecting modern malicious software by moving from traditional file and signature approaches to analyzing RAM dumps using machine learning models. This makes it possible to automate the processing of a large number of memory features, reduce the impact of manual analysis, and improve classification accuracy.

Scope-cyber threat detection and response systems. The developed approach can be integrated into IDS/IPS and EDR software complexes, as well as used in research and industrial solutions for detecting malicious software.

Keywords: malicious software, RAM dump, machine learning, Random Forest, neural network, CNN.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

ROC – характеристична крива (Receiver Operating Characteristic);

AUC – площа під ROC-кривою (Area Under Curve);

API – програмний інтерфейс прикладного програмування (Application Programming Interface);

MLP – багатошаровий перцептрон (Multilayer Perceptron)

CNN – згорткова нейронна мережа (Convolutional Neural Network).

ВСТУП

Стрімкий розвиток інформаційних технологій, поширення хмарних сервісів, віртуалізації та розподілених обчислювальних середовищ супроводжується лавиноподібним зростанням кількості кіберзагроз. Зловмисне програмне забезпечення активно використовує обфускацію, безфайлові техніки, модульні архітектури та легітимні системні інструменти, що значно ускладнює його виявлення традиційними сигнатурними та евристичними засобами. У таких умовах особливого значення набувають підходи, які аналізують реальний стан системи на рівні оперативної пам’яті, де фіксуються фактичні артефакти роботи шкідливого коду.

Актуальність роботи. Аналіз дампів оперативної пам’яті є перспективним напрямом виявлення сучасного шкідливого ПЗ, зокрема обфускованих, безфайлових та гібридних загроз, логіка яких може взагалі не бути представлена у вигляді традиційних виконуваних файлів на диску. Дамп пам’яті відображає запущені процеси, завантажені модулі, мережеві з’єднання, структури ядра, ключі шифрування та інші артефакти, що формують багатовимірний простір ознак, придатний для застосування методів машинного та глибокого навчання. Водночас перетворення сирих дампів на інформативні ознаки, вибір адекватних моделей і забезпечення стійкої багатокласової класифікації типів шкідливого ПЗ залишаються складною науково-практичною задачею. Це зумовлює актуальність розробки комп’ютерної моделі виявлення зловмисного програмного забезпечення на основі аналізу дампів пам’яті.

Метою кваліфікаційної роботи є підвищення точності класифікації шкідливого програмного забезпечення на основі аналізу дампів оперативної пам’яті з використанням методів машинного навчання.

Об’єкт дослідження – процес виявлення зловмисного програмного забезпечення в оперативній пам’яті комп’ютерних систем на основі аналізу дампів пам’яті за допомогою методів машинного навчання.

Предмет дослідження – методи та алгоритми побудови, навчання і оцінювання моделей машинного та глибокого навчання для класифікації шкідливого програмного забезпечення за даними дампів пам’яті.

Методи дослідження: методи системного аналізу та моделювання складних систем; порівняльний експериментальний аналіз; методи машинного навчання; методи глибокого навчання; методи візуалізації та інтерпретації результатів.

Завдання дослідження:

1. Виконати аналіз сучасних типів шкідливого програмного забезпечення, тенденцій їх розвитку та існуючих підходів до виявлення, з особливим акцентом на методах аналізу дампів оперативної пам’яті.
2. Проаналізувати науково-методичний апарат машинного та глибокого навчання, що застосовується для задач класифікації шкідливого ПЗ за технічними ознаками.
3. Розробити та реалізувати побудову моделей класифікації, що включає попередню обробку даних, навчання й налаштування класичних алгоритмів машинного навчання і нейромережевих архітектур.
4. Провести експериментальне дослідження якості запропонованої комп’ютерної моделі за різними метриками.
5. Оцінити практичну придатність розробленої моделі для інтеграції в системи виявлення загроз.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ВИЯВЛЕННЯ ЗЛОВМИСНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Класифікація та характеристики зловмисного програмного забезпечення

Розділ присвячено систематизації основних типів зловмисного програмного забезпечення та аналізу актуальних тенденцій його розвитку, що створюють додаткові виклики для засобів виявлення, зокрема на рівні аналізу пам’яті.

1.1.1 Типи шкідливого ПЗ та їх особливості

У загальному вигляді шкідливе програмне забезпечення розглядається як сукупність програмних компонентів, що навмисно виконують деструктивні, шпигунські або інші небажані дії без поінформованої згоди користувача або власника системи[25]. Основними критеріями класифікації виступають спосіб проникнення, модель розповсюдження, механізми приховування, рівень привілеїв та цілі зловмисника. На практиці для побудови моделей виявлення важливим є не лише формальний клас ПЗ, а й специфічні поведінкові та структурні патерни, які залишаються у файлах, реєстрі, мережевому трафіку та, особливо, в оперативній пам’яті[1].

Класичним типом зловмисного ПЗ є комп’ютерні віруси, що вбудовуються в інші файли або сектори носіїв і розповсюджуються при запуску заражених об’єктів. Для вірусів характерні механізми модифікації коду легітимних програм, використання точок входу та перехоплення системних викликів, що призводить до появи специфічних сигнатур і зміни структури виконуваних файлів. У пам’яті такі загрози часто проявляються через нетипові області коду, ін’єкції в процеси та змінені таблиці імпортів, що робить аналіз дампу ефективним інструментом виявлення.

Черв’яки (worms) відрізняються від вірусів здатністю до самостійного розповсюдження мережею без необхідності взаємодії з користувачем[24]. Вони активно сканують простір IP-адрес, використовують експлойти в протоколах та службах, створюють численні копії себе на скомпрометованих вузлах. Для цього класу ПЗ типовими є аномально велика кількість мережевих з’єднань, масове створення процесів або ниток виконання, що підвищує навантаження на ресурси системи й може бути виявлено як у логах, так і в дампах пам’яті.

Троянські програми (trojans) маскуються під легітимне програмне забезпечення або вбудовуються в нього, надаючи зловмиснику прихований контроль над системою чи забезпечуючи виконання додаткових шкідливих дій[25]. На відміну від вірусів і черв’яків, трояни зазвичай не мають власного механізму саморозповсюдження, однак вирізняються широким спектром функціональних можливостей - від встановлення бекдорів до викрадення конфіденційних даних. У пам’яті такі програми часто помітні завдяки нетиповим мережевим з’єднанням, наявності обфускованих рядків, нестандартних таблиць імпорту бібліотек та підозрілим викликам API[26].

Окремим великим класом є бекдори (backdoors) та інструменти віддаленого керування (RAT - Remote Access Tools), що надають зловмиснику прихований постійний доступ до системи. Вони можуть маскуватися під системні служби, драйвери або невеликі утиліти, запускатися як служби Windows чи демон-процеси в Unix-подібних системах. Типовими ознаками бекдорів у пам’яті є наявність постійно активних з’єднань до фіксованих хостів або доменів, приховані нитки виконання, нестандартні хукапи на системні функції, а також код у незвичних для виконання областях пам’яті[24].

До фінансово мотивованого класу загроз належить шпигунське ПЗ (spyware) та кейлогери. Вони призначені для прихованого збору конфіденційної інформації: натискань клавіш, вмісту буфера обміну, даних форм авторизації, скриншотів, файлів. Для таких програм характерна глибока

інтеграція з інтерфейсами операційної системи, що дозволяє перехоплювати події клавіатури, миші, вікон, а також збереження зашифрованих логів у пам’яті перед передаванням на сервер керування. Аналіз дампу в цьому випадку дає змогу виявити підозрілі буфери, послідовності натискань, структуровані журнали, що не відповідають типовому ПЗ.

Розвиток індустрії кіберзлочинності зробив особливо популярним ще один клас - програм-вимагачів (ransomware), які шифрують дані жертви та вимагають викуп за відновлення доступу[18]. Для них характерні швидке масове відкриття та модифікація файлів, використання криптографічних примітивів, зміна розширень і знищення тіньових копій. У пам’яті такі зразки, як правило, містять ключові криптографічні матеріали, таблиці оброблених файлів, структури для управління чергами шифрування, а також артефакти взаємодії з файловою системою, що може бути використано для класифікації.

Окрему групу становлять руткіти (rootkits) - засоби приховування присутності зловмисного ПЗ на рівні операційної системи. Вони можуть працювати в просторі користувача, на рівні ядра або навіть у прошивках, модифікуючи таблиці системних викликів, драйвери, модулі ядра, структури керування процесами. У контексті дампів пам’яті для руткітів характерні невідповідності між офіційними структурами й фактичним вмістом, приховані модулі та драйвери, а також нестандартні області коду, що не відображаються у звичайних інтерфейсах моніторингу.

Суттєвою категорією для сучасних систем виявлення є ботнети та їх клієнтські модулі. Вони перетворюють скомпрометовані пристрої на вузли розподілених мереж, які зловмисники можуть використовувати для DDoS-атак, розсилання спаму, майнінгу криптовалют або подальшого розповсюдження шкідливого ПЗ. В оперативній пам’яті така активність часто проявляється у вигляді періодичних запитів до командно-керуючої інфраструктури, наявності шифрованих конфігураційних блоків, таймерів для

виконання завдань, а також модулів, що відповідають за оновлення й самозахист[1].

До технічно близького сегменту належить програмне забезпечення для несанкціонованого майнінгу (cryptominers), яке використовує обчислювальні ресурси системи без відома власника. На відміну від класичних вірусів, криптомайнери можуть бути відносно «тихими» у файловій підсистемі, проте створюють стабільно підвищене навантаження на процесор і графічний адаптер. У дампах пам’яті такі програми можуть бути виявлені через наявність специфічних бібліотек, пул-адрес, криптографічних ядер хешування та черг завдань для потоків майнінгу.

Таблиця 1.1

Типові артефакти основних класів шкідливого ПЗ в оперативній пам’яті

Тип загрози	Основний вектор атаки	Ключові артефакти в пам’яті (Memory Artifacts)
Ransomware	Шифрування файлів користувача	Ключі шифрування (AES/RSA), таблиці файлових дескрипторів, висока ентропія в буферах.
Rootkits	Приховування присутності в системі	Модифікація SSDT/IDT, приховані процеси (DKOM), невідповідність між API-списком і VAD-деревом.
Fileless Malware	Використання легітимних інструментів (PowerShell, WMI)	Скрипти та командні рядки в пам’яті інтерпретатора, відсутність файлу на диску, ін’єкції коду.
Trojans / RAT	Віддалене керування, шпигунство	Активні сокети, з’єднання з C2-серверами, хуки (hooks) клавіатури/вікон, інжектовані DLL.
Cryptominers	Несанкціоновані обчислення	Високе навантаження на CPU/GPU, специфічні рядки (адреси пулів, гаманці), повторювані патерни хешування.

З огляду на велику кількість гібридних зразків, що поєднують риси кількох класів (наприклад, троян-бекдор із функціями криптомайнера та руткіт-приховування), традиційні сигнатурні підходи стають дедалі менш ефективними. У результаті особливого значення набуває аналіз поведінки та пам’яті, де фіксуються конкретні послідовності дій, патерни використання

ресурсів і конкретні артефакти, притаманні певним підкласам шкідливого ПЗ. Це створює передумови для застосування машинного та глибокого навчання, здатних автоматично виокремлювати інформативні ознаки в багатовимірних даних дампів[21].

1.1.2 Сучасні тренди розвитку зловмисного ПЗ

Сучасна еволюція зловмисного програмного забезпечення тісно пов’язана з професіоналізацією кіберзлочинних угруповань, появою розвиненої тіньової економіки та поширенням моделі «зловмисне ПЗ як послуга» (MaaS - Malware-as-a-Service). Все більше зразків створюється не окремими ентузіастами, а організованими командами, що дотримуються циклу розробки, тестування й супроводу, аналогічного до легітимних IT-продуктів. Це призводить до швидкого зростання складності загроз, регулярних оновлень, модульної архітектури та підтримки великої кількості цільових платформ.

Одним із ключових трендів є активне застосування обфускації та пакування шкідливого коду[23]. Зловмисники використовують шифрування, поліморфізм, метаморфізм, багаторівневе стиснення й шифрування секцій, динамічне завантаження кодових фрагментів, що значно ускладнює статичний аналіз файлів. Натомість під час виконання в пам’яті розпаковані або розшифровані фрагменти коду, таблиці імпорту, динамічні структури даних усе одно залишають слід, який може бути зафіксовано в дампах і використано для побудови поведінкових та структурних ознак у моделях класифікації.

Ще одним важливим напрямом розвитку є акцент на стійкість до виявлення (evasion) та протидію аналізу. Багато сучасних зразків містять механізми виявлення середовищ песочниць, віртуалізації, налагоджувачів, змін безпечного завантаження, перевіряють специфічні артефакти тестових лабораторій. У разі виявлення таких умов шкідливе ПЗ модифікує свою поведінку, «засинає» або виконує нешкідливий сценарій. На рівні пам’яті це

проявляється умовною активацією окремих гілок коду, наявністю додаткових перевірок, таймерів і нестандартних шаблонів роботи зі системними API.

Посилюється тренд орієнтації на безфайлові (fileless) атаки, коли основна логіка шкідливої діяльності виконується без збереження традиційних виконуваних файлів на диску. Замість цього використовуються легітимні механізми, як-от PowerShell, Windows Management Instrumentation, макроси офісних документів, вбудовані сценарні рушії, а код завантажується безпосередньо в пам’ять. Для класичних антивірусів така активність практично невидима на файловому рівні, проте вміст і структура пам’яті виявляється вкрай інформативним каналом, де фіксуються завантажені скрипти, байткод, змінені структури процесів і нетипові модулі в системних службах[27].

Зловмисники дедалі частіше експлуатують легітимні системні інструменти в межах стратегії «living off the land», коли для проведення атаки задіюються вже наявні в операційній системі або корпоративній інфраструктурі утиліти. Це дає змогу мінімізувати кількість сторонніх виконуваних файлів і зменшити шанси спрацювання сигнатурного аналізу. Однак у пам’яті така активність усе одно залишає низку аномалій - нетипові послідовності викликів, незвичні параметри команд, завантажені бібліотеки, що не відповідають типовому профілю використання системних компонентів.

Окремої уваги заслуговує тренд таргетованих атак і «розумного» вибору жертв. Сучасні кампанії часто орієнтуються на конкретні організації, галузі або навіть окремі інфраструктурні елементи, використовуючи вектор атаки, який важко відтворити в лабораторних умовах. Це стимулює перехід від суто сигнатурних та евристичних механізмів до моделей, здатних навчатися на поведінці ПЗ в конкретному середовищі, аналізувати багатовимірні простори ознак і виявляти відхилення від «норми», зафіксовані в дампах пам’яті.

Активно розвиваються модульні й багатоетапні архітектури шкідливого ПЗ. Перший етап (dropper чи downloader) зазвичай має мінімальний

функціонал і призначений для закріплення в системі, після чого завантажуються додаткові модулі: бекдори, інструменти переміщення мережею, шпигунські блоки, криптомайнери тощо. У пам’яті така структура відображається у вигляді послідовної появи нових модулів, динамічних завантажень бібліотек, нетипових потоків і міжпроцесних взаємодій, що створює додаткові ознаки для машинного аналізу.

Паралельно зростає кількість загроз, орієнтованих на нові платформи та середовища - мобільні операційні системи, пристрої Інтернету речей, віртуалізовані інфраструктури, хмарні сервіси[18]. Такі зразки враховують особливості стеку технологій, систем розгортання й оновлення, а також потоки даних між компонентами. Незважаючи на відмінності, багато з них мають спільну рису: значна частина шкідливої логіки реалізується в оперативній пам’яті, де зберігаються тимчасові ключі, токени доступу, конфігурації й завдання керування, що відкриває простір для уніфікованих підходів до побудови моделей виявлення.

Спостерігається також інтеграція прийомів, пов’язаних із машинним навчанням, безпосередньо в арсенал зловмисників. Деякі сімейства використовують класифікатори та механізми ухвалення рішень для автоматичного вибору вектора атаки, визначення цінних цілей, адаптації до профілю мережевого трафіку або систем захисту. Це створює асиметрію, у якій статичні правила втрачають актуальність, а моделі виявлення повинні вміти протистояти адаптивним загрозам, спираючись на глибший аналіз поведінки та пам’яті[21].

На фоні цих тенденцій помітно посилюється роль аналізу пам’яті як джерела інформативних ознак для класифікації. Дамп оперативної пам’яті відображає реальний стан процесів, завантажених модулів, таблиць імпорту, мережевих сесій, криптографічних структур, що дає змогу виявляти як традиційні, так і безфайлові, модульні, таргетовані загрози. Унаслідок цього саме аналіз дампів стає однією з найбільш перспективних основ для побудови

комп’ютерних моделей виявлення зловмисного ПЗ, орієнтованих на багатокласову класифікацію складних сучасних сімейств.

1.2 Методи аналізу зловмисного програмного забезпечення

Методи аналізу зловмисного програмного забезпечення посідають центральне місце в сучасній кібербезпеці, оскільки саме вони дають змогу зрозуміти поведінку шкідливих програм, їх структуру, цілі та способи обходу захисту. У межах магістерської особлива увага приділяється трьом ключовим підходам - статичному, динамічному та гібридним методам, які доповнюють один одного й створюють основу для подальшого формування ознак у задачі виявлення за дампами пам’яті[19].

1.2.1 Статичний аналіз

Статичний аналіз зловмисного ПЗ передбачає вивчення виконуваних файлів, бібліотек, скриптів та супутніх артефактів без їх фактичного запуску в операційній системі. У найпростішому варіанті цей підхід включає перегляд метаданих файлу, структури заголовків, таблиць імпорту та експорту, вбудованих рядків, ресурсів, конфігураційних блоків, а також обчислення криптографічних хешів для подальшої ідентифікації сімейства.

На більш глибокому рівні статичний аналіз спирається на дизасемблювання та декомпіляцію, які дають змогу відновити логіку роботи програми на рівні машинного коду або близького до вихідного представлення. Аналітики вивчають послідовності інструкцій, виклики API, структуру функцій, реалізацію криптографічних алгоритмів, механізми самозахисту та взаємодії з файловою системою, мережею, реєстром, службами операційної системи.

Серед основних переваг статичного аналізу виділяється відносна безпека: шкідливий код не виконується, отже, ризик компрометації середовища мінімізується. Крім того, цей підхід дає змогу отримати повну картину потенційної функціональності зразка, включно з гілками коду, які можуть ніколи не активуватися в конкретній динамічній сесії, але

залишаються небезпечними. Це особливо важливо для побудови сигнатур та евристик, які потім застосовуються в антивірусних рішеннях. Однак ефективність статичного аналізу серйозно знижується у випадку застосування обфускації, пакування й шифрування[23]. Багато сучасних зразків використовують спеціальні пакувальники, що приховують справжній код за шифрованими або стиснутими секціями, які розпаковуються вже під час виконання в пам’яті. У таких випадках аналітик змушений або виконувати попередню ручну розпаковку, або переходити до динамічних технік, які спостерігають за програмою в реальному часі.

Статичний аналіз напряму пов’язаний з побудовою ознак для моделей машинного навчання: з коду та структури файлів можуть виділятися n-грамми байтів, частотні характеристики опкодів, графи викликів функцій, статистика рядків та імпортованих бібліотек. Утім, фокус магістерської зміщується в бік аналізу пам’яті, де ці статичні властивості частково проявляються вже в розпакованому вигляді, утворюючи багатший простір ознак.

1.2.2 Динамічний аналіз

Динамічний аналіз передбачає спостереження за виконанням зловмисного ПЗ в контрольованому середовищі, наприклад у віртуальній машині, песочниці або ізольованому стенді. Програма запускається, після чого фіксується її активність: створення та модифікація файлів, зміни в реєстрі, мережеві з’єднання, робота зі службами, ін’єкції в інші процеси, завантаження драйверів тощо. Такий підхід дозволяє побачити реальну поведінку зразка, включно з ланцюжком дій після початкового запуску. Динамічний аналіз умовно поділяється на високорівневий (behavioral) та низькорівневий. У першому випадку фіксуються узагальнені події: які файли відкриваються, які ключі реєстру змінюються, до яких адрес встановлюються з’єднання, які процеси створюються. У другому випадку відстежуються системні виклики, стеки викликів, зміни в окремих ділянках пам’яті, що дає значно детальніше уявлення про внутрішню логіку програми[19].

Однією з ключових переваг динамічного аналізу є здатність виявляти реальну шкідливу поведінку, навіть якщо вихідний код сильно обфускований або запакований. Після розпаковки в оперативній пам’яті шкідливі модулі все одно виконуються й здійснюють дії, які важко повністю приховати: встановлюють мережеві з’єднання, читають конфіденційні файли, змінюють налаштування системи, створюють бекдори. У результаті навіть дуже заплутаний код залишає поведінкові «сигнали», що фіксуються системами моніторингу. Водночас динамічний підхід має низку обмежень. По-перше, для отримання повної картини необхідно, щоб шкідливий код справді активував усі або більшість своїх функцій, що не завжди відбувається під час короткого тестового запуску. По-друге, сучасне ПЗ часто містить механізми виявлення віртуалізації, песочниці чи відлагоджувача й у разі їх виявлення змінює поведінку, відкладає шкідливі дії або працює в «безпечному» режимі. Усе це ускладнює збір надійних даних, особливо для автоматизованих систем.

Цей метод аналізу цікавий тим, що саме під час виконання формується стан оперативної пам’яті, який фіксується в дампі. У пам’яті опиняються розпаковані модулі, тимчасові структури даних, ключі шифрування, конфігурації, черги завдань, контексти мережевих сесій. Це робить дамп пам’яті своєрідним «знімком» динамічної поведінки, який можна піддати подальшому автоматизованому аналізу методами машинного навчання, не виконуючи шкідливий код повторно[26].

Практичні інструменти динамічного аналізу часто поєднують логування подій та автоматичну генерацію звітів, що описують поведінку зразка у вигляді набору ознак. Ці ознаки включають кількість створених процесів, типи використаних API, параметри мережевих з’єднань, зміни в реєстрі, особливості взаємодії з файлами та службами. Частина цієї інформації прямо або опосередковано відображається в структурі дампу, що знову ж таки підкреслює зв’язок між динамічним аналізом і пам’яттєвим підходом.

1.2.3 Гібридні підходи

Гібридні методи аналізу зловмисного ПЗ поєднують статичні й динамічні підходи, намагаючись використати їхні сильні сторони та компенсувати слабкі21]. Ідея полягає в тому, що частина інформації про загрозу може бути отримана без запуску - наприклад, сигнатури, структурні особливості, граф викликів, - а інша частина вимагає спостереження за реальною поведінкою в системі. Комбінування цих рівнів дозволяє отримати більш повний набір ознак для класифікації та покращити стійкість до обфускації та протидії аналізу.

Один із поширених варіантів гібридного підходу полягає в попередньому статичному скринінгу зразків із подальшим поглибленим динамічним аналізом лише для підозрілих об’єктів. Це дає змогу зменшити навантаження на песочниці й віртуальні машини, водночас не ігноруючи потенційно небезпечні файли, які не збігаються зі відомими сигнатурами. У рамках такого pipeline статичні ознаки можуть використовуватися для початкової класифікації, а результати динамічного моніторингу - для уточнення й підтвердження висновків. Інший напрям розвитку гібридних методів пов’язаний із глибокою інтеграцією аналізу пам’яті в загальний процес. У цьому випадку статичний аналіз застосовується до вихідних файлів, динамічний - до процесу виконання, а дампи оперативної пам’яті виступають проміжним або кінцевим артефактом, у якому поєднуються результати роботи шкідливих модулів. Це дає можливість виявляти навіть ті фрагменти коду й даних, які не представлені у вихідному файлі в чистому вигляді, але з’являються в пам’яті після розпаковки або завантаження з мережі[21].

Гібридні системи зазвичай використовують складні моделі прийняття рішень, які враховують одразу кілька груп ознак: структуру файлу, поведінкові логі, пам’яттєві артефакти, контекст запуску, часові характеристики. Поширеною практикою є побудова ансамблів моделей, де окремі класифікатори спеціалізуються на різних типах ознак, а остаточне рішення ухвалюється на основі їхньої агрегованої оцінки. Такий підхід

підвищує точність і стійкість системи до нових, раніше невідомих сімейств шкідливого ПЗ.

Особливий інтерес становлять гібридні підходи, що використовують дампи пам’яті як один із центральних джерел даних. Після динамічного виконання або емульованого запуску програм формується дамп, у якому присутні як сліди статичної структури (розпаковані модулі, таблиці імпортів), так і результати поведінки (буфери мережевих протоколів, ключі шифрування, журнали дій). Це створює багатий простір ознак, придатний для побудови моделей багатокласової класифікації, що здатні розрізняти не лише наявність шкідливого ПЗ, а й конкретні типи або сімейства. У результаті поєднання статичного, динамічного та аналізу пам’яті формує цілісну методологічну основу, на якій ґрунтується подальша розробка комп’ютерної моделі виявлення зловмисного програмного забезпечення. Такий підхід підсилює стійкість системи до обфускації, безфайлових технік, модульних архітектур і таргетованих атак, що особливо важливо в умовах сучасного ландшафту кіберзагроз[27].

1.3 Аналіз дампів пам’яті як метод виявлення шкідливого ПЗ

Аналіз дампів пам’яті розглядається як процес вивчення знімків оперативної пам’яті, що відображають запущені процеси, завантажені модулі, мережеві з’єднання, структури ядра та користувацькі дані на конкретний момент часу. На відміну від файлового аналізу, такий підхід працює безпосередньо з результатом виконання програм, зокрема з розпакованим та розшифрованим шкідливим кодом, який може взагалі не бути присутнім на диску у чистому вигляді.

Пам’яттєвий аналіз особливо актуальний в умовах поширення безфайлових атак, обфускованих зразків та модульних шкідливих комплексів, де ключова логіка існує тільки в оперативній пам’яті. Захоплення дампу дозволяє зберегти повну картину подій навіть після завершення інциденту й проводити ретельний розбір без ризику впливу на живу систему, що важливо

для цифрової криміналістики та побудови наборів даних для машинного навчання.

1.3.1 Структура та особливості дампів пам’яті

Дамп пам’яті зазвичай являє собою бінарний файл, у якому збережено вміст фізичної або віртуальної оперативної пам’яті системи на момент зняття «знімка»[6]. Усередині такого образу містяться області, що відповідають просторам адрес різних процесів, сегментам коду й даних, стеку, купі, драйверам ядра, кешам, таблицям сторінок, а також допоміжним структурам операційної системи. На практиці виділяють кілька типів дампів: повні образи фізичної пам’яті, вибіркові дампи окремих процесів, дампи ядра або спеціальні «selective dumps», які зберігають лише релевантні ділянки для подальшого аналізу. Повний дамп забезпечує максимум інформації, але є об’ємним і складним для обробки, тоді як вибіркові варіанти дають змогу сфокусуватися на підозрілих процесах або контейнерах, зменшуючи обсяг даних і час обробки.

Особливістю структури пам’яті є відсутність чітких «кордонів» між окремими об’єктами з точки зору зовнішнього спостерігача: інформація зберігається у вигляді байтових послідовностей, інтерпретація яких залежить від контексту та знань про формат ОС. Тому для ефективного аналізу застосовуються спеціалізовані фреймворки, що вміють відтворювати логічні структури на кшталт списків процесів, таблиць модулів, дескрипторів відкритих файлів і мережевих сесій[26]. Ще одна важлива риса дампів пам’яті полягає в тому, що вони можуть містити як живі, так і «застарілі» фрагменти даних, які вже не належать активним структурам, але не були перезаписані. Це дозволяє відновлювати певні артефакти, навіть якщо процес уже завершив роботу, проте водночас ускладнює автоматизований аналіз через наявність «шуму» та частково пошкоджених структур.

1.3.2 Ознаки та артефакти зловмисної активності в пам’яті

У дампах пам’яті можуть бути виявлені численні артефакти, що свідчать про присутність і активність шкідливого ПЗ. До базових ознак належать аномальні процеси, приховані або ін’єктовані модулі, підозрілі DLL, незвичні таблиці імпорту, невідповідності між списками процесів на різних рівнях (user-mode vs kernel-mode)[26].

Важливим джерелом інформації виступає мережева активність, відображена в пам’яті: відкриті сокети, TCP/UDP-з’єднання, таблиці маршрутизації, буфери протоколів. Наявність підозрілих з’єднань до відомих C2-серверів, нетипових портів, великої кількості короткоживучих сесій або шифрованих каналів керування може вказувати на роботу ботнет-клієнтів, бекдорів чи шпигунського ПЗ. Серед інших характерних артефактів виділяються сліди ін’єкцій коду: додаткові секції у пам’яті процесів, сторінки з правами виконання, не пов’язані з жодним легітимним модулем, hook’и на критичні API-функції, модифіковані таблиці системних викликів. Для руткітів та безфайлових загроз типовими є приховані драйвери ядра, модулі, не відображені в стандартних структурах ОС, та нестандартні ланцюжки викликів.Окремий клас ознак пов’язаний із збереженими в пам’яті конфігураціями, ключами шифрування, токенами автентифікації, журналами дій або чергами завдань шкідливого ПЗ. Такі структури можуть містити адреси командних серверів, правила фільтрації цілей, параметри шифрування файлів (у випадку ransomware), що дозволяє не лише виявляти загрозу, а й класифікувати її за типами або сімействами.

Для задач машинного та глибокого навчання на основі дампів пам’яті застосовуються різні способи перетворення сирих бінарних даних на ознаки: від прямого аналізу байтових послідовностей і статистики байтів до побудови візуальних представлень у вигляді зображень. У низці робіт дампи конвертуються в відтінкові зображення, з яких потім витягуються дескриптори HOG або застосовуються згорткові нейромережі, що досягають високих показників точності класифікації[9].

1.3.3 Інструменти для аналізу дампів пам’яті

Ключовим інструментом пам’яттєвої форензіки є Volatility Framework, відкритий фреймворк, який став де-факто стандартом для аналізу дампів оперативної пам’яті. Він підтримує різні версії Windows, Linux, macOS, вміє відновлювати списки процесів, модулів, мережевих з’єднань, дескрипторів файлів, реєстру, а також виявляти потенційно приховані або ін’єктовані компоненти[26].

Volatility надає набір плагінів, що автоматизують типові задачі: пошук підозрілих процесів, виявлення аномальних DLL, аналіз мережевих артефактів, дамп окремих процесів для подальшого статичного аналізу, пошук рядків, пов’язаних із шкідливою активністю. Серед поширених можливостей - команди для сканування на наявність прихованих процесів, порівняння списків модулів, пошук сигнатур PE-файлів у сирому просторі пам’яті, що дозволяє виявляти ін’єкції. На базі Volatility створюються розширення й надбудови, зокрема графічні інтерфейси й інтегровані форензичні середовища, які спрощують роботу аналітиків і частково автоматизують виявлення шкідливих артефактів. Паралельно розвиваються інші інструменти для пам’яттєвої форензіки, а також спеціалізовані програмні рішення, що підтримують селективні дампи, інтеграцію з хмарною інфраструктурою й автоматичну генерацію ознак для моделей машинного навчання[6].

Також інтерес становлять інструменти, які дозволяють не лише інспектувати дампи вручну, а й програмно витягувати структуровані ознаки: списки процесів, мережеві сесії, інформацію про модулі, статистику пам’яті. На основі таких даних формуються набори для навчання класифікаторів, здатних розпізнавати присутність і типи шкідливого ПЗ.

1.4 Огляд існуючих рішень для виявлення шкідливого ПЗ

Сучасні системи виявлення шкідливого програмного забезпечення охоплюють широкий спектр підходів - від класичних сигнатурних антивірусів до поведінкових та машинно-навчальних платформ. Традиційні продукти

базуються на базах сигнатур, евристичних правилах і моделюванні шаблонів поведінки, проте їхні можливості обмежуються швидкістю оновлення та складністю сучасних методів ухилення.На цьому тлі активно розвиваються рішення, що використовують машинне й глибоке навчання для аналізу файлів, мережевого трафіку та пам’яті. Частина робіт сфокусована саме на дампах пам’яті: пропонуються методики перетворення дампів у візуальні представлення, витягнення поведінкових артефактів, побудова класифікаторів для розмежування «benign» та шкідливих зразків, а також для класифікації за сімействами[1].

Наприклад, описуються підходи, де дампи конвертуються у зображення, з яких нейронні мережі (зокрема CNN) виділяють патерни, притаманні різним типам загроз[9]. В інших роботах пропонується збір ознак на основі реєстрової активності, імпортованих бібліотек та викликів API, які витягуються з дампів і подаються на вхід класичним алгоритмам машинного навчання (SVM, Random Forest, MLP), демонструючи високу точність виявлення шкідливого ПЗ.

Хоча значна частина промислових рішень орієнтується переважно на файловий і мережевий рівні, у наукових публікаціях дедалі частіше підкреслюється потенціал пам’яттєвого аналізу для виявлення як відомих, так і нових загроз. Наявні роботи демонструють можливість досягнення високих показників точності, проте зазначають складність формування репрезентативних наборів дампів, високу розмірність даних і потребу в ефективній інженерії ознак та адаптивних моделях класифікації[21].

Висновки за розділом 1

У першому розділі було розглянуто теоретичні основи виявлення зловмисного програмного забезпечення, починаючи з класифікації основних типів шкідливого ПЗ, їхніх характеристик та сучасних тенденцій розвитку. Показано, що еволюція загроз пов’язана з обфускацією, безфайловими атаками, модульними архітектурами та активним використанням легітимних

системних інструментів, що суттєво ускладнює виявлення традиційними сигнатурними засобами.

Було проаналізовано ключові методи вивчення шкідливого ПЗ - статичний, динамічний і гібридні підходи, окреслено їхні сильні сторони та обмеження, а також показано, що жоден із них окремо не забезпечує повного охоплення всього спектра сучасних загроз. На цьому фоні аналіз дампів оперативної пам’яті розглянуто як перспективний напрям, що поєднує переваги поведінкового та структурного підходів, дозволяючи працювати з реально виконуваним кодом і його артефактами. Структура дампів пам’яті, типові ознаки й артефакти зловмисної активності, а також наявні інструменти пам’яттєвої форензіки створюють базу для формування інформативних ознак, придатних до подальшої автоматизованої обробки. Огляд існуючих рішень показав, що застосування методів машинного та глибокого навчання до даних пам’яті здатне забезпечити високу точність виявлення та класифікації шкідливого ПЗ, але водночас потребує ретельного проєктування наборів даних і моделей.

РОЗДІЛ 2
МЕТОДИ МАШИННОГО ТА ГЛИБОКОГО НАВЧАННЯ ДЛЯ ВИЯВЛЕННЯ ЗЛОВМИСНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Основи машинного навчання в контексті кібербезпеки

У сфері кібербезпеки машинне навчання розглядається як інструмент автоматичного виявлення закономірностей у великих обсягах технічних даних: мережевих трас, логах подій, файлових образах, дампах пам’яті. Замість ручного створення жорстких сигнатур і правил будується модель, яка навчається на прикладах відомих «чистих» і шкідливих об’єктів і згодом здатна узагальнювати знання на нові, раніше невідомі зразки. Базовий pipeline включає кілька етапів: відбір даних, попередню обробку та нормалізацію, відбір інформативних ознак, вибір і навчання моделі, оцінювання якості та подальше розгортання в реальному середовищі. У задачах виявлення шкідливого ПЗ даними можуть виступати статичні характеристики файлів, поведінкові логи, артефакти з пам’яті; особливість полягає в сильній незбалансованості класів, швидкій еволюції загроз і високій ціні помилок[22].

Для пам’яттєвого аналізу машинне навчання дає змогу працювати з багатовимірними описами дампів, де людина не в змозі інтуїтивно оцінити значущість кожної ознаки. У цьому випадку моделі не лише автоматизують класифікацію, а й дозволяють виявляти нетривіальні комбінації параметрів, що відповідають різним типам шкідливої активності.

2.1.1 Підходи машинного навчання

У контексті виявлення зловмисного ПЗ найбільше значення мають підходи навчання з учителем, навчання без учителя та напівконтрольовані методи. Навчання з учителем передбачає наявність розміченого набору даних, де кожному об’єкту (наприклад, дампу пам’яті або витягнутому вектору ознак) поставлено у відповідність мітку класу: «benign», «malicious» або конкретний тип шкідливого ПЗ[22]. Загальну схему процесу навчання на розмічених даних

наведено на рисунку 2.1. Для таких задач застосовуються класифікаційні алгоритми: логістична регресія, дерева рішень, випадкові ліси, метод опорних векторів, градієнтний бустинг, багатошарові перцептрони та інші моделі, здатні працювати з числовими та категоріальними ознаками. Перевага підходу полягає в тому, що модель безпосередньо оптимізується під розділення заданих класів, а якість можна чітко виміряти за допомогою стандартних метрик на тестовій вибірці.

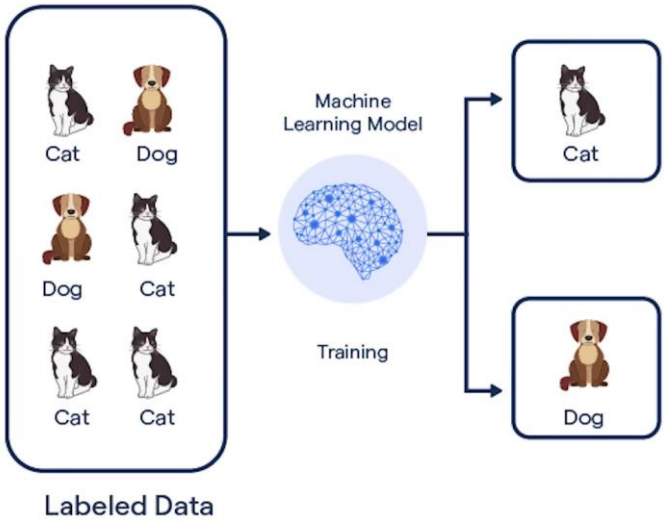

Рисунок 2.1 – Загальна схема процесу навчання з вчителем (Supervised Learning) на розмічених даних.

Навчання без учителя застосовується тоді, коли позначені дані відсутні або їх недостатньо. У такому випадку основна увага приділяється пошуку аномалій і кластеризації: алгоритми групують об’єкти за подібністю, виділяючи кластери «типової» поведінки, а відхилення від цих кластерів розглядаються як потенційно шкідливі[2]. Принцип роботи алгоритмів з нерозміченими вхідними даними ілюструє рисунок 2.2. Для дампів пам’яті подібні методи можуть виявляти нетипові профілі процесів або структур пам’яті, характерні для нових сімейств ПЗ.

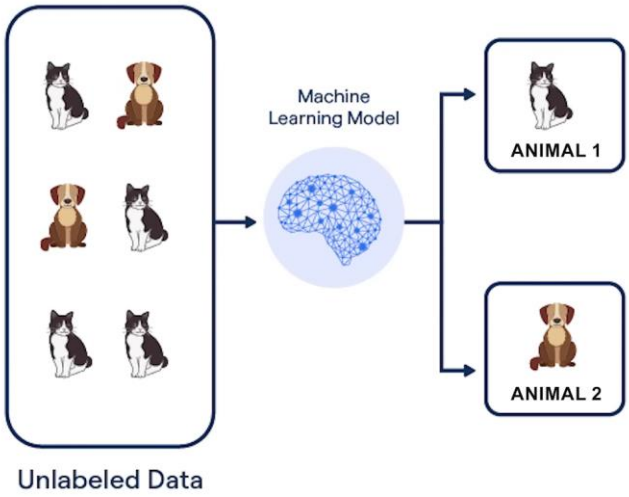

Рисунок 2.2 – Розподіл трафіку за змінною Label

Напівконтрольовані підходи поєднують ідеї обох напрямів, використовуючи невелику кількість розмічених даних разом із великою масою нерозмічених. Це актуально для кібербезпеки, де повна розмітка великого масиву дампів є ресурсомістким завданням, а «чисті» зразки значно переважають шкідливі. Такі схеми дозволяють розширювати знання моделі про простір ознак, одночасно зберігаючи орієнтацію на класи, визначені експертами[18].

2.1.2 Постановка задачі бінарної та багатокласової класифікації

У задачах виявлення шкідливого ПЗ на основі аналізу дампів пам’яті природно виділяються два основні формати постановки задачі: бінарна класифікація (шкідливе / легітимне) та багатокласова класифікація (класи або сімейства загроз). У бінарному випадку метою є визначення факту наявності зловмисної активності, тоді як у багатокласовому - уточнення типу шкідливого ПЗ, наприклад «ransomware», «trojan», «spyware», «botnet» тощо.

У бінарній постановці кожному об’єкту (наприклад, вектору ознак, отриманих з дампу пам’яті) ставиться мітка

$$y \in \{0,1\}, \tag{2.1}$$

де 0 може відповідати «benign», а 1 - «malicious».

Модель будується як відображення $f : X \to \{0,1\}$,, яке наближається на основі навчальної вибірки $\{(x_i, y_i)\}^N$, і має узагальнюватися на нові об'єкти. Основними ризиками виступають дисбаланс класів (значно більше «чистих» дампів) та різноманітність шкідливих зразків, що можуть відрізнятися від тих, на яких модель навчалася[4].

У багатокласовій постановці кожному об'єкту ставиться мітка

$$y \in \{1,2, \dots ,K\}, \tag{2.2}$$

де K - кількість класів (типів або сімейств ПЗ).

Відповідно, модель $f : X \to \{1, \dots ,K\}$повинна не просто відокремлювати шкідливе ПЗ від легітимного, а й розрізняти між собою різні групи загроз на основі їхніх пам'яттєвих ознак. Для таких задач часто застосовуються стратегії «one-vs-rest», «one-vs-one» або безпосереднє багатокласове навчання, інтегроване в алгоритм (наприклад, багатокласова логістична регресія, softmax-виходи в нейронних мережах).

Багатокласова класифікація є складнішою через накладання кількох факторів: частина класів може бути слабо представлена в даних, деякі сімейства мають подібні поведінкові патерни, а межі між класами часто розмиті через гібридний характер сучасного шкідливого ПЗ. Унаслідок цього метрики оцінювання виходять за межі простої «загальної точності» й включають аналіз матриці помилок, обчислення точності та повноти для кожного класу, а також зважені усереднення, що враховують дисбаланс[4].

У контексті магістерської роботи бінарна класифікація розглядається як базовий сценарій, який демонструє здатність моделей відрізняти «чисті» дампи від тих, що містять ознаки шкідливої активності. Проте основний акцент робиться саме на багатокласовій постановці, оскільки вона краще відображає реальні потреби кібербезпеки: важливо не лише констатувати факт компрометації, а й розуміти тип загрози для коректного реагування та пріоритезації заходів. При переході від бінарної до багатокласової класифікації змінюються як вимоги до набору даних (необхідна наявність

достатньої кількості прикладів для кожного класу), так і до самої моделі (потрібна висока роздільна здатність у просторі ознак). Це зумовлює потребу в більш ретельній інженерії ознак із дампів пам’яті, використанні потужніших алгоритмів та продуманих схем оцінювання, що й буде розглянуто в наступних підрозділах розділу.

2.2 Класичні методи навчання з вчителем

2.2.1 Алгоритми класифікації

Метод опорних векторів (Support Vector Machine, SVM) будує оптимальну гіперплощину, яка максимізує відступ - відстань між межею прийняття рішень та найближчими зразками різних класів[11].Для лінійно роздільних даних SVM знаходить площину

$$w^T x + b = 0, \tag{2.3}$$

де - вектор нормалі, - зміщення.

Геометричну інтерпретацію побудови гіперплощини та максимізації відступу (margin) зображено на рисунку 2.3.

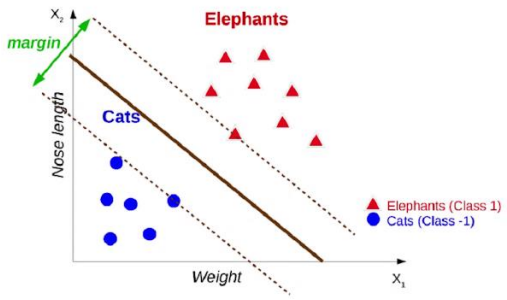

Рисунок 2.3 – Лінійне розділення класів алгоритмом SVM з виділенням опорних векторів.

Відступ визначається як , тому задача оптимізації формулюється як мінімізація $||w||^2$ за умови правильної класифікації всіх тренувальних зразків. Для нелінійно роздільних даних SVM застосовує ядровий трюк, відображаючи вхідні дані в простір вищої розмірності, де вони стають лінійно роздільними. Популярні ядра (kernels) включають радіальну базисну функцію, яка описується формулою:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2), \quad (2.4)$$

та поліноміальне ядро, сигмоїдне ядро. RBF-ядро демонструє значно кращу продуктивність порівняно з лінійним для складних патернів мережевого трафіку. SVM ефективний для задач з високою розмірністю простору ознак, що типово для виявлення шкідливого ПЗ. Дослідження демонструють, що SVM досягає високої точності при класифікації шкідливого ПЗ на основі ознак з реєстру, бібліотек імпорту та викликів API. Однак SVM вимагає ретельного підбору ядра та гіперпараметрів (C, gamma), що може бути обчислювально затратним[1].

Метод k-найближчих сусідів (K-Nearest Neighbors, KNN) відносить новий зразок до класу, найбільш представленого серед його найближчих сусідів в просторі ознак[11]. Відстань зазвичай вимірюється метрикою Евкліда, Манхеттена або Мінковського. Алгоритм не має фази навчання в традиційному розумінні - весь тренувальний набір даних зберігається в пам'яті, а класифікація відбувається під час виведення. Візуалізацію процесу визначення класу нового об’єкта на основі його сусідів подано на рисунку 2.4. KNN простий у реалізації та інтуїтивно зрозумілий. Однак алгоритм обчислювально неефективний для великих наборів даних, оскільки вимагає розрахунку відстані до всіх тренувальних зразків для кожного нового об'єкта. Чутливість до незбалансованих наборів даних та нерелевантних ознак також обмежує застосовність KNN.

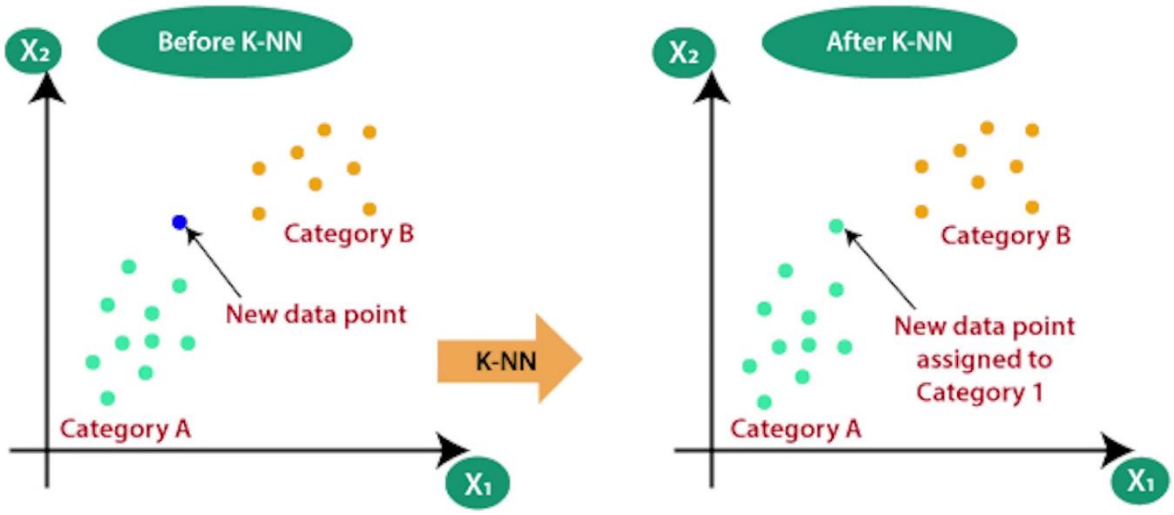

Рисунок 2.4 – Принцип класифікації нового об’єкта методом k-найближчих сусідів (k-NN).

Дерево рішень (Decision Tree) будує деревоподібну структуру рішень, де кожен внутрішній вузол представляє тест на значення ознаки, гілки - можливі результати тесту, а листкові вузли - класові мітки[Bishop C. M.]. Ієрархічну структуру прийняття рішень, що складається з кореневого вузла та розгалужень, наведено на рисунку 2.5. Алгоритм рекурсивно розбиває простір ознак на регіони, максимізуючи приріст інформації або мінімізуючи нечистоту Джині на кожному кроці. Нечистота Джині для вузла визначається як

$$G = 1 - \sum_{i=1}^{k} p_i^2, \tag{2.5}$$

де p_i - частка зразків класу у вузлі.

Дерево рішень вибирає таке розбиття, яке максимально зменшує зважену суму нечистоти Джині дочірніх вузлів. Дерева рішень легко інтерпретуються та візуалізуються, що важливо для розуміння логіки класифікації шкідливого ПЗ. Дослідження показують досягнення 100% точності на деяких наборах даних шкідливого ПЗ[16]. Проте окремі дерева схильні до перенавчання, особливо при великій глибині.

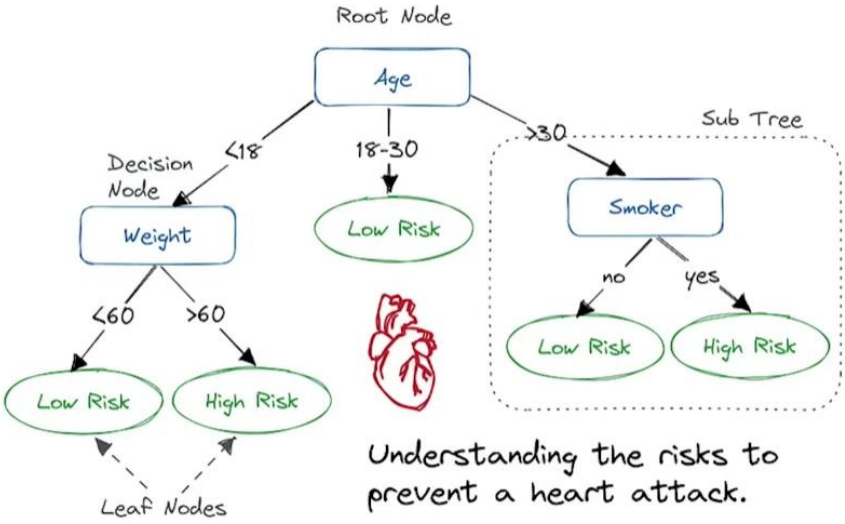

Рисунок 2.5 – Структура та принцип прийняття рішень у алгоритмі Decision Tree.

Логістична регресія (Logistic Regression) моделює ймовірність належності зразка до класу через логістичну функцію

$$\sigma(z) = 1/(1 + e^{-z}), \tag{2.6}$$

де $z = w^T x + b$.

Графік логістичної функції, що демонструє S-подібну залежність ймовірності від значення аргументу, представлено на рисунку 2.6.

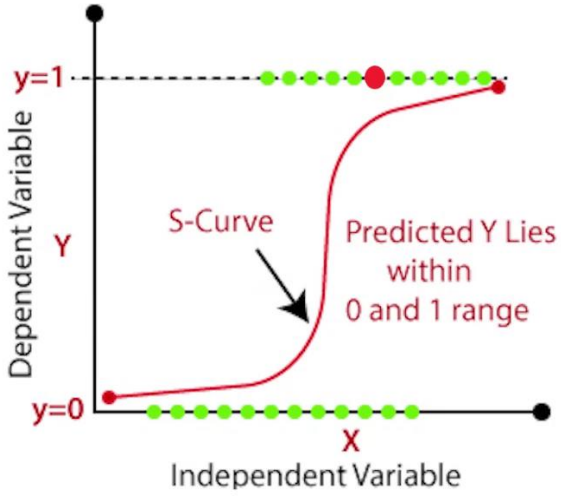

Рисунок 2.6 – Графічна інтерпретація логістичної функції (сигмоїди) у задачі бінарної класифікації.

Для багатокласової класифікації використовується функція софтмакс, яка описується формулою:

$$\text{softmax}(z_i) = e^{z}i/\sum^{k} e^{z}j, \quad (2.7)$$

Логістична регресія швидка в навчанні та виведенні, добре працює на лінійно роздільних даних[11]. Алгоритм використовувався в порівняльних дослідженнях виявлення шкідливого ПЗ, хоча демонструє нижчу точність порівняно з ансамблевими методами.

Наївний Байєс (Naïve Bayes) базується на теоремі Байєса з припущенням про умовну незалежність ознак. Ймовірність класу для зразка з ознаками $x_1, \ldots, x_n$ обчислюється як:

$$P(c|x_1, \ldots, x_n) \propto P(c)\prod^{n}P(x_i|c), \quad (2.8)$$

Незважаючи на нереалістичне припущення про незалежність, наївний Байєс часто показує конкурентну продуктивність та надзвичайно швидкий.

2.2.2 Ансамблеві методи

Ансамблеві методи комбінують передбачення множини базових моделей для досягнення вищої точності та робастності порівняно з окремими моделями[1]. Три основних підходи - беггінг, бустинг та стекінг.

Беггінг навчає множину моделей паралельно на різних бутстреп-вибірках тренувального набору даних. Бутстреп-вибірка створюється шляхом випадкового вибору зразків з поверненням, що призводить до різноманітності навчальних наборів для кожної моделі. Фінальне передбачення формується через голосування для класифікації або усереднення для регресії. Схематично процес паралельного навчання незалежних моделей у методі беггінг показано на рисунку 2.7. Беггінг зменшує дисперсію та запобігає перенавчанню через усереднення помилок окремих моделей. Метод особливо ефективний для нестабільних алгоритмів, таких як дерева рішень, які чутливі до невеликих змін у тренувальних даних.

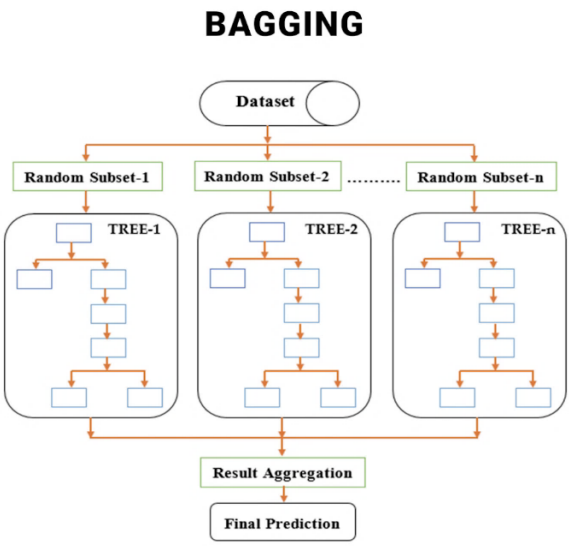

Рисунок 2.7 – Концептуальна схема роботи ансамблевого методу Bagging (Bootstrap Aggregating).

Випадковий ліс є найпопулярнішою реалізацією беггінгу на основі дерев рішень. Порівняння архітектури одиночного дерева з ансамблевими підходами (випадковий ліс та градієнтний бустинг) унаочнює рисунок 2.8. Окрім бутстреп-вибірки даних, випадковий ліс також застосовує випадковий підбір підмножини ознак для кожного розбиття вузла дерева. Це додатково декорелює дерева, покращуючи узагальнюючу здатність ансамблю. Випадковий ліс може обробляти як дискретні, так і неперервні ознаки[22]. Алгоритм робастний до шуму в даних та забезпечує природну кількісну оцінку невизначеності через розподіл голосів між деревами. Дослідження з виявлення шкідливого ПЗ демонструють, що випадковий ліс досягає однієї з найвищищ точностей серед класичних методів[16]. Використання випадкового лісу з відбором ознак забезпечує найкращі результати для класифікації шкідливого ПЗ. Випадковий ліс підтримує паралельні обчислення, оскільки дерева навчаються незалежно. Параметр n_jobs=-1 активує використання всіх ядер процесора, значно прискорюючи навчання. Типова кількість оцінювачів коливається від 100 до 500.

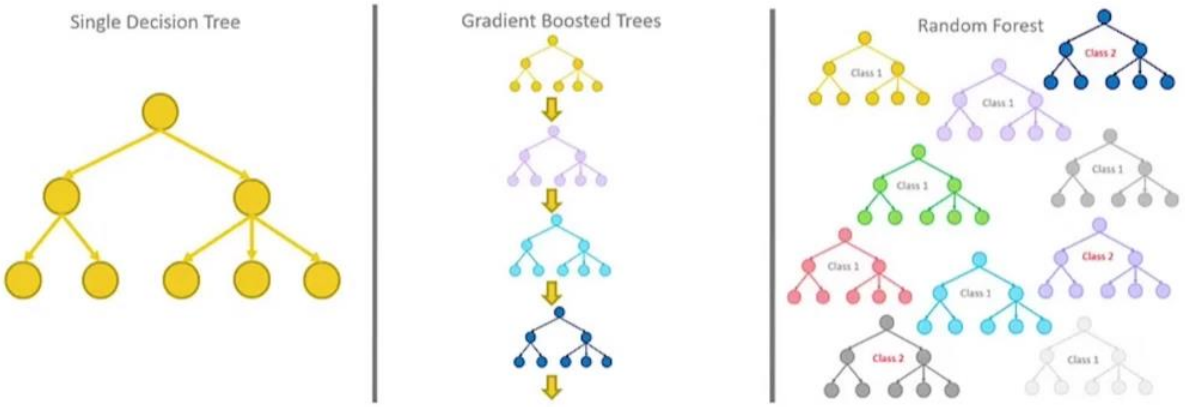

Рисунок 2.8 – Порівняння архітектур: одиночне дерево рішень, градієнтний бустінг та випадковий ліс (Random Forest).

Бустинг навчає моделі послідовно, де кожна наступна модель фокусується на корекції помилок попередніх. На відміну від беггінгу, де моделі незалежні, бустинг створює ланцюг взаємозалежних слабких навчальників. Ключову структурну відмінність між паралельним (Bagging) та послідовним (Boosting) формуванням ансамблів зображено на рисунку 2.9.

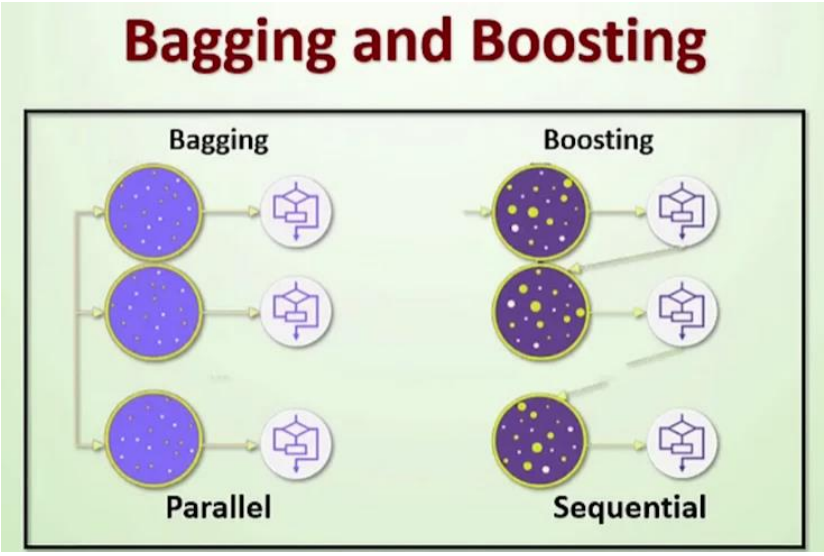

Рисунок 2.9 – Порівняння стратегій навчання ансамблевих методів: Bagging проти Boosting.

Градієнтний бустинг (Gradient Boosting) мінімізує функцію втрат через побудову адитивної моделі

$$F(x) = \sum^{M} \gamma_m h_m(x), \tag{2.9}$$

де h_m - слабкі навчальники, γ_m- їх ваги.

На кожній ітерації новий навчальник навчається на градієнті функції втрат відносно передбачень поточного ансамблю.

XGBoost (Extreme Gradient Boosting) є оптимізованою реалізацією градієнтного бустингу. XGBoost включає регуляризацію (L1 та L2) для запобігання перенавчанню, підтримує паралельну побудову дерев та ефективно обробляє пропущені значення. Алгоритм використовує другі похідні функції втрат (гесіан, Hessian) для точніших оновлень. XGBoost демонструє виняткову продуктивність на структурованих даних та широко застосовується в змагальному машинному навчанні. Швидкість навчання та ефективність роблять його популярним вибором для виявлення шкідливого ПЗ[17].

LightGBM - альтернативна реалізація градієнтного бустингу з акцентом на швидкість. Алгоритм використовує гістограмні методи та листове зростання дерев замість порівневого. Дослідження класифікації шкідливого ПЗ показують, що LightGBM досягає найвищої точності та точності позитивного класу серед XGBoost та логістичної регресії[17].

2.3 Нейронні мережі та глибоке навчання

2.3.1 Основи нейронних мереж

Штучні нейронні мережі інспіровані структурою біологічного мозку та складаються з взаємопов'язаних обчислювальних вузлів (нейронів), організованих у шари[13]. Кожен нейрон отримує зважену суму входів, застосовує функцію активації та передає результат наступному шару. Архітектура нейронної мережі включає вхідний шар, що приймає вектор ознак; один або більше прихованих шарів, які виконують нелінійні трансформації; та вихідний шар, що генерує передбачення. Для багатокласової класифікації вихідний шар містить нейронів (по одному на клас) з активацією софтмакс.

Обчислення в нейроні формально описується як:

$$z = \sum^{n} w_i x_i + b, \tag{2.10}$$

де x_i - входи, w_i - ваги, b - зміщення (bias). Активація нейрона визначається як $a = \sigma(z)$, де σ - функція активації.

Функції активації вносять нелінійність, дозволяючи мережі моделювати складні взаємозв'язки. Гіперболічний тангенс $\tanh(z) = (e^z - e^{-z})/(e^z + e^{-z})$ має діапазон (-1, 1) та центрований навколо нуля, що полегшує навчання. ReLU (Rectified Linear Unit, випрямлена лінійна одиниця) є найпоширенішою функцією активації для прихованих шарів. ReLU прискорює навчання через простоту обчислення градієнта та відсутність зникаючого градієнта для позитивних значень.

Пряме поширення (Forward propagation) обчислює активації шарів послідовно від входу до виходу. Для шару активація визначається як:

$$a^{(\ell)} = \sigma(W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}), \tag{2.11}$$

де $W^{(\ell)}$ - матриця ваг, $b^{(\ell)}$ - вектор зміщень.

Зворотне поширення (Backpropagation) - алгоритм навчання нейронних мереж через градієнтний спуск. Алгоритм обчислює градієнти функції втрат відносно кожної ваги через застосування правила ланцюга[13]. Процес починається з вихідного шару, де обчислюється похідна втрат відносно активацій.

Для кожного шару зворотне поширення обчислює градієнт відносно значень до активації.

Оптимізатор Adam комбінує ідеї інерції та адаптивних швидкостей навчання. Алгоритм підтримує окремі адаптивні швидкості навчання для кожного параметра на основі перших та других моментів градієнтів. Adam типово використовується зі швидкістю навчання 0.001 та демонструє швидку збіжність[13].

2.3.2 Архітектури глибоких нейронних мереж

Глибоке навчання (Deep Learning) оперує нейронними мережами з багатьма прихованими шарами, здатними вивчати ієрархічні представлення даних. Глибокі архітектури автоматично витягують ознаки від низькорівневих до високорівневих, зменшуючи потребу в ручній інженерії ознак.

Прямі нейронні мережі (Feedforward Neural Networks, FNN) або багатошарові перцептрони (multilayer perceptrons, MLP) являють собою найпростішу глибоку архітектуру з повнозв'язними шарами. Кожен нейрон в шарі з'єднаний з усіма нейронами шару . FNN для виявлення шкідливого ПЗ типово включає вхідний шар розмірності, що відповідає кількості ознак, 2-3 приховані шари з активацією ReLU та вихідний шар з софтмаксом. Дослідження послідовностей викликів API для класифікації шкідливого ПЗ показують, що FNN досягає точності понад 80% [7]. Однак FNN демонструє нижчу продуктивність порівняно з згортковими нейронними мережами (CNN) або LSTM для послідовних даних. Простота архітектури FNN робить його базовим рівнем для порівняння з більш складними моделями.

Щільні шари (Dense layers) з великою кількістю нейронів здатні моделювати складні нелінійні взаємозв'язки. Типова архітектура для виявлення шкідливого ПЗ включає шари з 128, 192 або 256 нейронами. Регуляризація через дропаут запобігає перенавчанню.

Пакетна нормалізація (Batch Normalization) нормалізує активації кожного шару, прискорюючи навчання та покращуючи стабільність. Техніка дозволяє використовувати вищі швидкості навчання та зменшує чутливість до ініціалізації ваг.

Глибокі мережі автоматично вивчають розрізняльні (discriminative) ознаки безпосередньо з необроблених даних (raw data), що є критичною перевагою для виявлення еволюціонуючого шкідливого ПЗ (evolving malware). Традиційні методи машинного навчання вимагають експертних знань предметної області (domain expertise) для інженерії ознак, тоді як моделі

глибокого навчання адаптуються до нових патернів через тонке налаштування (fine-tuning) на нових даних[8].

2.3.3 Згорткові нейронні мережі для аналізу послідовностей

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) спеціалізуються на виявленні локальних патернів через операцію згортки[13]. Первісно розроблені для обробки зображень, CNN успішно застосовуються до одновимірних послідовностей, таких як послідовності викликів API або байтові послідовності шкідливого ПЗ. Згортковий шар (Convolutional layer) застосовує набір навчальних фільтрів до входу. Для 1D послідовності фільтр розміру ковзає вздовж входу, обчислюючи скалярний добуток між вагами фільтра та відповідним вікном входу. Це генерує карту ознак, що відображає наявність специфічних патернів в різних позиціях послідовності. Формально одновимірна згортка визначається як:

$$(x * w)[i] = \sum^{k-1} x[i + j] \cdot w[j], \quad (2.12)$$

де - вхідна послідовність, - фільтр, - розмір ядра.

Кожен фільтр виявляє специфічний патерн (наприклад, послідовність певних викликів API, характерну для шкідливого ПЗ). Архітектура CNN для виявлення шкідливого ПЗ типово включає множину згорткових шарів з різною кількістю фільтрів. Перший шар може мати 48 фільтрів, другий 32, третій 64. Фільтри навчаються автоматично виявляти розрізняльні патерни через зворотне поширення.

Максимальне об'єднання (MaxPooling) зменшує просторову розмірність карт ознак, вибираючи максимальне значення в кожному вікні. Об'єднання забезпечує інваріантність до зсувів - модель розпізнає патерн незалежно від його точної позиції в послідовності[7]. Це критично важливо для виявлення шкідливого ПЗ, оскільки зловмисні послідовності викликів API можуть з'являтися в різних місцях трасування виконання.

Conv1D-шари ефективно виявляють просторові патерни в послідовностях викликів API. Модель CNN досягає найвищої точності (92%)

серед FNN, LSTM та двоспрямованого LSTM для класифікації шкідливого ПЗ на основі викликів API[7]. Швидкість навчання CNN також вища завдяки меншій кількості параметрів порівняно з повнозв'язними мережами.

Перетворення шкідливого ПЗ в зображення в градаціях сірого та застосування CNN демонструє високу точність. Байтові послідовності PE-файлів візуалізуються як зображення розмірності 32x32 або інших, де CNN виявляє структурні особливості сімейств шкідливого ПЗ.

2.3.4 Рекурентні та LSTM-мережі

Рекурентні нейронні мережі (Recurrent Neural Networks, RNN) спеціалізуються на обробці послідовних даних через механізм пам'яті, який зберігає інформацію з попередніх кроків[13]. На відміну від прямих мереж, RNN має зворотні зв'язки, дозволяючи інформації циркулювати всередині мережі. RNN обробляє послідовність $x = [x_1, x_2, \ldots, x_T]$ ітеративно. На кожному часовому кроці прихований стан оновлюється як:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \tag{2.13}$$

де W_{xh} та W_{hh} - матриці ваг, h_{t-1} - попередній прихований стан.

Вихід визначається як $y_t = W_{hy}h_t + b_y$ Стандартні RNN страждають від проблеми зникаючого градієнта при навчанні на довгих послідовностях. Градієнти експоненційно зменшуються при зворотному поширенні через багато часових кроків, що унеможливлює навчання довгострокових залежностей.

Мережі довгої короткострокової пам'яті (Long Short-Term Memory, LSTM) вирішують проблему зникаючого градієнта через спеціалізовану архітектуру комірок пам'яті[13]. Комірка LSTM включає вентиль забування, вхідний вентиль (input gate) та вихідний вентиль (output gate), які контролюють потік інформації. Вентиль забування визначає, яку інформацію з попереднього стану комірки видалити:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f), \tag{2.14}$$

LSTM здатен навчатися залежностям на відстані сотень часових кроків. Це критично важливо для аналізу послідовностей API шкідливого ПЗ, де зловмисні патерни можуть бути розділені багатьма безпечними (benign) викликами. Набір даних MAL-API-2019 з 7107 послідовностей API для восьми сімейств шкідливого ПЗ був класифікований з використанням одношарового та двошарового LSTM. Двошаровий LSTM забезпечує вищу точність через здатність моделювати більш складні ієрархічні патерни.

Двоспрямований LSTM (Bidirectional LSTM, Bi-LSTM) обробляє послідовність в обох напрямках - прямому та зворотному. Два окремі шари LSTM проходять послідовність від початку до кінця та від кінця до початку, після чого їх приховані стани конкатенуються. Двоспрямований LSTM враховує як попередній, так і наступний контекст для кожного елемента послідовності. Двоспрямований LSTM демонструє точність близько 90% для класифікації шкідливого ПЗ на викликах API. Проте навчання двоспрямованого LSTM вимагає більше часу та пам'яті порівняно з односпрямованим LSTM.

2.4 Застосування нейронних мереж для виявлення шкідливого ПЗ

2.4.1 Підходи на основі глибокого навчання

Гібридні архітектури CNN-LSTM комбінують переваги обох типів мереж для комплексного аналізу шкідливого ПЗ. Шари CNN спочатку витягують локальні просторові ознаки з послідовності, після чого шари LSTM моделюють часові залежності між цими ознаками. Типова архітектура CNN-LSTM включає 2-3 згорткові шари для витягування ознак, за якими слідує шар LSTM для моделювання послідовності. Фінальний щільний шар з активацією сигмоїд або софтмакс генерує вихід класифікації. Така архітектура автоматично навчається розрізняльним ознакам, мінімізуючи потребу в ручній інженерії ознак.

Дослідження виявлення шкідливого ПЗ у PE-файлах Windows демонструють, що CNN-LSTM досягає повноти 0.99 для зловмисних

екземплярів. Висока повнота критично важлива для виявлення шкідливого ПЗ, оскільки пропуск шкідливого зразка має серйозніші наслідки порівняно з помилково позитивним. Однак помірна точність позитивного 0.76 свідчить про наявність помилково позитивних результатів. CNN-LSTM для виявлення безфайлового шкідливого ПЗ через аналіз зображень мережевого трафіку показує ефективність виявлення загроз, які не залишають файлових слідів. Внаслідок цього підхід особливо релевантний для виявлення шкідливого ПЗ в пам'яті. Дослідження виявлення шкідливого ПЗ в пам'яті демонструє, що комбінація CNN з LSTM забезпечує ефективний підхід для аналізу дампів пам'яті. CNN витягує структурні патерни з областей пам'яті, тоді як LSTM моделює часову еволюцію зловмисної активності.

Механізм уваги (Attention mechanism) в поєднанні з CNN-LSTM покращує фокусування моделі на найбільш релевантних частинах послідовності. Самоувага (Self-attention) дозволяє мережі автоматично визначати, які ознаки або часові кроки найважливіші для класифікації. Модель Attention-CNN-LSTM для мережевої безпеки демонструє вищу продуктивність для виявлення складних атак. Перетворення дампів пам'яті в зображення з подальшим застосуванням CNN - популярний підхід виявлення шкідливого ПЗ на основі візуалізації. Гістограми орієнтованих градієнтів витягуються з зображень дампів пам'яті в градаціях сірого, після чого класифікуються багатошаровим перцептроном з високою точністю[8].

Трансферне навчання (Transfer learning) дозволяє використовувати попередньо навчені моделі, навчені на великих загальних наборах даних, та тонко налаштовувати їх для специфічних задач виявлення шкідливого ПЗ. Це особливо корисно, коли розмічені набори даних шкідливого ПЗ обмежені.

Автокодувальники (Autoencoders) для навчання без вчителя витягують стиснуті представлення дампів пам'яті. Навчений кодувальник (encoder) використовується для генерації ознак, які потім класифікуються алгоритмами

з вчителем. Це дозволяє використовувати великі обсяги нерозмічених даних для покращення представлень[8].

2.4.2 Переваги та обмеження нейромережевих моделей

Основна перевага глибокого навчання для виявлення шкідливого ПЗ полягає в автоматичному навчанні ознак. Моделі глибокого навчання витягують розрізняльні ознаки безпосередньо з необроблених даних (байти, виклики API, дампи пам'яті), уникаючи необхідності ручного витягування ознак. Це критично важливо, оскільки ручне витягування ознак вимагає експертних знань предметної області та може пропускати важливі патерни[13]. Нейронні мережі адаптуються до еволюціонуючих загроз через повторне навчання на нових зразках. Поліморфне та метаморфне шкідливе ПЗ, які змінюють власну структуру, можуть бути виявлені через вивчені представлення, навіть якщо специфічні сигнатури відрізняються. Здатність моделювати складні нелінійні взаємозв'язки дозволяє глибокому навчанню виявляти тонкі патерни, недоступні лінійним класифікаторам. Глибокі мережі з множиною шарів створюють ієрархічні представлення - від низькорівневих байтових патернів до високорівневих семантичних поведінок. Універсальність архітектур дозволяє застосовувати схожі моделі до різних типів даних - зображень, послідовностей, табличних даних. Архітектура CNN-LSTM успішно працює як з мережевим трафіком, так і з послідовностями викликів API або дампами пам'яті.

Однак нейромережеві моделі мають значні обмеження. Вимогливість до даних - глибоке навчання потребує великих обсягів розмічених зразків для ефективного навчання. Малі набори даних призводять до перенавчання, коли модель запам'ятовує тренувальні приклади замість навчання узагальнювальним патернам. Обчислювальна складність навчання глибоких мереж вимагає потужних графічних процесорів (GPU) та значного часу[7]. Моделі CNN-LSTM можуть навчатися години або дні на великих наборах даних шкідливого ПЗ. Це обмежує можливості швидкого повторного навчання

при появі нових загроз. Інтерпретованість моделей глибокого навчання залишається проблемою. Глибокі мережі є "чорними скриньками" - складно зрозуміти, чому модель прийняла конкретне рішення. Для критичних застосувань безпеки необхідна можливість пояснення рішень класифікації, що ускладнюється в глибокому навчанні. Вразливість до змагальних атак - зловмисники можуть спеціально створити входи, які обманюють нейронну мережу. Незначні модифікації шкідливого ПЗ можуть змінити передбачення, не впливаючи на функціональність. Залежність від якості даних - моделі глибокого навчання чутливі до упереджень в тренувальних даних. Якщо набір даних не репрезентує реальний розподіл типів шкідливого ПЗ, модель демонструватиме низьку продуктивність на робочих даних.

CNN демонструють гарну точність на приблизному рівні з LSTM та FNN для класифікації шкідливого ПЗ на основі API. Швидкість навчання CNN також переважає завдяки меншій кількості параметрів. Проте для дуже довгих послідовностей LSTM можуть краще захоплювати довгодіапазонні залежності. Класичні методи машинного навчання (випадковий ліс, SVM) часто демонструють конкурентну продуктивність при значно меншій обчислювальній складності. Випадковий ліс досягає 100% точності на деяких наборах даних шкідливого ПЗ[16]. Внаслідок цього вибір між глибоким навчанням та класичними методами залежить від специфічних вимог щодо точності, швидкості та інтерпретованості.

2.5 Метрики оцінювання моделей класифікації

2.5.1 Точність, повнота, F-міра

Оцінка продуктивності моделей класифікації вимагає комплексу метрик, які відображають різні аспекти якості передбачень.

Точність (Accuracy) визначається як частка правильно класифікованих зразків від загальної кількості:

$$Accuracy=((TP+TN))/((TP+TN+FP+FN)) \quad (2.14)$$

де TP (True Positive) - правильно класифіковані зразки шкідливого ПЗ, TN (True Negative) - правильно класифіковані безпечні файли (benign files), FP (False Positive) - безпечні файли, помилково класифіковані як шкідливе ПЗ, FN (False Negative) - шкідливе ПЗ, пропущене моделлю.

Точність інтуїтивно зрозуміла, але може бути оманливою для незбалансованих наборів даних. Якщо 95% зразків є безпечними, модель, яка завжди передбачає «benign», досягне 95% точності, але буде абсолютно неефективною для виявлення шкідливого ПЗ.

Влучність (Precision) вимірює частку справжніх зразків шкідливого ПЗ серед усіх зразків, класифікованих як шкідливе ПЗ:

$$Precision=TP/(TP+FP) \quad (2.15)$$

Висока влучність означає низький рівень хибних спрацювань - коли модель класифікує файл як шкідливий, вона рідко помиляється, що важливо для зменшення навантаження на аналітиків безпеки, які перевіряють сповіщення.

Повнота (Recall) вимірює частку виявленого шкідливого ПЗ від усього фактичного шкідливого ПЗ:

$$Recall=TP/(TP+FN) \quad (2.16)$$

Висока повнота означає, що модель пропускає мало зловмисних файлів, і для задач виявлення шкідливого ПЗ повнота критично важливіша за влучність, оскільки пропущений зразок (FN) може спричинити серйозні наслідки[14]. Компроміс між влучністю та повнотою є фундаментальним. Зменшення порогу класифікації збільшує повноту (більше зразків класифікується як шкідливі), але знижує влучність (більше хибно позитивних), а оптимальний баланс залежить від конкретних вимог безпеки.

F1-міра (F1-score) є гармонійним середнім влучності та повноти:

$$F1=(Precision \cdot Recall)/(Precision+Recall)=2TP/(2TP+FP+FN) \quad (2.17)$$

F1-міра забезпечує єдину метрику, що балансує обидва аспекти, і через використання гармонійного середнього її значення буде низьким, якщо хоча б одна з метрик (precision або recall) низька.

Специфічність (Specificity, True Negative Rate) вимірює частку правильно класифікованих безпечних файлів:

$$Specificity=TN/(TN+FP) \quad (2.18)$$

Висока специфічність означає низьку частоту хибно позитивних спрацювань (False Positive Rate), що зменшує кількість хибних тривог.

Дослідження виявлення шкідливого ПЗ на основі пам'яті оцінювали моделі за точністю, частотою хибно позитивних спрацювань (False Positive Rate, FPR) та статистичним t-критерієм. FPR визначається як

$$FPR=FP/(FP+TN) \quad (2.19)$$

та є критично важливим для практичного розгортання систем виявлення[14].

2.5.2 Матриця помилок та ROC-аналіз

Матриця помилок (Confusion Matrix) візуалізує продуктивність моделі класифікації через табличне представлення фактичних та передбачених класів. Рядки матриці відповідають фактичним класам, стовпці - передбаченим класам, а кожна комірка містить кількість зразків, які належать до певного фактичного класу та були передбачені як певний прогнозований клас.

Для бінарної класифікації матриця помилок є таблицею 2x2 з елементами TP, TN, FP, FN. Для багатокласової класифікації розмірність матриці дорівнює , де - кількість класів. Діагональні елементи представляють правильні класифікації, тоді як позадіагональні - помилки.

Інтерпретація багатокласової матриці помилок вимагає аналізу кожного класу окремо. Для класу істинно позитивні (True Positive) відповідають діагональному елементу , хибно позитивні (False Positive) - це сума стовпця без діагонального елемента (зразки інших класів, помилково класифіковані як), а хибно негативні (False Negative) - сума рядка без діагонального елемента

(зразки класу , класифіковані як інші класи). Істинно негативні (True Negative) - це всі інші елементи матриці. Матриця помилок для сімейств шкідливого ПЗ дозволяє виявити специфічні шаблони помилок. Наприклад, якщо троян (Trojan) часто класифікується помилково як бекдор (Backdoor), це свідчить про схожість їх поведінкових патернів, а аналіз помилок допомагає зрозуміти обмеження моделі та можливі напрямки її покращення.

Крива ROC (Receiver Operating Characteristic, ROC curve) відображає компроміс між істинно позитивною частотою (True Positive Rate, TPR = Recall) та частотою хибно позитивних (False Positive Rate, FPR) при варіюванні порогу класифікації. TPR відкладається по вертикальній осі, а FPR - по горизонтальній осі, і кожна точка на ROC-кривій відповідає певному значенню порога.

Висновки за розділом 2

У другому розділі проведено комплексний аналіз методів машинного та глибокого навчання в контексті задачі виявлення шкідливого програмного забезпечення, зокрема на основі аналізу дампів оперативної пам’яті. Було визначено, що специфіка предметної області - сильний дисбаланс класів, висока вартість помилок та еволюційний характер загроз - диктує необхідність переходу від простих сигнатурних методів до адаптивних моделей, здатних виявляти складні залежності у багатовимірних даних. Розглянуто два ключові формати постановки задачі: бінарну класифікацію для розмежування легітимного та шкідливого ПЗ, а також багатокласову класифікацію для ідентифікації конкретних сімейств загроз, що є критично важливим для коректного реагування на інциденти.

Детальний огляд класичних методів навчання з учителем показав, що алгоритми на кшталт методу опорних векторів (SVM) та k-найближчих сусідів (k-NN) можуть бути ефективними на невеликих вибірках або специфічних ознаках, проте мають обмеження щодо масштабованості та обчислювальної ефективності. Натомість ансамблеві методи, зокрема випадковий ліс (Random

Forest) та градієнтний бустинг (XGBoost, LightGBM), демонструють високу стійкість до шуму та здатність працювати зі змішаними типами даних, що робить їх надійним вибором для побудови базових моделей виявлення.

Особливу увагу приділено методам глибокого навчання, які дозволяють автоматизувати процес вилучення ознак безпосередньо з «сирих» даних. Аналіз архітектур згорткових нейронних мереж (CNN) та мереж довгої короткострокової пам’яті (LSTM) підтвердив їхню перевагу в роботі зі структурованими послідовностями байтів або API-викликів, характерними для дампів пам’яті. Обґрунтовано перспективність використання гібридних архітектур CNN-LSTM, які поєднують здатність витягувати локальні патерни зі здатністю моделювати часові залежності, що забезпечує високу точність виявлення як відомих, так і нових загроз.

Також у розділі систематизовано підходи до оцінювання якості моделей, наголошуючи на важливості використання метрик Precision, Recall та F1-score замість простої Accuracy в умовах незбалансованих даних. Розглянуто методи відбору ознак, такі як взаємна інформація, що дозволяють зменшити розмірність даних і підвищити інтерпретованість моделей. Загалом, проведений теоретичний аналіз формує необхідне підґрунтя для наступного етапу роботи - практичної розробки та навчання моделі виявлення шкідливого ПЗ на основі дампів пам’яті.

РОЗДІЛ 3
РОЗРОБКА ТА ДОСЛІДЖЕННЯ КОМП'ЮТЕРНОЇ МОДЕЛІ

3.1 Опис експериментальних даних

3.1.1 Набір даних та його характеристики

В якості експериментальної бази використано набір даних CIC-MalMem-2022 (Database of Obfuscated-MalMem2022), розроблений Canadian Institute for Cybersecurity університету New Brunswick. Цей набір даних спеціально спроектований для тестування методів виявлення обфускованого шкідливого ПЗ через аналіз дампів оперативної пам'яті. На відміну від попередніх наборів даних, CIC-MalMem-2022 максимально наближений до реальних сценаріїв атак, використовуючи актуальне шкідливе ПЗ, поширене в сучасному кіберпросторі[3].

Набір даних містить 58,596 записів дампів пам'яті, що робить його одним з найбільших публічно доступних наборів даних для виявлення шкідливого ПЗ на основі пам'яті. Кожен запис представляє знімок оперативної пам'яті системи Windows в момент виконання доброякісної або шкідливої програми. Процес захоплення дампів здійснювався в режимі відладки, що дозволило уникнути появи самого процесу дампування в знімках пам'яті. Внаслідок цього набір даних відображає реальну картину пам'яті, яку має середньостатистичний користувач під час атаки шкідливого ПЗ.

Набір даних включає чотири класи для багатокласової класифікації. Розподіл зразків за класами наведено на рисунку 3.1.

Рисунок 3.1 – Гістограма розподілу класів.

Клас доброякісних програм містить 29,298 зразків (50.0% від загального обсягу) - дампи пам'яті легітимних програм під час нормального функціонування системи. Три категорії шкідливого ПЗ представлені класами програм-шпигунів (10,020 зразків, 17.1%), програм-вимагачів (9,791 зразків, 16.7%) та троянів (9,487 зразків, 16.2%). Розподіл забезпечує збалансованість між доброякісними та шкідливими зразками (50% на 50%), проте всередині категорій шкідливого ПЗ спостерігається незначний дисбаланс.

Кожен клас шкідливого ПЗ представлений множиною реальних сімейств шкідливого ПЗ. Програми-шпигуни включають сімейства, що спеціалізуються на збиранні конфіденційної інформації користувача. Програми-вимагачі представлені зразками програм, які шифрують файли жертви. Клас троянів містить троянські програми різної функціональності - від програм бекдору до програм завантаження.

Набір даних складається з 54 числових ознак, витягнутих з дампів пам'яті за допомогою Volatility Framework та власних скриптів аналізу[10]. Ознаки відображають різноманітні аспекти стану пам'яті - характеристики процесів, мережеві з'єднання, завантажені бібліотеки, системні об'єкти, реєстр,

виклики API. Ознаки мають різні шкали вимірювання - від підрахунків (кількість процесів, дескрипторів) до розмірів пам'яті (розмір виділеної пам'яті). Обфускація шкідливого ПЗ в наборі даних досягається через використання пакувальників, криптерів та інших технік приховування коду. Це робить статичний аналіз неефективним та вимагає підходів на основі пам'яті для виявлення справжньої поведінки.

Вибір саме цього набору даних обґрунтовується кількома факторами. По-перше, фокус на обфускованому шкідливому ПЗ відповідає сучасним трендам розвитку загроз, де більшість складного шкідливого ПЗ використовує техніки приховування. По-друге, підхід на основі пам'яті дозволяє виявляти шкідливе ПЗ без файлів та отримувати розпакований код пакованих зразків. По-третє, баланс класів та значний обсяг даних забезпечують надійне навчання та валідацію моделей.

3.1.2 Попередня обробка даних

Попередня обробка даних є критичним етапом підготовки набору даних для навчання моделей машинного навчання. Процес включає створення цільової змінної, поділ на навчальну та тестову вибірки, стандартизацію ознак.

Оригінальний набір даних містить бінарну змінну Class з значеннями "Benign" та "Malware". Для постановки задачі багатокласової класифікації створено нову змінну MalwareType на основі поля Category. Логіка формування класів передбачає присвоєння мітки "Benign" для всіх зразків з Class == "Benign", а для шкідливих зразків - відповідних значень з Category (Spyware, Ransomware, Trojan). Внаслідок цього отримано чотирикласову задачу класифікації замість бінарної. Трансформація цільової змінної з категоріального у числовий формат здійснена за допомогою LabelEncoder. Кодувальник присвоює кожному класу унікальний цілочисловий ідентифікатор, зберігаючи відображення для подальшої інтерпретації результатів. Це дозволяє алгоритмам машинного навчання працювати з числовими мітками класів.

Поділ набору даних на навчальну та тестову вибірки виконано з використанням функції train_test_split з параметрами test_size=0.2 та stratify=y. Стратифікований поділ гарантує, що розподіл класів у тренувальній та тестовій вибірках відповідає розподілу в повному наборі даних. Це особливо важливо для дисбалансованих наборів даних, де випадковий поділ може призвести до недостатньої представленості класів меншості у тестовій вибірці. Внаслідок цього навчальна вибірка містить 46,876 зразків (80%), тестова - 11,720 зразків (20%).

Стандартизація ознак виконана за допомогою StandardScaler. Метод перетворює кожну ознаку так, щоб вона мала нульове середнє та одиничну дисперсію.. Стандартизатор навчається (fit) на тренувальному наборі даних та застосовується (transform) до обох вибірок. Критично важливо не використовувати тестові дані під час fit, щоб уникнути витоку даних. Стандартизація необхідна з кількох причин. По-перше, ознаки в наборі даних мають різні шкали - наприклад, кількість процесів може коливатися від 50 до 200, тоді як розмір області пам'яті вимірюється в мегабайтах. Без масштабування алгоритми, чутливі до величини ознак (логістична регресія, нейронні мережі), будуть надавати більшу вагу ознакам з більшими значеннями. По-друге, стандартизація прискорює збіжність градієнтної оптимізації в нейронних мережах. Альтернативні техніки попередньої обробки розглянуті, але не застосовані в фінальному pipeline. Нормалізація (L2-норма) корисна для алгоритмів на основі відстані, проте стандартне масштабування більш робастне до викидів. Масштабування мін-макс стискає значення в діапазон, але чутлива до екстремальних значень у тестових даних. Методи відбору ознак (PCA, LDA, на основі кореляції) можуть покращити продуктивність, однак збільшують складність pipeline.

3.2 Розвідувальний аналіз даних

3.2.1 Описова статистика

Розвідувальний аналіз даних виконано для глибокого розуміння структури набору даних, виявлення закономірностей та потенційних проблем. Описова статистика надає кількісні характеристики розподілу ознак та класів.

Аналіз розподілу цільової змінної виявив помірний дисбаланс класів. Клас доброякісних програм складає 50.0% набору даних (29,298 зразків), що забезпечує достатню кількість негативних прикладів. Серед категорій шкідливого ПЗ програми-шпигуни представлені 10,020 зразками (17.1%), програми-вимагачі - 9,791 (16.7%), трояни - 9,487 (16.2%). Різниця між найбільшим класом шкідливого ПЗ (програми-шпигуни) та найменшим (трояни) становить лише 5.6%, що вказує на відносну збалансованість.

Статистичний аналіз 54 числових ознак виявив широкий діапазон значень та розподілів. Частина ознак має дискретний характер - підрахунки процесів, дескрипторів, бібліотек DLL. Інші ознаки є неперервними - розміри пам'яті, часові позначки, відсотки. Деякі ознаки містять багато нульових значень, що може свідчити про відсутність певної активності в доброякісних зразках. Детальний опис основних груп ознак наведено в табл. 3.1:

Таблиця 3.1

Типові артефакти основних класів шкідливого ПЗ в оперативній пам’яті

Група ознак	**Приклади полів у датасеті**	**Опис**
1. Процеси (pslist)	pslist.nproc, pslist.nppid, pslist.avg_threads, pslist.nprocs64bit, pslist.avg_handlers	Характеристики процесів: кількість процесів, кількість потоків, кількість процесів-батьків, середня кількість хендлерів. Вказує на загальне навантаження та поведінку системи.
2. Завантажені DLL (dlllist)	dlllist.ndlls, dlllist.avg_dlls_per_proc	Кількість завантажених бібліотек, середнє навантаження DLL на процес. Аномальні DLL можуть вказувати на ін’єкцію коду.

3. Хендлери (handles)	handles.nhandles, handles.avg_handles_per_proc	Показники використання системних хендлерів: файлів, потоків, mutex тощо. Нестандартно високі значення - підозра на шкідливу активність.
4. Мережеві сокети (netscan)	netscan.netscan, netscan.tcp, netscan.udp, netscan.ip	Кількість активних мережевих з'єднань. Часто використовуються для виявлення бекдорів, C2-каналів та мережевої активності шкідливих програм.
5. Входи до таблиць потоків (thrdscan)	thrdscan.nthrdscan, thrdscan.ninjections	Показники потоків, включно з кількістю виявлених ін'єкцій. Пряма ознака можливих інструментів атаки.
6. Пам'ять процесів (memmap)	memmap.nmemmap	Кількість відображень пам'яті у процесах. Нестандартні карти пам'яті сигналізують про приховані секції або шеллкод.
7. Сторінкова пам'ять (malfind)	malfind.ninjected, malfind.commitCharge, malfind.protection	Ознаки ін'єкцій пам'яті, аномальних прав доступу та зарезервованої пам'яті. Один із ключових індикаторів шкідливих процесів.
8. Файли (filescan)	filescan.nfilescan, filescan.ncreated, filescan.text_files	Метадані файлів у пам'яті: загальна кількість, новостворені файли, типи файлів. Корисно для виявлення тимчасових шкідливих файлових артефактів.
9. Ядро (modscan / modules)	modscan.nmodscan, modules.nmodules	Ознаки модулів ядра: кількість модулів, їх завантаження. Підозрілі kernel-модулі - ключовий маркер rootkit-активності.
10. Внутрішні об'єкти ядра (driverscan)	driverscan.ndrivers	Кількість драйверів, виявлених у пам'яті. Аномалії можуть означати завантаження несанкціонованих драйверів.
11. Канали зв'язку (mutantscan, symlinkscan, devicetree)	mutantscan.nmutants, symlinkscan.nsymlinkscan, devicetree.ndevicetree	Різні види внутрішніх IPC та системних об'єктів. Ненормальні або масові створення можуть бути частиною шкідливого ПЗ.
12. Процеси Windows GUI (wndscan, desktops)	wndscan.nwndscan, deskscan.ndeskscan	Графічні об'єкти Windows: вікна, десктопи. Атаки на GUI або приховані вікна можуть бути видимі у цих ознаках.
13. Служби (svcscan)	svcscan.nservices, svcscan.kernel_drivers, svcscan.fs_drivers, svcscan.nactive	Статистика служб Windows. Шкідливе ПЗ часто створює або змінює системні сервіси для автозапуску.
14. Процес-перевірка відповідності (psxview)	psxview.not_in_*_false_avg	Набір перевірок, що показує приховані процеси. Якщо процес не видно у кількох списках - ознака rootkit-маскування.

15. Зворотні виклики (callbacks)	callbacks.ncallbacks, callbacks.nanonymous, callbacks.ngeneric	Виклики ядра, пов’язані з процесами. Атаки на рівні ядра можуть використовувати підміну callback-функцій.
16. Класифікаційні мітки	Category, Class	Мітки для навчання моделі та категоризації: Benign / Malware, окремі родини шкідливих програм.

Середні значення ознак суттєво відрізняються між класами. Наприклад, кількість підозрілих викликів API значно вища для шкідливого ПЗ порівняно з доброякісним. Області пам'яті з виконуваними атрибутами більш поширені в зразках троянів та програм-шпигунів. Програми-вимагачі характеризуються високою активністю файлових операцій та використанням криптографічного API.

Стандартні відхилення ознак вказують на варіативність поведінки в межах кожного класу. Висока дисперсія пояснюється різноманітністю сімейств шкідливого ПЗ та доброякісних програм в наборі даних. Це позитивний фактор для узагальнення моделей на нові зразки.

Аналіз викидів виявив екстремальні значення в деяких ознаках, особливо пов'язаних з використанням пам'яті та кількістю процесів. Викиди можуть представляти як справжні аномалії (складне шкідливе ПЗ), так і помилки захоплення дампів. StandardScaler менш чутливий до викидів порівняно з масштабуванням мін-макс, що обґрунтовує вибір методу нормалізації.

3.2.2 Кореляційний аналіз

Кореляційний аналіз виконано для виявлення взаємозв'язків між ознаками та ідентифікації надлишкових ознак. Коефіцієнт кореляції Пірсона обчислений для всіх пар числових ознак. Матриця кореляцій розмірності 54x54 візуалізована за допомогою теплової карти.

Сильна позитивна кореляція ($r > 0.7$) виявлена між групами семантично пов'язаних ознак. Наприклад, різні метрики використання пам'яті (загальна виділена, приватні байти, робочий набір) високо корельовані між собою.

Виклики API з однієї категорії (мережеві API, файлові API) також демонструють сильні кореляції.

Високо корельовані ознаки створюють проблему мультиколінеарності для деяких алгоритмів. Логістична регресія та лінійний дискримінантний аналіз чутливі до мультиколінеарності, що може призвести до нестабільних коефіцієнтів. Методи на основі дерев (випадковийліс, градієнтний бустинг) більш робастні до корельованих ознак.

Негативні кореляції виявлені між ознаками, що представляють взаємовиключні поведінки. Високі значення індикаторів легітимної активності (наприклад, патерни взаємодії з користувачем) негативно корельовані з ознаками, специфічними для шкідливого ПЗ (підозрілі послідовності API).

Відбір ознак на основі кореляції може покращити продуктивність моделей через видалення надлишкових ознак. Метод передбачає відбір ознак з високою кореляцією з цільовою змінною та низькою міжознаковою кореляцією. Дослідження показують, що зменшення набору ознак на 30-50% з використанням відбору на основі кореляції може підвищити точність та значно прискорити навчання.

Однак повне видалення корельованих ознак не завжди оптимальне. Ансамблеві методи, такі як випадковийліс, можуть використовувати надлишкові ознаки для побудови більш робастних меж рішення. Внаслідок цього в базових моделях збережено всі 54 ознаки, а відбір ознак розглянуто як напрямок подальших досліджень.

Аналіз важливості ознак з випадкового лісу надає альтернативний погляд на значущість ознак. Топ-10 ознак за важливістю включають виклики API, пов'язані з впровадженням процесів, мережевою активністю, модифікаціями реєстру. Ці ознаки мають найвищі значення середнього зменшення в чистоті Джині під час побудови дерев.

3.3 Побудова моделей класифікації

Реалізовано 14 алгоритмів для багатокласової класифікації шкідливого ПЗ, що охоплюють лінійні моделі, деревоподібні алгоритми, ансамблеві методи та нейронні мережі. Основні гіперпараметри, кількість ітерацій навчання та використання масштабування ознак зведено в табл. 3.2.:

Таблиця 3.2

Конфігурація гіперпараметрів моделей

Модель	Тип	Основні гіперпараметри	Епохи/Ітерації	Batch Size	Масштабування
Logistic Regression	Класичне ML	max_iter=500, random_state=42, solver=lbfgs	500	-	StandardScaler
Decision Tree	Класичне ML	max_depth=15, random_state=42	-	-	Ні
Random Forest	Класичне ML	n_estimators=50, max_depth=15, random_state=42	50 дерев	-	Ні
Extra Trees	Класичне ML	n_estimators=50, max_depth=15, random_state=42	50 дерев	-	Ні
Gradient Boosting	Класичне ML	n_estimators=50, max_depth=5, random_state=42	50 ітерацій	-	Ні
AdaBoost	Класичне ML	n_estimators=50, random_state=42	50 ітерацій	-	Ні
K-Nearest Neighbors	Класичне ML	n_neighbors=5	-	-	StandardScaler
Naive Bayes	Класичне ML	за замовчуванням	-	-	StandardScaler
Linear Discriminant Analysis	Класичне ML	за замовчуванням	-	-	StandardScaler
Feedforward NN	Нейронна мережа	шари=, dropout=0.3, optimizer=adam	20 епох	256	StandardScaler

Deep NN (with BatchNorm)	Нейронна мережа	шари=, dropout=0.3-0.4, BatchNorm	20 епох	256	StandardScaler
NN with L2 Regularization	Нейронна мережа	шари=, dropout=0.3, L2_reg=0.01	20 епох	256	StandardScaler
Wide & Deep Network	Нейронна мережа	deep=, wide=, concatenate	20 епох	256	StandardScaler
CNN	Нейронна мережа	Conv1D: filters=, kernel_size=3, pooling=MaxPool1D	20 епох	256	StandardScaler

Логістична регресія використовується як базова лінійна модель із мультиномним розширенням для чотирьох класів; попереднє масштабування ознак забезпечує коректну роботу градієнтних методів оптимізації. Дерева рішень та їх ансамблі (випадковий ліс і додаткові дерева) реалізують нелінійне розбиття простору ознак, де глибину дерев і кількість базових оцінювачів підібрано так, щоб поєднати виразність моделі з контролем перенавчання.Ансамблеві методи бустингу (Gradient Boosting та AdaBoost) навчають послідовності слабких класифікаторів, поетапно зменшуючи помилки попередніх моделей; кількість ітерацій та глибина базових дерев узгоджуються з типово рекомендованими налаштуваннями для задач багатокласової класифікації. Метод k-найближчих сусідів працює як алгоритм на основі прикладів, тому критично залежить від масштабування ознак та вибору кількості сусідів, що враховано в таблиці. Гаусовий наївний Байєс і лінійний дискримінантний аналіз представляють імовірнісні підходи з припущенням про гаусівський розподіл ознак; вони поєднують низьку обчислювальну вартість із можливістю працювати в багатокласовому режимі. Для всіх моделей час навчання та прогнозування оцінюється для подальшого порівняння обчислювальної ефективності поряд із якістю класифікації.

Базова мережа прямого поширення з трьома прихованими шарами використовується як відправна точка, тоді як глибша мережа з пакетною

нормалізацією розширює її ємність і прискорює збіжність за рахунок нормалізації активацій. Окрема архітектура з L2-регуляризацією додає штраф на великі ваги, а широка й глибока мережа поєднує “широкий” лінійний шлях із “глибоким” нелінійним, що дозволяє одночасно моделювати як прості асоціації, так і складні взаємодії ознак. CNN інтерпретує вектор ознак як одновимірну послідовність і застосовує згортки з подальшим субдискретизуванням, що дає змогу автоматично виділяти локальні патерни у просторі ознак. Усі нейромережеві моделі навчаються з оптимізатором Adam, функцією втрат категоріальної крос-ентропії й оцінюються за точністю класифікації, що забезпечує коректне порівняння з класичними алгоритмами.

3.4 Експериментальні дослідження

3.4.1 Порівняння продуктивності моделей

Результати багатокласової класифікації демонструють значну варіативність продуктивності між різними алгоритмами. Метрики оцінювання включають точність, влучність, повноту, F1-міру (зважену) та час навчання.

За результатами експериментів сформовано зведену табл. 3.3 метрик продуктивності для всіх 13 моделей:

Таблиця 3.3

Зведені результати ефективності моделей класифікації

Algorithm	Type	Accuracy	Precision	Recall	F1-Score	Training Time (s)
Random Forest	Classical ML	0.8549	0.8558	0.8549	0.8552	0.95
Gradient Boosting	Classical ML	0.8457	0.8463	0.8457	0.8458	83.4
Decision Tree	Classical ML	0.8445	0.8458	0.8445	0.8449	0.75
K-Nearest Neighbors	Classical ML	0.8119	0.8131	0.8119	0.8117	3.46
Extra Trees	Classical ML	0.8044	0.8117	0.8044	0.8046	0.45
Feedforward NN	Neural Network	0.7975	0.801	0.7975	0.7978	35.18
Wide & Deep Network	Neural Network	0.783	0.7902	0.783	0.7816	16.75
Deep NN (with BatchNorm)	Neural Network	0.7634	0.7816	0.7634	0.7535	25.85
Logistic Regression	Classical ML	0.7401	0.7459	0.7401	0.7396	11.15
CNN	Neural Network	0.748	0.782	0.748	0.7364	125.92
Linear Discriminant Analysis	Classical ML	0.7139	0.7199	0.7139	0.7099	0.21
AdaBoost	Classical ML	0.714	0.7464	0.714	0.6968	6.1
NN with L2 Regularization	Neural Network	0.7044	0.7295	0.7044	0.6773	15.89
Naive Bayes	Classical ML	0.6784	0.7266	0.6784	0.6328	0.08

Випадковий ліс досяг найвищої точності 85.49% серед усіх тринадцяти тестованих моделей. Влучність 85.58%, повнота 85.49% та F1-міра 85.52% демонструють збалансовану продуктивність по всіх класах. Час навчання лише 0.95 секунди робить випадковий ліс оптимальним вибором за співвідношенням точність/швидкість. Ансамблева природа випадкового лісу дозволяє ефективно захоплювати складні патерни в ознаках пам'яті.

Градієнтний бустинг посів друге місце з точністю 84.57%. Влучність 84.63%, повнота 84.57%, F1-міра 84.58% лише незначно поступаються випадковому лісу. Однак час навчання 83.4 секунди у багато разів довший порівняно з випадковим лісом. Послідовна природа бустингу пояснює обчислювальні витрати. Градієнтний бустинг може бути кращим вибором, коли максимальна точність критична, а час навчання менш важливий.

Дерево рішень показало точність 84.45% при найшвидшому часі навчання 0.75 секунди серед лідерів продуктивності. Близька продуктивність до ансамблевих методів є несподіваною та може вказувати на перенавчання через необмежену глибину дерева. Однак баланс між влучністю (84.58%) та повнотою (84.45%) свідчить про адекватне узагальнення.

Метод k-найближчих сусідів досяг точності 81.19% при часі навчання 3.46 секунди. Метод як алгоритм на основі прикладів не має фази навчання в традиційному розумінні - час відображає індексування навчальних даних. Час виведення для методу буде значно вищим через необхідність обчислення відстаней до всіх навчальних зразків.

Додаткові дерева з точністю 80.44% демонструють нижчу продуктивність порівняно з випадковим лісом, незважаючи на швидше навчання (0.45с). Випадковий вибір точок поділу замість оптимального пошуку знижує якість окремих дерев.

Нейронні мережі показали систематично нижчу точність порівняно з класичними методами машинного навчання. Найкраща нейромережева

модель (нейронна мережа прямого поширення) досягла 79.75% точності. Глибока нейронна мережа з пакетною нормалізацією - 76.34%, широка і глибока - 78.30%, CNN - 74.80%, нейронна мережа з L2-регуляризацією - 70.44%. Час навчання нейронних мереж (16–35 секунд) значно вищий за більшість класичних методів. Нижча продуктивність нейронних мереж може пояснюватися кількома факторами. По-перше, набір даних містить структуровані табличні дані з вручну розробленими ознаками, де методи на основі дерев історично переважають. По-друге, обсяг навчальних даних (46,876 зразків) може бути недостатнім для повноцінного навчання глибоких мереж.

Логістична регресія з точністю 74.01% значно поступається лідерам продуктивності. Лінійна природа алгоритму обмежує здатність моделювати складні нелінійні взаємодії між ознаками. Час навчання 11.15 секунди відносно високий для лінійного методу через ітеративну оптимізацію.

Лінійний дискримінантний аналіз (71.39%) та AdaBoost (71.40%) демонструють схожу низьку продуктивність. Припущення лінійного дискримінантного аналізу про гаусові розподіли може не виконуватися для ознак пам'яті. AdaBoost з лише 50 оцінювачами може потребувати більшої кількості ітерацій для збіжності.

Наївний Байєс показав найнижчу точність 67.84% при найшвидшому часі навчання 0.08 секунди. Припущення про незалежність ознак явно порушується для високо корельованих ознак пам'яті. Проте екстремальна швидкість робить наївний Байєс корисним для швидкого прототипування або застосунків реального часу з менш строгими вимогами до точності.

Для наочного порівняння точності моделей можна звернутись до діаграми на рисунку 3.2.

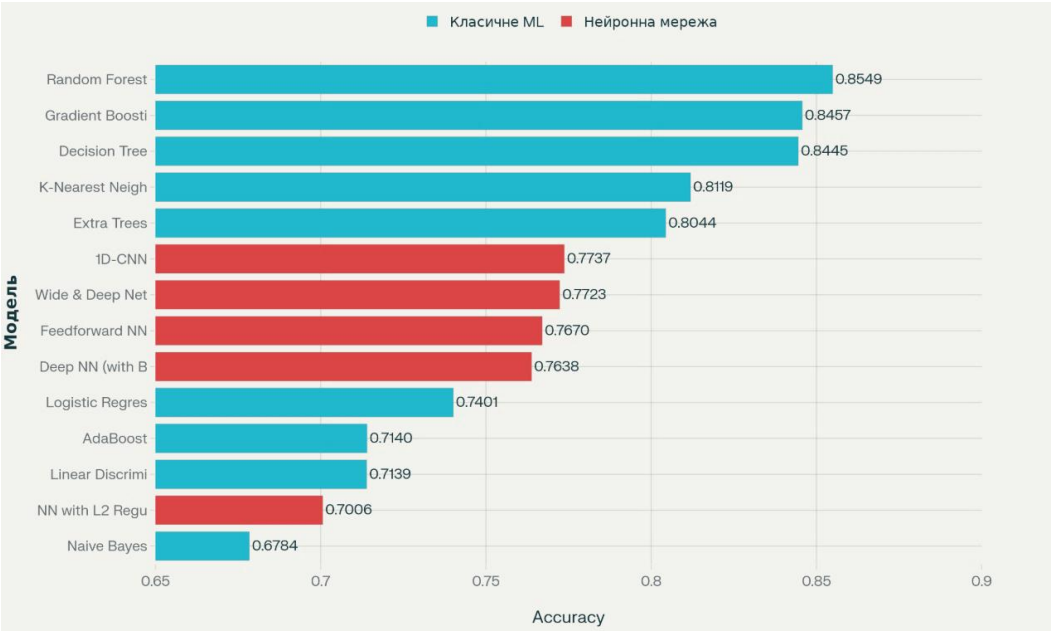

Рисунок 3.2 – Діаграма порівняння точності моделей

3.4.2 Аналіз результатів класифікації

Детальний аналіз результатів класифікації виявляє сильні та слабкі сторони кожного підходу. Матриця помилок для випадкового лісу зображена на рисунку 3.3 та демонструє патерни помилок.

Рисунок 3.3 – Матриця помилок для випадкового лісу

Клас доброякісних програм класифікується з найвищою точністю через найбільший розмір вибірки та відмітні ознаки.

Плутанина між класами троянів та програм-шпигунів є найбільш поширеною помилкою серед типів шкідливого ПЗ. Багато сучасних троянів включають функціональність програм-шпигунів (реєстрація клавіатури, захоплення екрану), що розмиває межі між категоріями. Програми-вимагачі мають найменше плутанини з іншими класами завдяки відмітному патерну інтенсивних операцій шифрування файлів.

Метрики для окремих класів розраховані окремо для кожного класу. Влучність для класу доброякісних програм найвища (близько 99%), оскільки модель рідко помилково класифікує шкідливе ПЗ як доброякісне. Повнота для доброякісних програм також висока - майже всі доброякісні зразки правильно ідентифіковані. Для класів шкідливого ПЗ влучність та повнота нижчі (70-80%) через менший розмір вибірки та більшу внутрішню різноманітність сімейств шкідливого ПЗ.

Аналіз важливості ознак з випадкового лісу виявляє ключові предиктори, які зображені на рисунку 3.4:

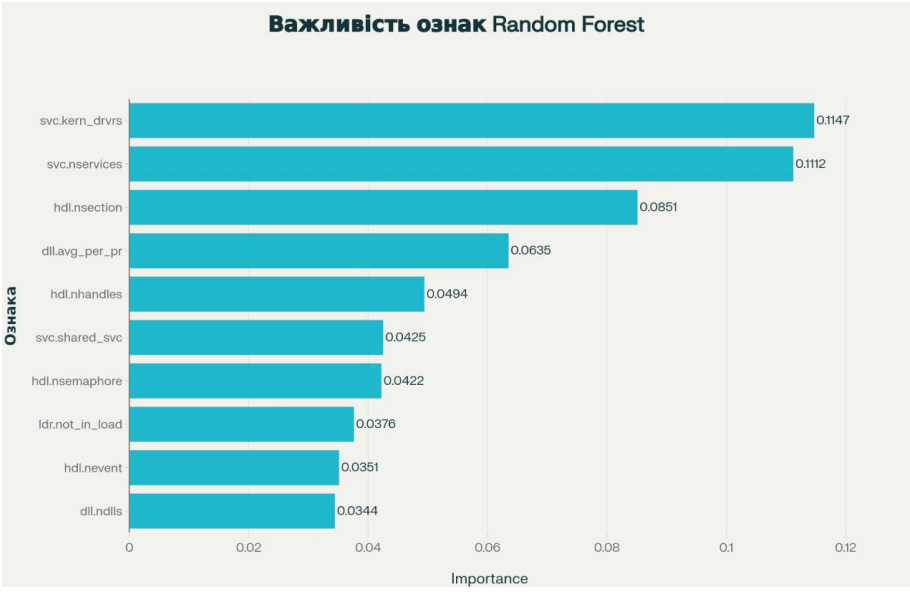

Рисунок 3.4 – Важливість ознак для Random Forest

svcscan.kernel_drivers – кількість завантажених драйверів ядра в системі, що відображає рівень взаємодії процесів із низькорівневими компонентами операційної системи, характерний для шкідливого програмного забезпечення. svcscan.nservices – загальна кількість сервісів Windows, пов'язаних з активними процесами, яка дає уявлення про масштаби використання служби Service Control Manager. handles.nsection – кількість дескрипторів типу Section (сегменти пам'яті), що вказує на активні операції з розділюваними областями пам'яті між процесами. dlllist.avg_dlls_per_proc – середня кількість завантажених DLL-модулів на процес, яка відображає складність і модульність процесів, притаманну складним зразкам шкідливого ПЗ. handles.nhandles – загальна кількість дескрипторів, відкритих процесами, що є інтегральною ознакою їхньої активності щодо ресурсів операційної системи. svcscan.shared_process_services – кількість сервісів, що працюють у спільних процесах, показує використання спільних служб і може бути індикатором прихованої шкідливої активності. handles.nsemaphore – кількість дескрипторів семафорів, яка характеризує механізми міжпроцесної синхронізації та складність потокової логіки програм. ldrmodules.not_in_load – кількість модулів, присутніх у пам'яті, але відсутніх у стандартних списках завантаження, що вказує на можливу інжекцію коду або нестандартне завантаження модулів. handles.nevent – кількість дескрипторів подій, що відображає використання процесами механізмів синхронізації подій у системі. dlllist.ndlls – загальна кількість DLL-модулів, знайдених у системі, яка є глобальною ознакою обсягу завантаженого виконуваного коду, потенційно використовуваного шкідливими процесами.

Порівняння з базовою бінарною класифікацією виявляє очікуване зниження точності. Бінарна задача (доброякісне проти шкідливе ПЗ) досягала точності понад 95% для більшості методів. Багатокласове розширення до чотирьох класів знизило точність до 85.49% для найкращої моделі. Це очікувано, оскільки розрізнення між троянами, програмами-шпигунами та

програмами-вимагачами складніше, ніж просте виявлення присутності шкідливого ПЗ.

Висновки за розділом 3

Проведено комплексне експериментальне дослідження методів машинного навчання для багатокласової класифікації обфускованого шкідливого ПЗ на основі аналізу дампів пам'яті.

Використано набір даних CIC-MalMem-2022 з 58,596 зразками дампів пам'яті, що включають чотири класи: доброякісні програми (50.0%), програми-шпигуни (17.1%), програми-вимагачі (16.7%) та трояни (16.2%). Набір даних розділено на навчальну (46,876) та тестову (11,720) вибірки зі стратифікованою вибіркою. Попередня обробка включала створення багатокласової цільової змінної, стандартизацію 54 числових ознак.

Реалізовано та порівняно чотирнадцять алгоритмів - дев'ять класичних методів машинного навчання та п’ять нейромережевих архітектур. Випадковий ліс досяг найвищої точності 85.49% при часі навчання 0.95 секунди, демонструючи оптимальний баланс продуктивності та швидкості. Градієнтний бустинг показав точність 84.57%, але вимагав 83.4 секунди навчання. Дерево рішень з точністю 84.45% та часом навчання 0.75 секунди виявився найшвидшим серед лідерів продуктивності.

Нейронні мережі систематично поступалися класичним методам машинного навчання з максимальною точністю 79.75% для нейронної мережі прямого поширення. Це узгоджується з відомими результатами для структурованих табличних даних, де методи на основі дерев історично переважають.

Аналіз патернів помилок виявив, що клас доброякісних програм класифікується з найвищою точністю. Основні помилки трапляються при розрізненні троянів та програм-шпигунів через перекриття функціональності. Програми-вимагачі мають відмітні патерни, що спрощує виявлення.

Порівняння з бінарною класифікацією підтвердило очікуване зниження точності при переході до багатокласової задачі. Збільшення кількості класів з двох до чотирьох знизило точність з 95%+ до 85.49% для найкращої моделі.

Практична значущість результатів полягає в демонстрації ефективності виявлення на основі пам'яті для обфускованого шкідливого ПЗ. Точність 85.49% є конкурентною порівняно з передовими методами на схожих наборах даних. Швидкість навчання (0.95с) та виведення дозволяє розгортання у виробничих середовищах.

ВИСНОВКИ

У даній роботі розв’язано задачу розробки та дослідження комп’ютерної моделі виявлення шкідливого програмного забезпечення на основі аналізу дампів оперативної пам’яті із застосуванням методів машинного та глибокого навчання. На основі огляду сучасних типів шкідливого ПЗ, тенденцій його розвитку та аналізу існуючих підходів показано, що традиційні сигнатурні та файлові методи виявлення втрачають ефективність в умовах обфускації, безфайлових та гібридних атак, тоді як аналіз пам’яті забезпечує доступ до розпакованого коду, конфігурацій, ключів шифрування й поведінкових артефактів.

Було досліджено теоретичні основи пам’яттєвої форензіки, структуру дампів пам’яті, типові ознаки зловмисної активності та інструменти їх аналізу, зокрема інструменти, що дозволяє автоматизовано виділяти процеси, модулі, дескриптори та мережеві сесії. Показано, що саме на основі таких структурованих артефактів можна сформувати багатовимірний простір ознак, придатний для подальшого застосування моделей машинного й глибокого навчання.

У роботі систематизовано методи машинного та глибокого навчання, розглянуто класичні алгоритми навчання з учителем, ансамблеві моделі, базові архітектури нейронних мереж та їх застосування для виявлення шкідливого ПЗ. Окрему увагу приділено постановці задачі бінарної й багатокласової класифікації, вибору метрик якості та інтерпретації

Експериментальні результати показали, що ансамблеві моделі на кшталт Random Forest забезпечують високу точність і збалансовані показники повноти та F1-міри при відносно невеликих обчислювальних витратах, що робить їх доцільними для практичного використання в оперативних системах виявлення загроз.

Аналіз важливості ознак для найкращих моделей показав, що значний внесок у якість класифікації забезпечують параметри, пов’язані з кількістю та типами системних сервісів, драйверів ядра, дескрипторів секцій і подій, а також характеристиками завантажених DLL-модулів. Це підтверджує доцільність використання пам’яттєвих артефактів як основи для виявлення різних класів шкідливого ПЗ та відкриває перспективи для подальшої оптимізації моделей шляхом відбору ознак.

Практична значущість отриманих результатів полягає в тому, що розроблені та досліджені моделі можуть бути інтегровані в системи моніторингу та реагування, а також використані як основа для побудови промислових рішень, орієнтованих на аналіз дампів оперативної пам’яті. Запропонований підхід демонструє можливість досягнення конкурентних показників точності за прийнятного часу навчання й виведення, що є критичним для реальних умов експлуатації.

Подальші напрями досліджень включають розширення набору даних за рахунок дампів із реальних інцидентів, адаптацію моделей до концептуального дрейфу даних та інтеграцію запропонованої комп’ютерної моделі з існуючими інструментами цифрової криміналістики та платформами безпеки. Це дозволить підвищити стійкість систем до нових типів атак і забезпечити довготривалу актуальність розроблених рішень у динамічному ландшафті кіберзагроз.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aslan Ö., Samet R. A comprehensive review on malware detection approaches. *IEEE access*. 2020. Vol. 8. P. 6249–6271. URL: https://ieeexplore.ieee.org/document/8949524.

2. Bishop C. M. Pattern recognition and machine learning. New York : Springer, 2006. 738 p.

3. Canadian Institute for Cybersecurity. Malware memory analysis. URL: https://www.unb.ca/cic/datasets/malmem-2022.html.

4. Caron C., Others. Malware classification from memory dumps using machine learning and transformers. *ArXiv preprint*. 2024. URL: https://arxiv.org/abs/2503.02144.

5. Carrier T., Tekeoglu A., Lashkari A. H. Detecting obfuscated malware using memory feature engineering. *Proceedings of the 8th international conference on information systems security and privacy (ICISSP)*. 2022. P. 177–188.

6. Case A., Richard G. G. Memory forensics: the path forward. *Digital investigation*. 2017. Vol. 20. P. 23–33.

7. Catak F. O., Others. Deep learning based Sequential model for malware analysis using Windows exe API Calls. *PeerJ computer science*. 2020. Vol. 6.

8. Chen L., Others. Deep learning for malware classification: a survey. *ArXiv preprint*. 2024.

9. Dai Y., Others. A malware classification method based on memory dump grayscale image. *Digital investigation*. 2018. Vol. 27. P. 30–37.

10. Foundation T. V. Volatility 3 framework. URL: https://github.com/volatilityfoundation/volatility3.

11. Géron A. Hands-On machine learning with scikit-learn, keras, and tensorflow: concepts, tools, and techniques to build intelligent systems. O'Reilly Media, Incorporated, 2022. 483 p.

12. Gibert D., Mateu C., Planes J. The rise of machine learning for detection and classification of malware: research developments, trends and challenges. *Journal of network and computer applications*. 2020. Vol. 153. P. Article 102526. URL: https://doi.org/10.1016/j.jnca.2019.102526.

13. Goodfellow I., Bengio Y., Courville A. Deep learning. Cambridge : MIT Press, 2016. 800 p.

14. Grandini M., Bagli E., Visani G. Metrics for multi-class classification: an overview. *ArXiv preprint*. 2020. URL: https://arxiv.org/abs/2008.05756.

15. IMCFN: image-based malware classification using fine-tuned convolutional neural network architecture / D. Vasan et al. *Computer networks*. 2020. Vol. 171. P. 107138.

16. Kumar R., Others. Comparison of machine learning algorithms for malware detection using edge-iiotset dataset in iot. *International journal of advanced computer science and applications*. 2025. Vol. 16, no. 1.

17. Kumar R., Subbiah K. Effective malware detection using xgboost and lightgbm. *Proceedings of the international conference on electronics and sustainable communication systems (ICESC)*. Coimbatore, India, 2021. P. 1–6.

18. Malware detection using memory analysis data in big data environment / U. Urooj et al. *IEEE access*. 2022. Vol. 10. P. 1–16.

19. Or-Meir O., Others. Dynamic malware analysis in the modern era: a state of the art survey. *ACM computing surveys*. 2019. Vol. 52, no. 5. P. 1–48.

20. Raff E., Others. Malware detection by eating a whole exe. *Workshops at the thirty-second AAAI conference on artificial intelligence*. 2018.

21. Review S. Malware detection and classification in cybersecurity. *Applied sciences*. 2024. Vol. 15, no. 14. P. 7747. URL: https://www.mdpi.com/2076-3417/15/14/7747.

22. Saxe J., Sanders H. Malware data science: attack detection and attribution. San Francisco : No Starch Press, 2018. 272 p.

23. Sihwail R., Omar K., Ariffin K. A survey on malware analysis techniques: static, dynamic, hybrid and memory analysis. *Asian journal of information technology*. 2018. Vol. 17. P. 1–9.

24. Sikorski M., Honig A. Practical malware analysis: the hands-on guide to dissecting malicious software. San Francisco : No Starch Press, 2012. 800 p.

25. Souppaya M., Scarfone K. Guide to malware incident prevention and handling for desktops and laptops. Gaithersburg : National Institute of Standards and Technology, 2013. 47 p.

26. The art of memory forensics: detecting malware and threats in windows, linux, and mac memory / M. H. Ligh et al. Indianapolis : Wiley, 2014. 912 p.

27. Wueest C. Living off the land and fileless attack techniques. Mountain View : Symantec Corporation, 2017. 21 p.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) ***Магістр***
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

ЛАНІНА Євгена Сергійовича
(прізвище, ім'я, по батькові студента)

1. Тема роботи **«КОМП'ЮТЕРНА МОДЕЛЬ ВИЯВЛЕННЯ ЗЛОВМИСНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ПІДСТАВІ АНАЛІЗУ ДАМПУ ПАМ'ЯТІ»**

керівник роботи **БАКУМЕНКО Ніна Станіславівна, кандидат технічних наук, доцент.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету ***від 30 вересня 2025 року № 4101-5/3554***

2. Строк подання студентом роботи ***30 листопада 2025 року***

3. Перелік питань, які потрібно розробити)
1. Аналіз сучасних методів виявлення зловмисного програмного забезпечення.
2. Дослідження методів аналізу дампів оперативної пам'яті.
3. Огляд існуючих інструментів для аналізу пам'яті.
4. Вивчення методів машинного навчання для класифікації зловмисного ПЗ.
5. Аналіз датасетів та їх характеристик.
6. Порівняльний аналіз класичних алгоритмів машинного навчання для виявлення malware.
7. Дослідження архітектур нейронних мереж для багатокласової класифікації загроз.
8. Розробка комп'ютерної моделі виявлення зловмисного ПЗ.
9. Реалізація системи попередньої обробки та нормалізації даних з дампів пам'яті.
10. Тестування та валідація розробленої моделі на тестових даних.
11. Оцінка ефективності моделі за метриками.
12. Аналіз часу навчання та продуктивності різних алгоритмів.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми роботи та керівника.	02.09.2024 - 02.10.2024
2	Аналіз та пошук методичної літератури щодо виявлення зловмисного ПЗ та методів аналізу дампів пам'яті.	03.09.2024 - 24.10.2024
3	Огляд і аналіз існуючих інструментів для аналізу пам'яті та методів виявлення malware.	25.10.2024 - 09.11.2024
4	Вибір та характеристика набору даних для експериментальних досліджень.	10.11.2024 - 24.11.2024
5	Розробка математичної моделі та архітектури системи виявлення зловмисного ПЗ.	25.11.2024 - 08.12.2024
6	Реалізація та навчання класичних алгоритмів машинного навчання.	09.12.2024 - 29.01.2025
7	Розробка та навчання архітектур нейронних мереж.	30.01.2025 - 28.02.2025
8	Тестування, валідація та порівняльний аналіз ефективності алгоритмів.	01.03.2025 - 01.04.2025
9	Оптимізація обраної моделі та дослідження можливостей практичного впровадження.	02.04.2025 - 30.04.2025
10	Підготовка і оформлення звітних матеріалів.	01.05.2025 - 30.08.2025
11	Оформлення звіту про науково-дослідну практику. Написання статті за матеріалами кваліфікаційної роботи.	01.09.2025 - 30.10.2025
12	Підготовка і оформлення звітних матеріалів кваліфікаційної роботи. Оформлення списку літератури.	10.10.2025 - 30.10.2025
13	Оформлення пояснювальної записки кваліфікаційної роботи відповідно вимогам до звітів про НДР.	10.10.2025 - 30.11.2025
14	Підготовка і оформлення звітних матеріалів та додатків кваліфікаційної роботи.	01.11.2025 - 30.11.2025
15	Оформлення звіту про переддипломну практику.	24.11.2025 - 30.11.2025
16	Представлення кваліфікаційної роботи керівнику та рецензенту.	24.11.2025 - 30.11.2025

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент **Є. С. Ланін** _____
ініціали, прізвище підпис

Керівник роботи **Н. С. Бакуменко** _____
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«_____»_____2025 р.

Технічне завдання
на розробку програмного виробу «Комп'ютерна модель виявлення зловмисного програмного забезпечення на підставі аналізу дампу пам'яті»

1.	Введення	1.1. Назва: Метод аналізу інформативності змінних стану при діагностиці систем з використанням інформаційних критеріїв. 1.2. Галузь застосування: інформаційні технології, штучний інтелект.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 123 – Комп’ютерна інженерія 2.2. Завдання на кваліфікаційну роботу магістра № 4101-5/3554 від «30» вересня 2025 (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3.	Призначення розробки	3.1. Мета розробки: розробка та експериментальне дослідження комп’ютерної моделі виявлення шкідливого програмного забезпечення на основі аналізу дампів оперативної пам’яті з використанням методів машинного навчання.. 3.2. Призначення розробки надання можливості аналітикам безпеки та автоматизованим системам виявляти приховані загрози, які не детестуються традиційними антивірусними засобами, шляхом аналізу артефактів у RAM, підвищення точності класифікації типів загроз та зменшення часу реагування на інциденти. 3.3. Вихідні дані розробки: статистичні експериментальні дані — набір даних CIC-MalMem-2022 (Obfuscated-MalMem2022)
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: Зчитування та попередня обробка набору даних дампів пам'яті (формат CSV), проведення розвідувального аналізу даних та кореляційного аналізу ознак, реалізація конвеєра машинного навчання: стандартизація даних, поділ на вибірки, навчання моделей класифікації, багатокласова класифікація

		зразків на 4 класи: Benign, Spyware, Ransomware, Trojan, розрахунок метрик ефективності: Accuracy, Precision, Recall, F1-score, матриця помилок. 4.2. Вимоги до надійності: забезпечувати точність класифікації (Accuracy) не нижче 75% на тестовій вибірці, забезпечувати стійкість до дисбалансу класів шкідливого ПЗ. 4.3.Вимоги до умов експлуатації: наявність встановленого інтерпретатора Python 3.8+ та необхідних бібліотек (scikit-learn, pandas, numpy, tensorflow). 4.4. Вимоги до складу і параметрів технічних засобів: персональний комп'ютер з процесором не нижче Intel Core i5 / AMD Ryzen 5, оперативна пам'ять: не менше 8 ГБ (рекомендовано 16 ГБ для швидкої обробки dataset), вільне місце на диску: не менше 1 ГБ. 4.5. Вимоги до інформаційної та програмної сумісності: сумісність з ОС Windows 10/11, Linux (Ubuntu 20.04+), MacOS. Вхідні дані у форматі структурованих таблиць (CSV). 4.6. Вимоги до маркування та упаковки: немає 4.7. Вимоги до транспортування і зберігання: на звичайних носіях інформації (Flash-накопичувачі, хмарні сховища, репозиторії GitHub). 4.8. Спеціальні вимоги: результати експериментів мають бути відтворюваними.
5.	Вимоги до програмної документації	Програмною документацією до виробу «Метод аналізу інформативності змінних стану при діагностиці систем з використанням інформаційних критеріїв» вважати: 1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Методику побудови моделей класифікації та аналізу ознак (у вигляді розділів 3.1–3.3 пояснювальної записки до кваліфікаційної роботи). 3) Опис результатів експериментальних досліджень та програмної реалізації (представити в розділі 3.4 пояснювальної записки до кваліфікаційної роботи).
6.	Вимоги до техніко-	Програмною документацією до виробу «Комп'ютерна модель виявлення зловмисного програмного забезпечення на підставі аналізу дампу пам'яті» вважати:

	економічних показників	1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Методику розрахунку інформативності змінних стану (у вигляді глав 3.3 та 3.4 пояснювальної записки до кваліфікаційної роботи). 3) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)	
7.	Стадії і етапи розробки	Дата	Назва етапу
		від 1 жовтня 2025 до 3 жовтня 2025	Аналіз наукових джерел і методичних матеріалів щодо методів виявлення шкідливого ПЗ та аналізу дампів пам'яті.
		від 3 жовтня 2025 до 5 жовтня 2025	Вивчення існуючих підходів до класифікації шкідливого ПЗ на основі машинного навчання.
		від 5 жовтня 2025 до 8 жовтня 2025	Вибір інструментів і бібліотек для реалізації моделей класифікації (Python, Scikit-learn, Volatility).
		від 8 жовтня 2025 до 17 жовтня 2025	Розробка загальної архітектури комп'ютерної моделі та алгоритму обробки даних.
		від 17 жовтня 2025 до 25 жовтня 2015	Реалізація основного функціоналу: завантаження, очищення та попередня обробка набору даних.

		від 25 жовтня 2025 до 5 листопада 2025	Навчання та налаштування моделей машинного навчання для класифікації загроз.
		від 5 листопада 2025 до 7 листопада 2025	Проведення тестування розроблених моделей на тестовій вибірці та усунення недоліків в алгоритмах.
		від 7 листопада 2025 до 8 листопада 2025	Оцінка точності класифікації, аналіз матриць помилок та ефективності роботи системи
		від 11 листопада 2025 до 15 листопада 2025	Оформлення звіту про науково-дослідну практику.
		від 15 листопада 2025 до 18 листопада 2025	Написання статті за матеріалами кваліфікаційної роботи.
		від 18 листопада 2025 до 25 листопада 2025	Підготовка і оформлення звітних матеріалів кваліфікаційної роботи.
		від 25 листопада 2025 до 27 листопада 2025	Оформлення списку літератури.
		від 27 листопада 2025 до 29 листопада 2025	Оформлення пояснювальної записки кваліфікаційної

		від 29 листопада 2025 до 30 листопада 2025	роботи відповідно вимогам до звітів про НДР. Підготовка і оформлення звітних матеріалів та додатків кваліфікаційної роботи. Оформлення звіту про переддипломну практику.
8.	Порядок контролю і приймання програмного продукту (моделі)	1. Перевірку ходу розробки програми виконувати раз в 3 тижні. 2. Захист розробленої моделі провести на засіданні Атестаційної комісії. 3. Пояснювальну записку подати на паперових носіях в 1 примірнику і в електронному вигляді в 1 примірнику.	

Виконавець
студент групи КІ- 61
Ланін Є. С.

Замовник
д. т. н., доцент.
Бакуменко Н. С.

Додаток В

Програма і методика випробувань програмного виробу

«Комп'ютерна модель виявлення зловмисного програмного забезпечення на підставі аналізу дампу пам'яті»

1. Об’єкт випробувань

1.1 Назва розробленого прототипу: «Комп'ютерна модель виявлення зловмисного програмного забезпечення на підставі аналізу дампу пам'яті».

1.2 Галузь застосування: Інформаційні технології , комп’ютерні системи, кібербезпека.

1.3 Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональні можливості системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

3.1 Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2 Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп’ютерного класу кафедри в період роботи атестаційної комісії.

3.3 Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі відповідному цієї програми і методики випробувань.

3.4 Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавці та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

4.1. Функціональні вимоги:

- завантаження та попередню обробку набору даних (дампів пам'яті CIC-MalMem-2022);
- моделювання та багатокласову класифікацію загроз (Benign, Spyware, Ransomware, Trojan);
- розрахунок метрик ефективності (Accuracy, Precision, Recall, F1-score);
- генерацію матриці помилок для візуалізації результатів.

4.2. Нефункціональні вимоги:

- висока доступність системи для забезпечення безперебійної роботи під час експериментів;
- висока продуктивність системи з можливістю ефективної обробки великих обсягів даних (понад 50 000 записів);
- простота в управлінні та можливість відтворення результатів (reproducibility).

4.3. Вимоги до інтеграції:

- підтримка стандартних форматів даних (CSV);
- використання відкритих бібліотек машинного навчання (Scikit-learn, Pandas);
- можливість запуску в середовищі Jupyter Notebook або через командний рядок.

4.4. Вимоги до безпеки:

- цілісність вхідних даних (захист від модифікації dataset);
- коректна обробка виключних ситуацій (відсутність файлів, невірний формат даних);
- відсутність шкідливого впливу на операційну систему, де проводяться випробування.

5. Вимоги до програмної документації

Програмною документацією до прототипу «Комп'ютерна модель виявлення зловмисного програмного забезпечення на підставі аналізу дампу пам'яті» вважати:

1) Справжнє Технічне завдання на розробку прототипу мережі (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).

2) Опис реалізованого прототипу мережі (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).

3) Джерела базової інформації.

6. Засоби і порядок випробувань

6.1. Засоби випробувань

Операційна система: Windows 10/11 або Linux. Мова програмування: Python 3.8+. Середовище розробки: PyCharm, VS Code або Jupyter Lab. Бібліотеки: Pandas, NumPy, Scikit-learn, Matplotlib, Seaborn.

6.2. Порядок проведення випробувань

- **Перший етап:**
  Перевірка комплектності та якості програмної документації відповідно до ГОСТ 34.602-89.

- **Другий етап:**

- Запуск програмного скрипту та завантаження датасету CIC-MalMem-2022.
- Виконання процедури навчання моделі.
- Оцінка точності моделі класифікації шкідливого ПЗ на тестовій вибірці.
- Перевірка швидкодії навчання та класифікації.

7. Проведення випробувань

7.1 Завантаження та аналіз даних:
Під час випробувань система має вміти правильно завантажувати файл з ознаками дампів пам'яті та виводити інформацію про структуру даних.

```
=== DATA PREPARATION ===

Dataset size: (58596, 55)
Classes: ['Benign' 'Ransomware' 'Spyware' 'Trojan']
Class distribution:
MalwareType
Benign        29298
Spyware       10020
Ransomware     9791
Trojan         9487
Name: count, dtype: int64

Training samples: 46876
Test samples: 11720
```

Рисунок В.1 – Результат правильного завантаження даних

7.2 Процес навчання моделей:
Під час випробувань система має вміти безпомилково проводити навчання моделей класифікації.

```
================================================================================
TRAINING CLASSICAL ML ALGORITHMS
================================================================================

Training: Logistic Regression...
  ✓ Accuracy: 0.7401 | F1: 0.7396 | Time: 19.17s

Training: Decision Tree...
  ✓ Accuracy: 0.8445 | F1: 0.8449 | Time: 1.80s

Training: Random Forest...
  ✓ Accuracy: 0.8549 | F1: 0.8552 | Time: 1.52s

Training: Extra Trees...
  ✓ Accuracy: 0.8044 | F1: 0.8046 | Time: 0.69s

Training: Gradient Boosting...
  ✓ Accuracy: 0.8457 | F1: 0.8458 | Time: 183.95s

Training: AdaBoost...
  ✓ Accuracy: 0.7140 | F1: 0.6968 | Time: 12.30s

Training: K-Nearest Neighbors...
  ✓ Accuracy: 0.8119 | F1: 0.8117 | Time: 5.02s

Training: Naive Bayes...
  ✓ Accuracy: 0.6784 | F1: 0.6328 | Time: 0.11s

Training: Linear Discriminant Analysis...
  ✓ Accuracy: 0.7139 | F1: 0.7099 | Time: 0.34s
```

Рисунок В.2 – Успішне навчання моделей

7.3 Результати випробувань:

Здійснюється класифікація типів шкідливого ПЗ за допомогою навчених моделей. Результати перевіряються на відповідність критерію точності (Accuracy > 75%). Будується матриця помилок для візуалізації якості розпізнавання класів для найкращою моделі.

	C1	C2	C3	C4	C5	C6	C7
1	Algorithm	Type	Accuracy	Precision	Recall	F1-Score	Training Time (s)
2	Random Forest	Classical ML	0.8549	0.8558	0.8549	0.8552	1.52
3	Gradient Boosting	Classical ML	0.8457	0.8463	0.8457	0.8458	183.95
4	Decision Tree	Classical ML	0.8445	0.8458	0.8445	0.8449	1.8
5	K-Nearest Neighbors	Classical ML	0.8119	0.8131	0.8119	0.8117	5.02
6	Extra Trees	Classical ML	0.8044	0.8117	0.8044	0.8046	0.69
7	Feedforward NN	Neural Network	0.7976	0.8004	0.7976	0.7975	53.98
8	CNN	Neural Network	0.7769	0.7824	0.7769	0.7747	133.94
9	Wide & Deep Network	Neural Network	0.7728	0.7812	0.7728	0.7696	23.3
10	Deep NN (with BatchNorm)	Neural Network	0.7637	0.7778	0.7637	0.7558	40.02
11	Logistic Regression	Classical ML	0.7401	0.7459	0.7401	0.7396	19.17
12	Linear Discriminant Analysis	Classical ML	0.7139	0.7199	0.7139	0.7099	0.34
13	AdaBoost	Classical ML	0.714	0.7464	0.714	0.6968	12.3
14	NN with L2 Regularization	Neural Network	0.7194	0.746	0.7194	0.692	22.82
15	Naive Bayes	Classical ML	0.6784	0.7266	0.6784	0.6328	0.11

Рисунок В.3 – Виведення таблиці результатів класифікації

Рисунок В.3 – Виведення матриця помилок для найкращою моделі задля оцінки прогнозування моделі

Виконавець: студент групи КІ-61, Ланін Є.С. _____