Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Модель федеративного навчання для виявлення вторгнень в IoT

(тема)

Виконав:

здобувач 2 року навчання,

групи СПм-23-3

Владислав РИКОВ
(власне ім'я, прізвище)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: доцент Олексій ЛЯШЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Рикову Владиславу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Модель федеративного навчання для виявлення вторгнень в IoT

затверджена наказом по університету від "21" квітня 2025 р. № 296 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 2025 р.

3. Вхідні дані до роботи
глибоке навчання, кластеризація, атака DDOS, нейронна мережа LSTM
інтернет речей, система виявлення вторгнень, федеративне навчання,

4. Перелік питань, що потрібно опрацювати у роботі
Теоретичні основи дослідження
Модель IDS
Побудова системи та експеременти
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій 14

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання теми кваліфікаційної роботи	21.04	
2	Аналіз літератури	22.04-1.05	
3	Побудова моделі	2.05-15.05	
4	Тестування системи та отримання результатів	16.05-30.05	
5	Формування пояснювальної записки	31.05-04.06	
6	Перевірка на плагіат	05.06-06.06	
7	Рецензування роботи	07.06-11.06	
8	Подача роботи в ЕК	12.06	
9	Захист роботи		

Дата видачі завдання " 21 " квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доцент Олексій ЛЯШЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 73 с., 10 рис., 10 табл., 2 дод., 28 джерел.

ІНТЕРНЕТ РЕЧЕЙ, СИСТЕМА ВИЯВЛЕННЯ ВТОРГНЕНЬ, ФЕДЕРАТИВНЕ НАВЧАННЯ, ГЛИБОКЕ НАВЧАННЯ, КЛАСТЕРИЗАЦІЯ, АТАКА DDOS, НЕЙРОННА МЕРЕЖА LSTM.

Метою кваліфікаційної роботи є розробка моделі федеративного навчання для виявлення вторгнень в IoT.

У ході виконання кваліфікаційної роботи проведено аналіз архітектур федеративного навчання в системах виявлення вторгнень. Запропоновано напівдецентралізовану модель FL. Створено програмну реалізацію та протестувано запропоновану модель на наборі даних CICIOT2023.

ABSTRACT

Master’s thesis: 73 pages, 10 figures, 10 tables, 2 appendices, 28 sources.

INTERNET OF THINGS, INTRUSION DETECTION SYSTEM, FEDERATED LEARNING, DEEP LEARNING, CLUSTERING, DDOS ATTACK, LSTM NEURAL NETWORK.

The major goal of this thesis is to develop a federated learning model for intrusion detection in IoT.

During the qualification work, an analysis of federated learning architectures in intrusion detection systems was conducted. A semi-decentralized FL model was proposed. A software implementation was created and the proposed model was tested on the CICIOT2023 dataset.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

IoT – Інтернет речей (Internet of Things)

IDS – система виявлення вторгнень (Intrusion Detection System)

ML – машинне навчання (Machine Learning)

DL – глибоке навчання (Deep Learning)

FL – федеративне навчання (Federated Learning)

DDoS – розподілена відмова в обслуговуванні (Distributed Denial of Service)

LSTM – мережа довготривалої та короткочасної пам’яті (Long Short-Term Memory)

BiLSTM – двонапрямлена LSTM (Bidirectional Long Short-Term Memory)

WGAN – вдосконалена генеративна змагальна мережа з ваговою нормалізацією (Wasserstein Generative Adversarial Network)

CICIoT2023 – набір даних атак в IoT-середовищі

ВСТУП

Широкий діапазон і неоднорідність мереж Інтернету речей (IoT) роблять їх схильними до кібератак. Більшість пристроїв IoT мають обмежені ресурси (наприклад, ємність пам’яті, потужність обробки та споживання енергії), щоб функціонувати як звичайні системи виявлення вторгнень (IDS). Дослідники застосували багато підходів до легких IDS, включаючи IDS на основі енергії, IDS на основі машинного/глибокого навчання (ML/DL) і IDS на основі федеративного навчання (FL). FL став багатообіцяючим рішенням для IDS в мережах IoT, оскільки він зменшує накладні витрати в процесі навчання, залучаючи пристрої IoT під час процесу навчання. Для боротьби з IDS у мережах IoT використовуються три архітектури FL, включаючи централізовану (клієнт-сервер), децентралізовану (від пристрою до пристрою) і напівдецентралізовану. Однак жоден із них не вирішив гетерогенність пристроїв IoT, враховуючи при цьому легкість і продуктивність. Запропоновано напівдецентралізовану модель на основі FL для легкої IDS, яка відповідає можливостям пристроїв IoT. Запропонована модель базується на кластеризації пристроїв IoT – клієнтів FL – призначенні голови кластера кожному кластеру, який діє від імені клієнтів FL. Отже, кількість пристроїв IoT, які спілкуються із сервером, зменшується, що допомагає зменшити накладні витрати на зв’язок. Крім того, кластеризація допомагає покращити процес агрегації, оскільки кожен кластер надсилає середні ваги моделі на сервер для агрегації в одному раунді FL. Розподілена атака типу «відмова в обслуговуванні» (DDoS) є основною проблемою в нашій моделі IDS, оскільки вона легко виникає в пристроях IoT з обмеженими можливостями ресурсів. Запропонована модель налаштована за допомогою трьох методів глибокого навчання – LSTM, BiLSTM і WGAN – із використанням набору даних CICIoT2023. Експериментальні результати показують, що BiLSTM забезпечує кращу продуктивність і підходить для

пристроїв IoT з обмеженими ресурсами залежно від розміру моделі. В роботі тестуємо попередньо підготовлену напівдецентралізовану модель на основі FL на трьох наборах даних – BoT-IoT, WUSTL-IIoT-2021 і Edge-IIoTset – і результати показують, що наша модель має найвищу продуктивність у більшості класів, особливо для атак DDoS.

1 ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

1.1 Теоретичні аспекти побудови моделі

Оскільки кількість пристроїв IoT значно зростає, необхідність збереження безпеки та конфіденційності даних, що передаються, стала складним завданням. Багато механізмів використовувалися для захисту мереж IoT, наприклад, брандмауери та механізми контролю доступу, але ці методи забезпечують лише конфіденційність і автентичність даних у мережі пристроїв IoT. Таким чином, мережі IoT потребують механізмів безпеки, які можуть служити лінією захисту для виявлення зловмисників. Для виявлення зловмисної поведінки необхідно застосувати систему виявлення вторгнень (IDS). Два поширені підходи до систем виявлення вторгнень (IDS) – це системи виявлення вторгнень на основі сигнатур (SIDS) і системи виявлення вторгнень на основі аномалій (AIDS). SIDS, тобто «виявлення на основі евристики чи неправильного використання», визначає набір відомих шкідливих шаблонів даних (сигнатур) або правил атак (евристик), які використовуються для виявлення атак. Однак цей підхід може ідентифікувати лише відомі атаки, для яких він зберіг шаблони або правила для порівняння. З іншого боку, SIDS, тобто «IDS на основі поведінки», передбачає розробку профілів нормальної поведінки користувачів протягом певного періоду часу, щоб будь-яке відхилення від цього профілю розглядалося як аномалія або ненормальна поведінка [1].

Більшість пристроїв IoT мають обмежені ресурси для виконання звичайних завдань IDS. Тому були запропоновані полегшені підходи IDS, щоб відповідати можливостям пристроїв IoT. По-перше, енергетичні IDS базуються на виявленні атак за допомогою споживання енергії як індикатора конкретних типів атак. Однак не всі типи атак можна виявити за допомогою цього методу, оскільки багато атак не впливають на споживання енергії

(наприклад, атаки Sybil або маніпулювання даними) [2,3]. По-друге, моделі машинного та глибокого навчання (ML/DL) використовуються для аналізу даних та їхніх залежностей, щоб допомогти розпізнати шаблони атак [4]. Крім того, IDS на основі ML/DL використовуються як легкі IDS, витягуючи найбільш відповідні функції для виявлення атак. Однак, оскільки кількість взаємопов'язаних пристроїв IoT різко зростає та генерує величезну кількість даних, моделі ML/DL важко розгорнути в мережах IoT з обмеженими ресурсами [5,6].

Одна модель ML, відома як федеративне навчання (FL), розподілена модель ML, яка дозволяє пристроям IoT спільно вивчати спільну модель ML без розкриття локальних даних. Таким чином, впровадження FL в IDS допомагає зменшити обчислювальне навантаження на сервери центральної обробки, зберігаючи конфіденційність даних. Зазвичай використовуються три архітектури FL: централізована FL (клієнт-сервер); децентралізований FL (від пристрою до пристрою); і напівдецентралізовані FL. У централізованій архітектурі FL кожен пристрій навчає локальну модель на власних даних, а потім її параметри/ваги агрегуються для створення глобальної моделі, керованої центральним об'єктом (тобто сервером або координатором) [7]. Незважаючи на те, що централізований FL покращує ефективність процесу навчання, він має деякі недоліки, а саме відсутність масштабованості та підключення та вразливість до єдиної точки відмови. Децентралізований FL долає ці проблеми, дозволяючи пристроям співпрацювати в процесі навчання без координатора. Однак процес агрегування є неефективним, оскільки він не проводиться під наглядом одного пристрою, що призводить до низької точності [8, 9]. Напівдецентралізований FL досягає балансу за допомогою кластеризації пристроїв, причому кожен кластер має локальний координатор для полегшення навчання та агрегації, що призводить до швидшої конвергенції та покращеної масштабованості [10,11].

У всіх архітектурах FL неоднорідність є основною проблемою, яка перешкоджає продуктивності FL [12], оскільки неоднорідність сильно

впливає на процес навчання, що призводить до розбіжності моделі [13, 14]. Обмежений характер пристроїв IoT вимагає легких моделей. Крім того, більшість сучасних підходів не досягають балансу між високою продуктивністю та ефективним використанням ресурсів.

У цій роботі представляємо напівдецентралізовану модель FL для виявлення вторгнень у гетерогенну мережу IoT. Основний внесок нашої роботи викладено нижче:

Запропонована напівдецентралізована модель FL кластеризує клієнтів FL, щоб усунути вплив неоднорідності в процесі навчання FL і зменшити споживання ресурсів.

Підхід до кластеризації в запропонованій моделі допомагає покращити продуктивність FedAvg, алгоритму агрегації, оскільки кожен кластер надсилає середні ваги моделі на сервер для агрегації в одному раунді FL.

Досліджується три методи DL, LSTM, BiLSTM і WGAN, як локальні моделі в напівдецентралізованій архітектурі FL. Наскільки нам відомо, ми перші застосували WGAN з одним генератором для всіх клієнтів FL, щоб навчати їх на однакових згенерованих даних, оскільки наша мета – виявити вторгнення, а не відрізнити справжні дані від згенерованих. Кожен клієнт FL має модель дискримінатора для навчання на його локальних даних.

Напівдецентралізована модель FL із BiLSTM забезпечує баланс між продуктивністю та легкістю, щоб відповідати можливостям IoT, оскільки вона налаштована з параметрами низької складності (кількість шарів і нейронів).

1.2 IDS

Оскільки пристрої IoT мають обмежені ресурсні можливості, необхідно враховувати це обмеження при розробці IDS для IoT. Було запропоновано багато підходів для розробки легких IDS, а саме IDS на основі енергії, IDS на основі ML/DL та IDS на основі федеративного навчання.

1.2.1 Легкі IDS на основі енергії

Оскільки звичайні IDS не застосовуються для пристроїв IoT з обмеженими ресурсами, багато досліджень зосереджено на мінімізації енергоспоживання за допомогою легких IDS. В [2] та [3] запропонував енергетичний підхід, який акцентує увагу на аналізі споживання енергії для виявлення атак шляхом оцінки очікуваного споживання енергії на основі минулих спостережень для кожного пристрою. Таким чином, якщо споживання енергії даним пристроєм відхиляється від очікуваного значення, це розглядається як аномалія. На жаль, цей підхід не може виявити всі типи атак, лише ті, що впливають на споживання енергії, наприклад, різні типи DoS-атак (флуд, чорні діри тощо).

1.2.2 IDS на основі легкого машинного/поглибленого навчання

Методи машинного та глибокого навчання (ML/DL) зазвичай використовуються в IDS для аналізу даних та їхніх залежностей як способу вивчення шаблонів атак [2]. У ML розробка функцій виконується для вилучення відповідних функцій для ідентифікації та класифікації атак. Багато алгоритмів ML використовуються в IDS, наприклад, дерево рішень (DS), опорна векторна машина (SVM), логістична регресія (LR), наївний Байєс (NB) і штучна нейронна мережа (ANN). Однак алгоритми DL можуть автоматично отримувати функції високого рівня за допомогою кількох нелінійних рівнів обробки. Згідно з дослідженнями [5] та [6], моделі DL, які найчастіше використовуються в IDS, включають згорткову нейронну мережу (CNN), рекурентну нейронну мережу (RNN), довго-короткочасну пам’ять (LSTM) і автокодер (AE). Щоб створити полегшені IDS для середовища IoT, IDS на основі ML/DL використовуються для вибору функцій і зменшення розмірності [15, 16]. Однак через значне зростання кількості взаємопов’язаних пристроїв IoT, які генерують величезні обсяги даних, IDS

на основі ML/DL стало важко розгортати в мережах IoT з обмеженими ресурсами.

1.2.3 Полегшені федеративні IDS на основі навчання

FL – це розподілена модель ML із збереженням конфіденційності, яка дозволяє клієнтам FL спільно вивчати спільну модель ML, не розкриваючи локальні дані. Застосування FL для мереж IoT використовується нещодавно, оскільки модель ML поширюється серед клієнтів FL, що допомагає зберегти обчислювальні ресурси. Зазвичай використовуються три архітектури FL: централізована FL (клієнт-сервер), децентралізована FL (від пристрою до пристрою) і напівдецентралізована FL.

Централізований FL (клієнт-сервер). У цій архітектурі кожен пристрій самостійно навчає свої локальні моделі, використовуючи власні дані. Параметри або ваги з цих локальних моделей потім агрегуються, щоб сформувати глобальну модель, якою керує центральна сутність, наприклад сервер або координатор, як показано рисунку 1.1. Тренувальний процес складається з кількох турів. У кожному раунді клієнти надсилають свої ваги на сервер, який оновлює глобальну модель. Цей процес триває, доки не буде досягнуто бажаної точності або не буде виконано заздалегідь визначену кількість раундів [6].

Рисунок 1.1 – Архітектура Centralized FL (клієнт-сервер)

Розгортання централізованого FL для мереж Інтернету речей відбувається за двома схемами: по-перше, архітектура край-хмара, у якій хмарний сервер функціонує як сервер FL, а крайовий пристрій – як клієнт FL; по-друге, архітектура крайового рівня, в якій крайовий пристрій функціонує як сервер FL, а пристрій IoT (кінцевий вузол) – як клієнт FL. Багато досліджень було запропоновано на основі хмарної архітектури. В. [17] розробив виявлення кібератак низької складності в периферійних обчисленнях IoT (LocKedge), в якому хмарний сервер витягує найбільш релевантні функції за допомогою методу аналізу головних компонентів (PCA), і можна встановити гіперпараметри та початкові ваги моделі NN. Потім ця інформація надсилається на всі периферійні пристрої для навчання їхніх моделей власними даними за допомогою SGD. Після того, як усі периферійні пристрої завершують процес навчання, вони надсилають оновлені ваги своєї моделі на сервер для процесу агрегації. LocKedge оцінюється в двох режимах, традиційному ML і FL, і результати показали, що традиційний ML має вищий рівень виявлення, ніж FL, через нерівномірний розподіл даних – не IID (незалежні та однаково розподілені між периферійними пристроями). В. [18] розробив модель на основі FL для виявлення кібератак у мережах IIoT. Вони застосували CNN і RNN як для традиційних ML, так і для FL. За їхніми результатами модель FL з малою кількістю раундів і традиційна модель ML значно відрізняються (як CNN, так і RNN). Однак із збільшенням кількості раундів FL маржа зменшується, і на 50-му раунді модель на основі FL досягає такої ж точності виявлення вторгнень, як і традиційна ML. Подібна робота представлена в [19], але з наборами даних, розподіленими в іншому співвідношенні між клієнтами. Їхні результати показують, що відмінності в розподілі наборів даних можуть бути причиною розриву в продуктивності між FL і традиційним ML. Це пояснюється тим, що дані, відмінні від IID, можуть призвести до конфлікту локальних оновлень від різних клієнтів, що призведе до зниження продуктивності глобальної моделі. В [20] запропонував структуру

федеративного навчання, відому як FedDetect для кібербезпеки IoT. Запропонована модель покращила продуктивність завдяки використанню локального адаптивного оптимізатора та планувальника перехресного навчання замість FedAvg для локального навчання. Відповідно до результатів оцінки двох налаштувань – традиційного ML і FL – точність виявлення FL гірша, ніж традиційного ML, оскільки в першому випадку сервер не може безпосередньо вивчати характеристики даних, як у традиційному ML, що робить FL гіршим для вивчення функцій. Однак аналіз ефективності системи вказує на те, що як час наскрізного навчання, так і витрати на пам’ять є доступними та перспективними для пристроїв IoT з обмеженими ресурсами. Ще одна модель IDS на основі FL була розроблена [21] за допомогою глибокого автокодувальника для виявлення атак ботнетів за допомогою децентралізованих даних трафіку на пристрої. Вони встановили віртуальну машину на кожному периферійному пристрої для проведення навчання локальної моделі, а також використали віддзеркалення портів на периферійних пристроях, щоб потік мережевого трафіку до пристроїв IoT не переривався. Однак використання віртуальної машини пов’язане з багатьма проблемами, такими як складність, менша ефективність і висока вартість. Ансамбль, багаторакурсна модель на основі FL була запропонована [22], яка класифікує пакетну інформацію на три групи: двонаправлені характеристики, однонаправлені характеристики та пакетні характеристики. Кожне представлення тренується за допомогою NN, а потім їхні прогнозовані результати надсилаються до класифікатора випадкового лісу (RF), який діє як ансамбль, який об’єднує три передбачення представлень і забезпечує єдиний прогноз атаки на основі її ймовірності та виникнення. Подібним чином в [23] також запропонував виявлення атак на основі FL, яке використовує радіочастоту для об’єднання глобальних моделей ML – GRU з різними розмірами вікон – для виявлення атак у мережах транспортних датчиків (VSN). Згідно з експериментальними результатами обох досліджень [24], використання блоку ансамблю підвищує точність прогнозу атаки. В роботі

[25] застосував FL в сільськогосподарській мережі IoT, використовуючи три моделі DL – DNN, CNN і RNN – на трьох різних наборах даних. Їх результати показують, що один із наборів даних, а саме InSDN, перевершив традиційну модель ML. Результати також демонструють, що на споживання часу та енергії вплинула модель DL та кількість клієнтів, залучених до процесу FL.

Децентралізований FL (від пристрою до пристрою). Децентралізована архітектура FL покладається на зв'язок між пристроями (D2D), де пристрої спільно навчають свої локальні моделі за допомогою стохастичного градієнтного спуску (SGD) і методів на основі консенсусу. Під час кожного кроку консенсусу пристрої діляться своїми локальними оновленнями моделі зі своїми сусідами з одного переходу. Кожен пристрій потім інтегрує оновлення моделі, отримані від своїх сусідів, і вводить результати в процес SGD [9]. У мережах 6G очікується, що пристрої будуть з'єднані між машинами (M2M), що робить зв'язок D2D перспективною технологією для децентралізованої FL [7]. На рисунку 1.2 показано децентралізовану архітектуру FL.

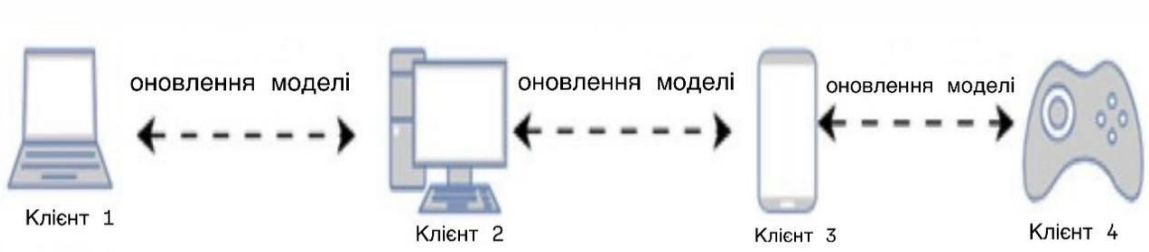

Рисунок 1.2 – Децентралізована архітектура FL (D2D)

Незважаючи на те, що централізований FL продемонстрував успіх у середовищі IoT з точки зору збереження обчислювальних ресурсів, а також покращення процесу навчання, він має деякі недоліки, тобто проблеми з єдиною точкою відмови та масштабування для збільшення розміру мережі. Це призводить до того, що FL є повністю розподіленим (без сервера), у

якому кінцеві пристрої співпрацюють для виконання операцій з даними всередині мережі шляхом повторення локальних обчислень і взаємної взаємодії. В [9] запропонував децентралізований консенсусний підхід на основі FL для мереж IoT, який може забезпечити пряму співпрацю між пристроями (D2D) без опори на центрального координатора. Цей підхід розглядається як багатообіцяюча структура для мереж Інтернету речей через його гнучку оптимізацію моделі в мережах, що характеризуються децентралізованими моделями підключення. Однак вони застосували свої методи до простих NN, які було б важко застосувати до більш глибоких мереж. Надійний децентралізований алгоритм на основі FL був інтегрований із консенсусним підходом у [10]. Їх результати вказують на те, що надійний децентралізований алгоритм FL робить модель стійкою проти атак отруєння. Федеративне навчання блокчейну (B-FL), засноване на консенсусному підході, було запропоновано в [11] для запобігання підробці моделі від зловмисних атак як у локальних, так і в глобальних моделях. Система B-FL складається з кількох периферійних серверів і пристроїв, які генерують блокчейн на основі консенсусного протоколу, щоб підтвердити, що дані в блоці правильні та незмінні. Вони застосували цифрові підписи, щоб гарантувати, що інші не втручаються в інформацію пакетів даних. Хоча цифрові підписи забезпечують автентичність і цілісність даних, вони вимагають обчислювальних витрат, які виходять за межі можливостей пристроїв IoT.

2 МОДЕЛЬ IDS

Представлено запропоновану модель IDS для мереж IoT. Дослідження починається з опису того, як генеруються FL-клієнти з урахуванням усіх типів атак. Після цього показано, як напівдецентралізований FL застосовується після кластеризації клієнтів FL. Далі наводиться огляд методів глибокого навчання, а саме LSTM, BiLSTM і WGAN, які служать локальними моделями в архітектурі FL.

2.1 Створення клієнтів FL

Перш за все, набір даних буде попередньо оброблено та розділено на набори для навчання, перевірки та тестування. Потім навчальний набір розподіляється між клієнтами FL, враховуючи, що кожен клієнт повинен мати всі типи атак, а також безпечні дані. Лістинг 2.1 представляє метод генерації клієнтів FL.

Лістинг 2.1 – Метод генерації клієнтів FL

```
import numpy as np
from typing import List, Dict, Any

def generate_federated_clients(TrainSet: np.ndarray, Labels:
np.ndarray, NumberOfClients: int) -> List[Dict[str, Any]]:

    # Shuffle the dataset
    shuffle_idx = np.random.permutation(len(TrainSet))
    TrainSet = TrainSet[shuffle_idx]
    Labels = Labels[shuffle_idx]

    # Get unique classes and their indices
    classes = np.unique(Labels)
    NumberOfClasses = len(classes)

    # Get indices for each class
    class_indices = {cls: np.where(Labels == cls)[0] for cls in
classes}
    ClassSize = {cls: len(indices) for cls, indices in
```

```
class_indices.items()}

    Clients = []
    StartIndex = {cls: 0 for cls in classes}

    for i in range(NumberOfClients):
        client_data = {'features': [], 'labels': []}

        # For last client, take all remaining data
        if i == NumberOfClients - 1:
            for cls in classes:
                indices = class_indices[cls][StartIndex[cls]:]

client_data['features'].extend(TrainSet[indices])
                client_data['labels'].extend(Labels[indices])
        else:
            for cls in classes:
                # Generate random ratio for this class (between
0.1 and 0.9)
                ratio = np.random.uniform(0.1, 0.9)
                EndIndex = StartIndex[cls] + int(ClassSize[cls]
* ratio)

                # Ensure we don't go out of bounds and leave
some data for other clients
                EndIndex = min(EndIndex, ClassSize[cls] -
(NumberOfClients - i - 1))
                if EndIndex <= StartIndex[cls]:
                    EndIndex = StartIndex[cls] + 1

                indices =
class_indices[cls][StartIndex[cls]:EndIndex]

client_data['features'].extend(TrainSet[indices])
                client_data['labels'].extend(Labels[indices])
                StartIndex[cls] = EndIndex

        # Convert to numpy arrays
        client_data['features'] =
np.array(client_data['features'])
        client_data['labels'] = np.array(client_data['labels'])
        Clients.append(client_data)
    return Clients
```

Оскільки класи набору даних є незбалансованими, нам потрібно враховувати це під час розподілу набору даних, беручи співвідношення для кожного класу на основі розміру класу (кількість екземплярів для кожного типу класу). Розподіл набору даних здійснюється випадковим чином; Таким чином, ми генеруємо п’ять різних дистрибутивів клієнтів, відомих як

«запуски».

2.2 Запропонована напівдецентралізована модель FL

Запропонована модель напівдецентралізованого федеративного навчання (FL) кластеризує клієнтів на основі оновлень моделі, отриманих від клієнтів FL. Зокрема, кожен клієнт навчається використанню простого багаторівневого персептрона (MLP) для одного раунду як етап попередньої обробки перед кластеризацією. Далі оновлені ваги моделі збираються від кожного клієнта для підготовки до кластеризації. Оскільки вагові коефіцієнти моделі мають велику розмірність, перед кластеризацією клієнтів застосовується зменшення розмірності, щоб уникнути проблем із розрідженістю в кластерах. В роботі [26] провели порівняльне дослідження між аналізом головних компонентів (PCA) і автокодером (AE) для зменшення розмірності перед кластеризацією. Їх результати показують, що AE перевершує PCA в їхньому прикладі. Тому до отриманих модельних ваг застосовують AE для зменшення їх розмірів. Зменшені ваги потім масштабуються перед застосуванням алгоритму кластеризації [27]. Щоб визначити найбільш прийнятний алгоритм кластеризації, ми досліджуємо різні алгоритми, щоб визначити найкращий для нашого дослідження. В [28] розглянули та проаналізували існуючі алгоритми кластеризації на основі ключових показників, таких як часова складність, масштабованість і якість кластеризації. Відповідно до їхнього порівняльного дослідження, k- середніх є найбільш підходящим алгоритмом кластеризації для нашого випадку, оскільки він ефективно обробляє великі набори даних з низькою складністю часу. Щоб застосувати кластеризацію k-середніх, необхідно визначити кількість кластерів, а також метрику відстані. Згідно з експериментальними результатами, k-середнє досягає найвищих результатів. Голова кластера (ГК) вибирається для кожного кластера на основі усередненого балу Silhouette. Оцінка Silhouette використовується, оскільки вона вказує, чи правильно

позиціонується клієнт у межах призначеного кластера та чи не надто близько до межі сусідніх кластерів. Лістинг 2.2 узагальнює розроблену нами методологію кластеризації.

Лістинг 2.2 – Кластерізація

```
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from typing import List, Dict, Any

def cluster_clients(clients: List[Dict[str, Any]],
                    input_shape: tuple,
                    n_epochs: int = 5) -> List[List[int]]:

    # 1. Train client models and collect weights
    weights = []
    client_models = []

    for i, client in enumerate(clients):
        print(f"Training client {i} model...")
        model = create_client_model(input_shape)  # Функція
створення моделі
        history = model.fit(client['features'],
client['labels'],
                            epochs=n_epochs, verbose=0)
        weights.append(model.get_weights())
        client_models.append(model)

    # 2. Flatten and standardize weights for dimensionality
reduction
    flattened_weights = [np.concatenate([w.flatten() for w in
weight]) for weight in weights]
    flattened_weights = np.array(flattened_weights)
    scaler = StandardScaler()
    scaled_weights = scaler.fit_transform(flattened_weights)

    # 3. Dimensionality reduction (using PCA instead of
Autoencoder for simplicity)
    pca = PCA(n_components=0.95)  # Зберігаємо 95% дисперсії
    reduced_weights = pca.fit_transform(scaled_weights)

    # 4. Find optimal number of clusters using Elbow method
    distortions = []
    max_clusters = min(10, len(clients))  # Не більше 10
кластерів або кількості клієнтів
    for k in range(1, max_clusters + 1):
        kmeans = KMeans(n_clusters=k, random_state=42)
```

```
            kmeans.fit(reduced_weights)
            distortions.append(kmeans.inertia_)

    # Find elbow point (simplified method)
    n_clusters = find_elbow_point(distortions)

    # 5. Perform K-means clustering with Manhattan distance
    # KMeans doesn't support custom distance metrics, so we use
precomputed
    manhattan_dist = pairwise_distances(reduced_weights,
metric='manhattan')
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    clusters = kmeans.fit_predict(manhattan_dist)

    # 6. Organize clients into clusters
    clustered_clients = [[] for _ in range(n_clusters)]
    for client_idx, cluster_idx in enumerate(clusters):
        clustered_clients[cluster_idx].append(client_idx)

    return clustered_clients

def create_client_model(input_shape: tuple):
    """Create a simple client model"""
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense

    model = Sequential([
        Dense(64, activation='relu', input_shape=input_shape),
        Dense(32, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy')
    return model

def find_elbow_point(distortions: List[float]) -> int:
    """Simplified elbow point detection"""
    diffs = np.diff(distortions)
    diff_ratios = diffs[:-1] / diffs[1:]
    elbow_point = np.argmax(diff_ratios) + 2  # +2 because we're
comparing ratios
    return min(elbow_point, len(distortions))
```

Напівцентралізована модель FL починається після кластеризації клієнтів і призначення ГК для кожного кластера. Сервер транслює модель із початковими вагами та параметрами до ГК, потім ГК надсилає їх усім клієнтам у своєму кластері. Потім кожен клієнт навчає свою локальну модель і надсилає оновлену модель до відповідного ГК. Коли ГК отримує оновлену

модель від усіх клієнтів, він обчислює середні ваги та надсилає їх на сервер. Після того як сервер отримує всі нові вагові коефіцієнти від кожного кластера, він агрегує отримані вагові коефіцієнти моделі від усіх ГК і обчислює глобальну модель за допомогою FedAvg. Потім сервер транслює нову модель головам кластера для оновлення їх локальної моделі. Цей процес повторюється для заданої кількості раундів або доти, доки модель не зійдеться, при цьому суттєвого покращення продуктивності не спостерігається. Щоб оцінити продуктивність під час навчання, глобальна модель, створена після кожного раунду, перевіряється за допомогою набору перевірки, і обчислюються показники продуктивності. В додатку Б представлено реалізований алгорим на Python, який узагальнює напівдецентралізовану модель. Листінги 2.3-2.5 узагальнюють напівдецентралізовану модель, демонструючи процедуру навчання на стороні сервера, стороні кластера та стороні клієнта відповідно. Після завершення навчання FL глобальна модель тестується на наборі для тестування, щоб оцінити її продуктивність. Рисунок 2.1 ілюструє запропоновану напівдецентралізовану архітектуру на основі FL.

Листінг 2.3 – Напівдецентралізоване навчання кластера FL

```
import numpy as np
from typing import List, Dict, Any
from tensorflow.keras.models import clone_model

class ClusterAggregator:
    def __init__(self, cluster_model):

        self.cluster_model = cluster_model
        self.client_weights = []

    def aggregate_weights(self, global_weights:
List[np.ndarray],
                          clients: List[Dict[str, Any]],
                          n_epochs: int = 3) -> List[np.ndarray]:

        # Step 1-2: Initialize cluster model with global weights
        cluster_model = clone_model(self.cluster_model)
        cluster_model.set_weights(global_weights)
        cluster_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy')
```

```
        # Reset client weights storage
        self.client_weights = []

        # Step 3: Distribute model to clients and collect updates
        for client in clients:
            # Create client model copy
            client_model = clone_model(cluster_model)
            client_model.set_weights(global_weights)
            client_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy')

            # Train client model
            client_model.fit(
                client['features'],
                client['labels'],
                epochs=n_epochs,
                verbose=0
            )

            # Store client weights
            self.client_weights.append(client_model.get_weights())

        # Step 4-5: Aggregate client weights
        if not self.client_weights:
            return global_weights  # No clients case

        aggregated_weights = []
        for layer_idx in range(len(global_weights)):
            # Collect this layer's weights from all clients
            layer_weights = [client_weight[layer_idx] for
client_weight in self.client_weights]

            # Average weights for this layer
            avg_layer_weights = np.mean(layer_weights, axis=0)
            aggregated_weights.append(avg_layer_weights)

        return aggregated_weights

    def federated_averaging(self, client_weights:
List[List[np.ndarray]]) -> List[np.ndarray]:
        if not client_weights:
            return []

        # Initialize with first client's weights structure
        avg_weights = [np.zeros_like(w) for w in client_weights[0]]
        n_clients = len(client_weights)

        # Sum all weights
        for weights in client_weights:
            for layer_idx, layer_weights in enumerate(weights):
                avg_weights[layer_idx] += layer_weights /
n_clients

        return avg_weights
```

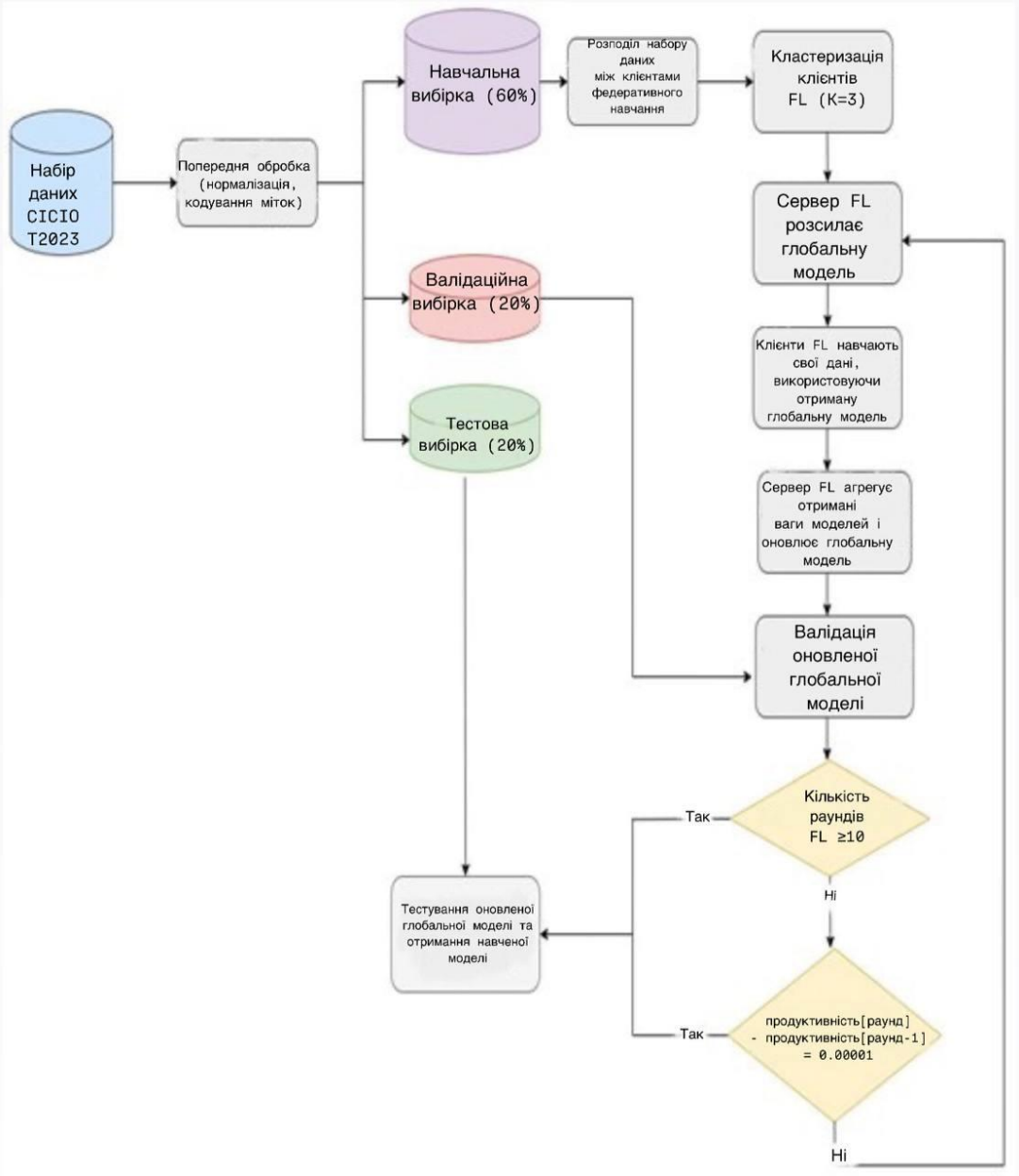

Рисунок 2.1 – Напівдецентралізована архітектура FL

Лістінг 2.4 – Напівдецентралізоване навчання клієнта FL

```
import numpy as np
from typing import Tuple, List
from tensorflow.keras.models import clone_model

class FLClient:
    def __init__(self, client_id: int, features: np.ndarray, labels:
np.ndarray):

        self.client_id = client_id
        self.features = features
        self.labels = labels
```

```
        self.local_model = None
        self.training_history = []

    def train_local_model(self,
                          global_weights: List[np.ndarray],
                          model_template,
                          n_epochs: int = 3,
                          batch_size: int = 32,
                          verbose: int = 0) ->
Tuple[List[np.ndarray], dict]:

        # Step 1-2: Initialize local model with received weights
        self.local_model = clone_model(model_template)
        self.local_model.set_weights(global_weights)
        self.local_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy')

        # Step 3: Train local model
        history = self.local_model.fit(
            self.features,
            self.labels,
            epochs=n_epochs,
            batch_size=batch_size,
            verbose=verbose,
            validation_split=0.1  # Optional validation split
        )

        # Store training history
        self.training_history.append(history.history)

        # Step 4: Get updated weights
        updated_weights = self.local_model.get_weights()

        # Step 5: Return updated weights and training info
        training_info = {
            'client_id': self.client_id,
            'final_loss': history.history['loss'][-1],
            'final_accuracy': history.history.get('accuracy', [None])[-
1],
            'n_samples': len(self.features)
        }

        return updated_weights, training_info

    def evaluate_local_model(self, test_data: Tuple[np.ndarray,
np.ndarray]) -> dict:

        if self.local_model is None:
            raise ValueError("Model not trained yet. Call
train_local_model() first.")

        X_test, y_test = test_data
        results = self.local_model.evaluate(X_test, y_test,
verbose=0)
```

```
        if isinstance(results, list):
            metrics = {
                'loss': results[0],
                'accuracy': results[1] if len(results) > 1 else None
            }
        else:
            metrics = {'loss': results}

        metrics['client_id'] = self.client_id
        return metrics
```

Листінг 2.5 – Тестування моделі FL

```
import numpy as np
from typing import Tuple, Dict, Any
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix

class FLModelEvaluator:
    def __init__(self, global_model):
        """
        Initialize Federated Learning Model Evaluator.

        Args:
            global_model: Trained global model to evaluate
        """
        self.global_model = global_model
        self.metrics_history = []

    def evaluate_model(self, X_test: np.ndarray, y_test:
np.ndarray) -> Dict[str, float]:
        """
        Evaluate global model on test set and compute multiple
performance metrics.

        Args:
            X_test: Test features
            y_test: True labels for test set

        Returns:
            Dictionary with computed metrics
        """
        # Step 1-2: Already have X_test and y_test as inputs
        # Step 3: Get model predictions
        y_pred = self.global_model.predict(X_test, verbose=0)

        # For classification models, get class predictions
        if y_pred.ndim > 1 and y_pred.shape[1] > 1:
            y_pred_classes = np.argmax(y_pred, axis=1)
        else:
            y_pred_classes = (y_pred > 0.5).astype(int)
```

```
        # Step 4: Calculate multiple performance metrics
        metrics = {
            'accuracy': accuracy_score(y_test, y_pred_classes),
            'precision': precision_score(y_test, y_pred_classes,
average='weighted', zero_division=0),
            'recall': recall_score(y_test, y_pred_classes,
average='weighted', zero_division=0),
            'f1_score': f1_score(y_test, y_pred_classes,
average='weighted', zero_division=0),
            'confusion_matrix': confusion_matrix(y_test,
y_pred_classes).tolist()
        }

        # For binary classification add additional metrics
        if len(np.unique(y_test)) == 2:
            metrics.update({
                'binary_precision': precision_score(y_test,
y_pred_classes),
                'binary_recall': recall_score(y_test,
y_pred_classes),
                'binary_f1': f1_score(y_test, y_pred_classes)
            })

        # Store metrics in history
        self.metrics_history.append(metrics)

        return metrics

    def get_detection_rate(self, X_test: np.ndarray, y_test:
np.ndarray, positive_class: int = 1) -> float:
        """
        Calculate detection rate (recall for specific class) as
required in the algorithm.

        Args:
            X_test: Test features
            y_test: True labels
            positive_class: Class to consider as positive for
detection

        Returns:
            Detection rate (recall for specified class)
        """
        y_pred = self.global_model.predict(X_test, verbose=0)

        if y_pred.ndim > 1 and y_pred.shape[1] > 1:
            y_pred_classes = np.argmax(y_pred, axis=1)
        else:
            y_pred_classes = (y_pred > 0.5).astype(int)

        # Calculate true positives and false negatives
        tp = np.sum((y_pred_classes == positive_class) & (y_test
== positive_class))
```

```
        fn = np.sum((y_pred_classes != positive_class) & (y_test
== positive_class))

        detection_rate = tp / (tp + fn) if (tp + fn) > 0 else
0.0
        return detection_rate

    def compare_with_baseline(self, baseline_metrics: Dict[str,
float]) -> Dict[str, Any]:
        """
        Compare current metrics with baseline metrics.

        Args:
            baseline_metrics: Dictionary with baseline metrics

        Returns:
            Dictionary with comparison results
        """
        if not self.metrics_history:
            raise ValueError("No metrics available. Run
evaluate_model first.")

        current_metrics = self.metrics_history[-1]
        comparison = {}

        for metric in current_metrics:
            if metric in baseline_metrics and not
isinstance(current_metrics[metric], list):
                comparison[metric] = {
                    'current': current_metrics[metric],
                    'baseline': baseline_metrics[metric],
                    'improvement': current_metrics[metric] -
baseline_metrics[metric]
                }

        return comparison
```

2.3 Алгоритм агрегації

Після того, як усі клієнти навчили свої дані, кожен головний вузол кластера збирає нові параметри/ваги від своїх клієнтів, обчислює середні ваги та надсилає їх на сервер для агрегації. Потім сервер обчислює агреговані ваги шляхом усереднення отриманих ваг/параметрів за допомогою алгоритму FedAvg, як показано в наступній формулі:

$$w_{\text{global}} = \sum_{k=1}^{K} \frac{n_k}{n} w_k \tag{2.1}$$

- w_{global}– це ваги глобальної моделі.
- K – кількість кластерів.
- n_k– загальна кількість вибірок даних у кожному кластері k.
- n – загальна кількість вибірок даних у всіх кластерах,
- w_k – ваги моделі, навченої в кластері k.

Рівняння 2.1 показує, що глобальні ваги моделі є зваженою сумою ваг моделей кластера, причому ваги пропорційні кількості вибірок даних у кожному кластері. FedAvg використовується на рівні сервера, враховуючи, що кожен кластер надсилає середні ваги своїх клієнтів на сервер, на відміну від інших напівдецентралізованих підходів, де процес агрегації виконується на двох рівнях: голова кластера та сервер. Голова кластера проводить процес агрегації протягом кількох раундів для отримання моделі кластера. Потім сервер агрегує отримані моделі кластера для отримання глобальної моделі та надсилає її назад головам кластера для наступного раунду FL. Цей процес є обчислювально ресурсоємним через повторну агрегацію та розподіл всередині кластерів. Запропонована нами модель зменшує цю складність, дозволяючи кожному голові кластера збирати ваги від своїх клієнтів, усереднювати їх один раз та надсилати усереднені ваги безпосередньо на сервер. Потім сервер агрегує отримані моделі кластера та повертає глобальну модель головам кластера, все в межах одного раунду FL, тим самим зменшуючи накладні витрати на зв'язок та роблячи його доступним для Інтернету речей з обмеженими ресурсами.

2.4 Локальні моделі глибокого навчання

2.4.1 Довготривала короткочасна пам'ять (LSTM)

LSTM – це розширення рекурентних нейронних мереж (RNN) зі здатністю ефективно вирішувати проблему зникнення градієнтів. LSTM розширює можливості пам'яті RNN, дозволяючи їм вивчати довгострокові залежності від вхідних даних. Ця розширена пам'ять може зберігати інформацію протягом триваліших періодів, що дозволяє моделі зчитувати, записувати та видаляти інформацію за потреби. Пам'ять LSTM структурована як «закрита» комірка, тобто вона може вирішувати, чи зберігати, чи відкидати інформацію. Цей механізм стробування дозволяє LSTM захоплювати важливі ознаки з вхідних даних та зберігати цю інформацію протягом тривалих послідовностей. Рішення про збереження або видалення інформації залежить від ваг, призначених під час навчання, що дозволяє моделі дізнатися, які деталі варто зберегти або відкинути. Модель LSTM зазвичай складається з трьох вентилів: вентиля забуття, вхідного вентиля та вихідного вентиля, як показано на рисунку 2.2. «Вентиль забуття» вирішує, яку інформацію слід зберегти або відкинути. Вона використовує сигмоїдну функцію для оцінки як поточного вхідного, так і x_t попередній прихований стан $h_{t-}1$. Вихідний сигнал вентиля забуття, представлений як f_t, коливається від нуля до одиниці. Значення, близьке до нуля, вказує на те, що інформація буде відкинута, тоді як значення, близьке до одиниці, означає, що вона буде збережена. Цей механізм формулюється як

$$f_t = \sigma\left(W_f \cdot h_{t-1}, x_t + b_f\right) \tag{2.2}$$

де W_f представляє вагу та b_f упередження для воріт забуття, та σ позначає функцію активації сигмоїда.

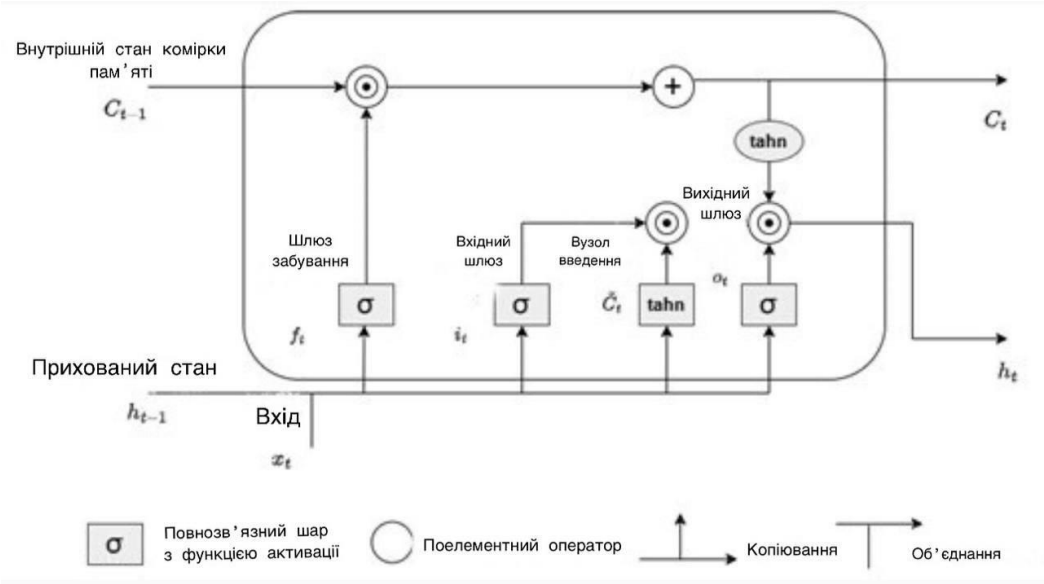

Рисунок 2.2 – Архітектура комірки LSTM

Потім «вхідний вентиль» визначає, яку нову інформацію слід додати до стану комірки. Він також використовує сигмоїдну функцію, яка призводить до значень від нуля (неважливо) до одиниці (важливо). Функція вхідного вентиля визначається як

$$i_t = \sigma\left(W_i \cdot h_{t-1}, x_t + b_i\right) \tag{2.3}$$

Далі функція tanh обробляє поточний вхідний сигналxnхні попередній прихований стан h_{t-1}, генеруючи вектор нових значень-кандидатів, C_t, який можна додати до стану комірки:

$$\tilde{C}_t = \tanh\left(W_c \cdot h_{t-1}, x_t + b_c\right) \tag{2.4}$$

Стан клітини C_t оновлюється шляхом попереднього множення стану попередньої комірки C_{t-1} від f_n, а потім додавання добутку i_t і C_t де $\odot$ позначає поелементне множення, а tanh – гіперболічний тангенс активаційної функції.

$$C_t = \left(f_t \odot C_{t-1} + i_t \odot \tilde{C}_t\right) \tag{2.5}$$

Нарешті, «вихідний вентиль» визначає наступний прихований стан h_t Він враховує як стан комірки, так і вихід попереднього шару:

$$o_t = \sigma\left(W_o \cdot h_{t-1}, x_t + b_o\right) \tag{2.6}$$

$$h_t = o_t \odot \tanh\left(C_t\right) \tag{2.7}$$

У цих рівняннях, W_o і b_o – це ваги та зміщення, пов'язані з вихідним вентилем. Вихідний вентиль використовує сигмоїдну функцію σ і функцію tanh для обчислення кінцевого результату h_t, що представляє наступний прихований стан.

2.4.2 Двонаправлена довготривала короткочасна пам'ять (BiLSTM)

BiLSTM – це розширення моделі LSTM, де до вхідних даних застосовуються дві моделі LSTM. На відміну від стандартної мережі LSTM, яка використовує лише інформацію, з якою вона вже зіткнулася в послідовності, архітектура BiLSTM включає два шари LSTM – один обробляє вхідну послідовність вперед (прямий LSTM), а інший обробляє її назад (зворотний LSTM), як показано на рисунку 2.3. Таке подвійне застосування LSTM значно розширює можливості моделі вивчати довгострокові залежності, що зрештою призводить до покращення продуктивності.

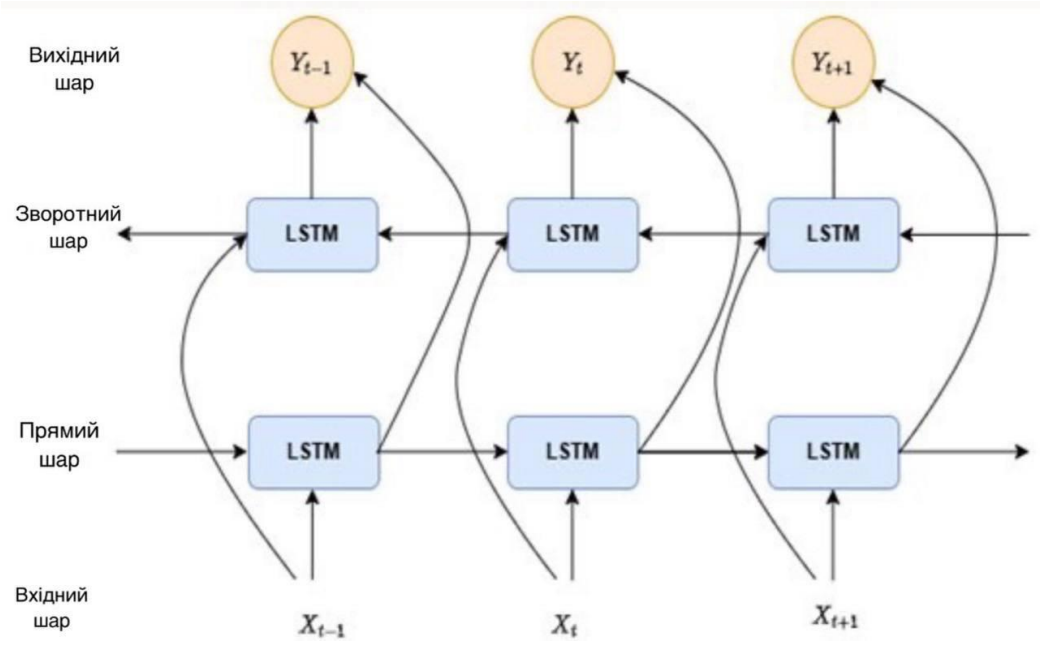

Рисунок 2.3 – Двонаправлена LSTM

2.4.3 Генеративна змагальна мережа Вассерштайна (WGAN)

Вассерштейн GAN (WGAN) є альтернативою традиційній моделі GAN. У GAN є дві нейронні мережі – генератор і дискримінатор – які беруть участь у динамічному змагальному процесі. Генератор прагне захопити розподіл даних, а потім створити синтетичні дані, які виглядають як реальні. З іншого боку, дискримінатор оцінює вхідні зразки та прогнозує, чи походять вони з реального набору даних, чи були згенеровані синтетично. Ця конкуренція спонукає обидві мережі до вдосконалення, що призводить до створення все більш реалістичних даних.

WGAN покращує стабільність навчання традиційної GAN, замінюючи загальновживані метрики дивергенції, які можуть не бути неперервними щодо параметрів генератора, на відстань Earth-Mover (Вассерштейн-1). Ця відстань вимірює мінімальну вартість перетворення одного розподілу ймовірностей в інший, враховуючи як кількість переміщеної маси, так і відстань перенесення. За м'яких припущень відстань Вассерштейна є неперервною та диференційованою майже скрізь, що забезпечує стабільніше навчання. Однак процес оптимізації в WGAN може бути складним через

взаємодію між ваговим обмеженням та функцією вартості, що може призвести до зникнення або вибуху градієнтів, якщо поріг відсікання не буде ретельно налаштовано. Щоб вирішити ці проблеми, WGAN з градієнтним штрафом застосовує обмеження Ліпшица за допомогою градієнтного штрафу, що є більш надійною альтернативою відсіканню ваг.

Диференційовна функція є 1-ліпшицевою тоді і тільки тоді, коли її градієнти мають норму не більше 1 всюди. Щоб забезпечити виконання цього обмеження, WGAN зі штрафом за градієнт вводить послаблену версію, штрафуючи норму градієнта для випадкових вибірок, що забезпечує стабільну динаміку навчання. Нова цільова функція задана як

$$L = \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}\left[D\left(\tilde{x}\right)\right] - \mathbb{E}_{x\sim\mathbb{P}_r}\left[D\left(x\right)\right] + \lambda \cdot \mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}\left[\left(\| \nabla_{\hat{x}} D\left(\hat{x}\right) \|_2 - 1\right)^2\right], \quad (2.8)$$

У контексті федеративного навчання (FL) GAN використовувалися з одним генератором, спільним для всіх клієнтів, для генерації узгоджених синтетичних даних. Кожен FL-клієнт навчає локальний дискримінатор на своїх даних, а оновлені моделі агрегуються на сервері для побудови глобального дискримінатора. Однак навчання GAN у цій напівдецентралізованій системі FL часто дає неефективні результати, оскільки генератор має труднощі з навчанням у клієнтів з різноманітними розподілами даних, що не відповідають IID, що призводить до високих втрат генератора. Щоб вирішити цю неефективність, використовується WGAN з градієнтним штрафом шляхом багаторазового навчання критика на локальних даних та агрегації локальних дискримінаторів у глобальну модель.

Реалізовано коефіцієнт градієнтного штрафу встановлено на 10 (λ=10). Критик навчається п'ять разів для кожного оновлення генератора, щоб забезпечити точне наближення відстані Вассерштейна перед оновленням генератора. Таке поєднання градієнтного штрафу та кількох оновлень критика значно покращує градієнтний потік, зменшує колапс мод та

забезпечує стабільну динаміку навчання, що робить WGAN з градієнтним штрафом надійною альтернативою традиційним GAN у федеративних навчальних системах.

3 ПОБУДОВА СИСТЕМИ ТА ЕКСПЕРЕМЕНТИ

У цьому розділі показано експериментальні деталі запропонованих моделей. Усі експерименти проводилися на Google Colab, який забезпечував доступ до графічного процесора NVIDIA Tesla T4 з 12 ГБ оперативної пам'яті через свою хмарну інфраструктуру. Як централізовані, так і напівдецентралізовані моделі на основі FL були реалізовані за допомогою мови програмування Python 3.12.12 з TensorFlow 2.17.0, Keras 3.4.1 та scikit-learn 1.3.2. Кластеризація реалізована за допомогою PyClustering, бібліотеки з відкритим кодом.

3.1 Набори даних

3.1.1 Навчальний набір даних

Використаний набір даних CICIoT2023, створений Канадським інститутом кібербезпеки (CIC). Набір даних CICIoT2023 був згенерований з використанням топології, що включає 105 пристроїв Інтернету речей. Серед них 67 пристроїв активно брали участь в атаках, а п'ять хабів підключили ще 38 пристроїв Zigbee та Z-Wave. Ця схема відтворює реальний сценарій продуктів та послуг Інтернету речей у середовищі розумного дому. Пристрої, що використовуються в цій топології, включають пристрої розумного дому, камери, датчики та мікроконтролери, всі з яких були взаємопов'язані та налаштовані для сприяння різним атакам та захоплення трафіку, що виникає в результаті атаки. Таким чином, набір даних CICIoT2023 є неоднорідним, оскільки він має різноманітні типи пристроїв та протоколи зв'язку.

CICIoT2023 складається з 33 типів атак, які поділяються на сім категорій: розподілена відмова в обслуговуванні (DDoS), відмова в обслуговуванні (DoS), розвідка, веб-атаки, груба сила, підміна та Mirai.

Нешкідливі дані збираються з трафіку Інтернету речей у станах очікування, коли пристрої налаштовані з параметрами за замовчуванням та без шкідливих або атакуючих скриптів. CICIoT2023 складається з 46217820 екземплярів, кожен з яких представляє мережевий потік із 47 функціями. На рисунку 3.1 показано кількість екземплярів для кожної категорії атак.

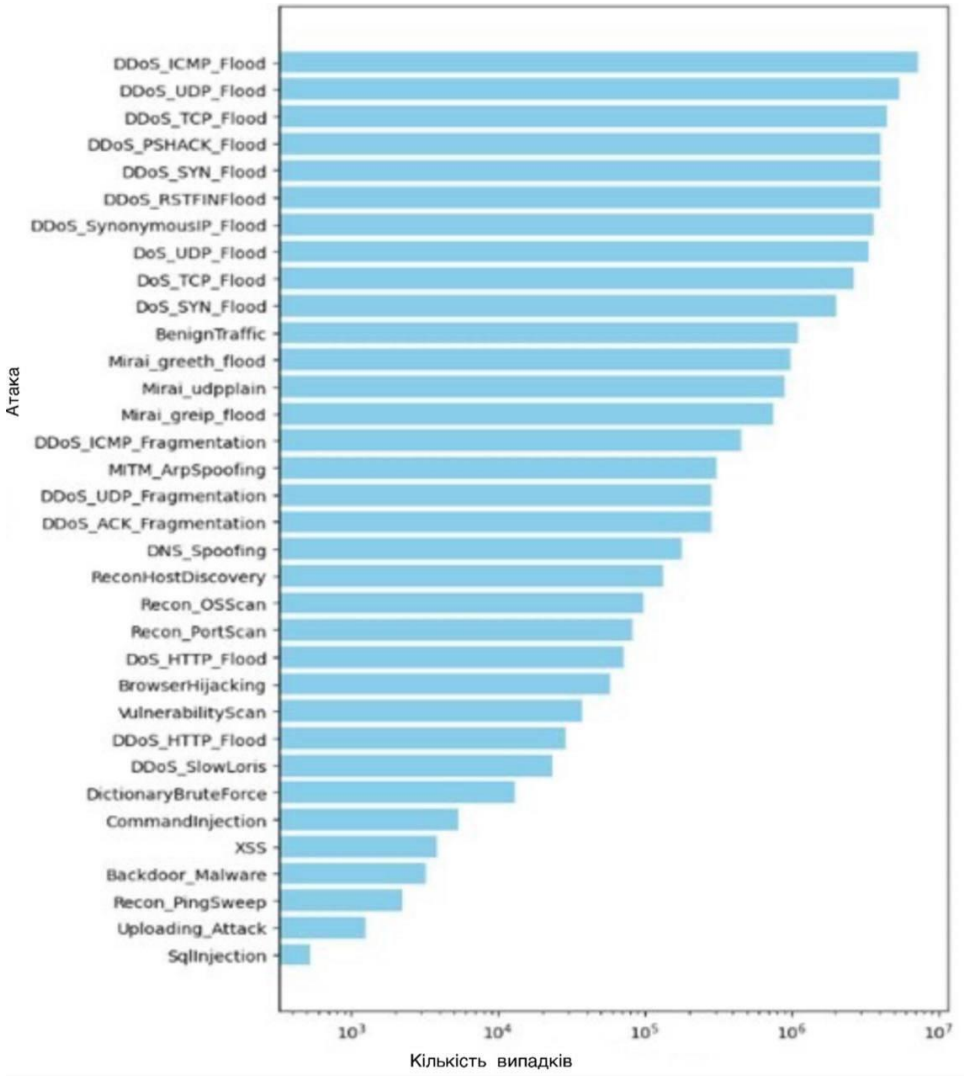

Рисунок 3.1 – Розподіл класів CICIoT2023

Було обрано набір даних CICIoT2023, оскільки він включає всі класи як у навчальному, так і в тестовому наборах, на відміну від інших наборів даних. Крім того, кількість екземплярів на клас достатньо велика, щоб її можна було розподілити між клієнтами FL, що дозволяє навчити кожного клієнта на кожному класі. Це, у свою чергу, сприяє розробці точної

глобальної моделі. Наскільки нам відомо, ми першими використовуємо весь набір даних CICIoT2023, враховуючи всі типи атак, оскільки інші дослідження працюють з частиною набору даних або розглядають конкретні типи атак

Перш за все, набір даних потрібно попередньо обробити та розділити на навчальний, валідаційний та тестовий набори з пропорціями 60%, 20% та 20% відповідно. Ознаки нормалізуються за допомогою методу StandardScaler, тоді як класи (мітки) кодуються за допомогою One-Hot Encoding для багатокласової класифікації (34/8 класів) та Binary Encoding для двійкової класифікації.

3.1.2 Набір даних тестування

Для тестування запропонованої попередньо навченої моделі використовуються три набори даних, а саме BoT-IoT, WUSTL-IIOT-2021 та Edge-IIoTset. Метою дослідження було перевірити узагальнюваність запропонованої моделі на додаткових наборах даних та покращити її продуктивність шляхом точного налаштування. Причиною вибору цих наборів даних є те, що вони мають схожі характеристики з навчальним набором даних. Крім того, атаки DoS та DDoS представлені у всіх наборах даних, які є основними атаками, на які спрямована наша модель. Що ще важливіше, всі набори даних розроблені для трафіку Інтернету речей.

Набір даних BoT-IoT було створено шляхом проектування реалістичного мережевого середовища IoT з п'ятьма різними сценаріями IoT: метеостанція, розумний холодильник, дистанційно активовані світильники, світильники, що активуються рухом, та розумний термостат. Ми використали 5% версію, отриману з оригінального набору даних, яка має 35 ознак. Набір даних включає 10 типів атак: DDoS (HTTP, TCP, UDP), DoS (HTTP, TCP, UDP), зчитування відбитків ОС, сканування сервера, кейлоггеринг та атаки витоку даних, як показано в таблиці 3.1. Загалом більше 70% даних

використовуються для навчання, а решта 30% – для тестування. Оригінальна модель мала 47 вхідних ознак, тоді як набори даних BoT-IoT мали 35 ознак. Таким чином, ми модифікували вхідний шар моделі, щоб врахувати цю різницю.

Таблиця 3.1 – Типи атак та кількість зразків у наборі даних BoT-IoT

Типи атак	Кількість зразків
DoS-HTTP	1485
DoS-TCP	615 800
DoS-UDP	1 032 975
DDoS-HTTP	989
DDoS-TCP	977 380
DDoS-UDP	948 255
Зняття відбитків ОС	17 914
Сканування сервера	73 168
Кейлоггер	73
Крадіжка даних	6
Звичайний	477
Всього	3 668 522

Набір даних WUSTL-IIOT-2021 містить мережевий трафік із систем промислового Інтернету речей (IIoT), які використовувалися в дослідженнях кібербезпеки. Набір даних має 41 ознаку після видалення 4 ознак, оскільки видавець набору даних зазначив, що «вони унікальні для атак і розкривають тип атаки для моделі; отже, модель не буде узагальнена для невидимих даних». Крім того, видаляються невикористані стовпці, а саме «StartTime», «LastTime», «SrcAddr», «DstAddr», «sIpId» та «dIpId». В таблиці 3.2 представлена інформація про набір даних. Набір даних містить чотири типи різних атак: DoS, впровадження команд, розвідувальні атаки та атаки Backdoor. Оскільки DoS-атаки зазвичай генерують великі обсяги трафіку та

велику кількість зразків, 90% даних атаки виділяється для їх представлення. Інші типи атак трапляються рідше, і коли вони трапляються, вони передають лише обмежену кількість даних трафіку. В таблиці 3.3 наведено статистику набору даних. Загалом 80% використовується для навчання, а решта 20% – для тестування.

Таблиця 3.2 – Специфікація набору даних WUSTL-IIOT-2021

Кількість зразків	1194464
Кількість функцій	41
Кількість зразків атаки	87016
Кількість нормальних зразків	1107448

Таблиця 3.3 – Статистична інформація про типи трафіку в WUSTL-IIOT-2021

Тип трафіку	Відсоток (%)
Звичайний рух	92,72
Загальний трафік атаки	7.28
Трафік впорскування команд	0,31
Трафік DoS	89,98
Розвідувальний рух	9.46
Трафік бекдору	0,25

Набір даних Edge-IIoTset було створено за допомогою тестового середовища, яке точно імітує реальне середовище Інтернету речей/IIoT. Набір даних містить реальні дані, зібрані шляхом виконання реалістичних кібератак та захоплення як легітимного, так і шкідливого мережевого трафіку. Набір даних має 61 функцію та 14 типів атак, які поділені на п'ять категорій: DDoS, ін'єкції, MITM, шкідливе програмне забезпечення та атаки сканування. В таблиці 3.4 показано розподіл класів в Edge-IIoTset. Набір даних складається з кількох файлів, включаючи трафік атаки, звичайний трафік та вибрані набори даних для машинного навчання (ML) та об'єднання

(DL), які містять два файли CSV: DNN-EdgeIIoT-dataset.csv та ML-EdgeIIoT-dataset.csv. Ми використовуємо DNN-EdgeIIoT-dataset.csv для оцінки нашої моделі.

Таблиця 3.4 – Типи атак та кількість семплів в Edge-IIoTset

Категорія атаки	Атаки	Кількість екземплярів
DoS/DDoS-атаки	DDoS-атака_HTTP	229022
	DDoS-атака_ICMP	2914354
	DDoS-атака_TCP	2020120
	DDoS-атака_UDP	3201626
Ін'єкційні атаки	Атака SQL_ін'єкцій	51203
	Атака завантаження	37634
	XSS-атака	15915
Атаки шкідливого програмного забезпечення	Атака з бекдором	24862
	Атака на пароль	1053385
	Атака програм-вимагачів	10925
Сканування атак	Атака сканування портів	22564
	Атака сканера вразливостей	145869
	Атака з відбитків пальців	1001
Атака MITM		1229
Звичайний		11223940
Всього		20952648

Сімдесят відсотків даних використовуються для навчання, а решта тридцять відсотків зарезервовані для тестування. Нами застосовано точне

налаштування до всіх наборів даних для тестування нашої попередньо навченої моделі, оскільки вхідні ознаки відрізняються від попередньо навченої моделі, а кількість класів змінюється. Під час точного налаштування всі шари, крім вхідного та вихідного, заморожуються, щоб зберегти вивчені представлення, дозволяючи моделі адаптуватися до нового простору ознак. Цей процес значно покращує продуктивність моделі на невидимих наборах даних, демонструючи ефективність точного налаштування в узагальненні між наборами даних.

3.2 Показники ефективності

У цій кваліфікаційній роботі для оцінки ефективності моделі використовується кілька показників оцінки ефективності. Для аналізу ефективності виявлення вторгнень зазвичай використовуються такі метрики:

- істинно позитивний результат (TP): це кількість зразків атак, які правильно класифіковані як атаки;
- хибнопозитивний результат (FP): це стосується кількості доброякісних зразків, які помилково класифікуються як атаки;
- істинно негативний (TN): це вказує на кількість доброякісних зразків, які правильно класифіковані як доброякісні;
- хибнонегативний результат (FN): це кількість зразків атаки, які помилково класифіковані як доброякісні.

Точність: Цей показник вказує на частку правильно класифікованих екземплярів від загальної кількості прикладів. Він розраховується за допомогою

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.1}$$

Прецизійність (Precision): Вимірює співвідношення істинно позитивних прогнозів до загальної кількості позитивних прогнозів (як вірних, так і помилкових).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.2}$$

Відклик (Recall, або чутливість): Визначає частку істинно позитивних прогнозів серед усіх фактично позитивних прикладів.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.3}$$

F1-оцінка (F1-Score): Забезпечує баланс між прецизійністю та відкликом шляхом обчислення їхнього гармонійного середнього.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.4}$$

Для мультикласифікацій використовується макроусереднення для розрахунку повноти, точності та F1-оцінки для всіх класів. Оскільки набір даних незбалансований, макроусереднення гарантує, що кожен клас робить рівний внесок в оцінку, шляхом незалежного обчислення та усереднення метрик для кожного класу. Отже, загальний показник ефективності не зміщений до жодного конкретного класу. Усі метрики ефективності були обчислені за допомогою бібліотеки scikit-learn (sklearn) на Python. Ця бібліотека надає стандартизовані реалізації цих метрик, забезпечуючи узгодженість в оцінці моделі.

3.3 Налаштування параметрів

3.3.1 Локальні параметри моделі

Оскільки BiLSTM має вищі показники продуктивності, її обрано як локальну модель для нашої запропонованої напівдецентралізованої моделі FL. На продуктивність будь-якої глибокої нейронної мережі значно впливають її гіперпараметри. Зокрема, на модель BiLSTM впливають швидкість навчання, розмір пакета, коефіцієнт відсіву, функція активації, кількість шарів та кількість нейронів на шар. Таким чином, у цьому розділі ці гіперпараметри налаштовуються для досягнення найкращих результатів у моделі. Оскільки наша модель орієнтована на середовища Інтернету речей з обмеженими ресурсами, кількість шарів та кількість нейронів на шар налаштовуються для забезпечення балансу між продуктивністю та легкістю, де легкість вимірюється розміром моделі. Кожна конфігурація параметра застосовується до кожного запуску, результати усереднюються, а стандартне відхилення обчислюється, як представлено в таблицях 3.5-3.6. Після проведення парного t-тесту для порівняння показників продуктивності за різними параметрами моделі (m1, m2, m3 та m4), результати, представлені в таблиці 3.7, не вказують на статистично значущу різницю в продуктивності між моделями m1, m2 та m3. Однак, між m2 та m4 спостерігалася значна різниця, причому m2 продемонструвала кращу продуктивність. Враховуючи нашу головну мету – досягнення високої продуктивності при мінімізації складності моделі, ми обрали m2. Ця конфігурація, що містить лише два шари, забезпечує легку структуру, придатну для пристроїв Інтернету речей з обмеженими ресурсами, зберігаючи при цьому конкурентоспроможну продуктивність.

Таблиця 3.5 – Середні показники продуктивності для кожної конфігурації параметрів у BiLSTM

Параметри	Середня Accuracy	Середня Recall	Середня Precision	Середній F1-Score
m1: 1 шар (64 нейрони)	0,99	0,6741	0,7567	0,6935
m2: 2 шари (128, 64)	0,9909	0,6805	0,7948	0,7045
m3: 3 шари (128, 64, 32)	0,9896	0,6692	0,8143	0,6916
m4: 4 шари (128, 64, 64, 32)	0,9859	0,6664	0,7849	0,6861

Таблиця 3.6 – Стандартне відхилення показників ефективності для кожної конфігурації параметрів у BiLSTM

Параметри	Відхилення показників Точність	Відхилення показників Відкликання	Відхилення показників Precision	Відхилення показників F1-Оцінка
m1: 1 шар (64 нейрони)	0,0004	0,0026	0,0098	0,0021
m2: 2 шари (128, 64)	0,0001	0,0025	0,0064	0,0028
m3: 3 шари (128, 64, 32)	0,0014	0,0011	0,0196	0,0025
m4: 4 шари (128, 64, 64, 32)	0,0079	0,0017	0,0178	0,0034

Таблиця 3.7 – Результати парних t -тестів для порівняння продуктивності між різними конфігураціями параметрів у BiLSTM

Параметри	p -значення	Значення
m1 проти m2	0,1864	Несуттєво
m1 проти m3	0,4634	Несуттєво
m1 проти m4	0,8124	Несуттєво
m2 проти m3	0,8534	Несуттєво
m2 проти m4	0,0257	Значний
m3 проти m4	0,2029	Несуттєво

Після налаштування параметрів модель BiLSTM конфігурується з двома шарами (128 та 64 нейрони), знаючи, що вхідний шар має таку ж кількість ознак, як і CICIoT2023, тобто 46, а вихідний шар має таку ж кількість класів: 34, 8 та бінарні класи. Інші гіперпараметри були налаштовані після ретельних експериментів наступним чином: «ReLU» використовувалася як функція активації в прихованих шарах, а «Softmax» застосовувалася до вихідного шару як для моделей з 34 класами, так і для моделей з 8 класами. Однак для бінарної класифікації у вихідному шарі використовувалася функція «Sigmoid». Щоб запобігти перенавчанню, після кожного прихованого шару додавалося шар відсіву з коефіцієнтом 20%, випадковим чином ігноруючи 20% нейронів під час навчання. Як функція втрат використовувалася категоріальна перехресна ентропія для багатокласової класифікації та бінарна перехресна ентропія для бінарної класифікації. Усі експерименти проводилися протягом 1 епохи з розміром партії 64 та коефіцієнтом навчання 0,001, а як оптимізатор використовувався «adam».

3.3.2 Параметри FL

Для налаштувань федеративного навчання (FL) кількість клієнтів-

учасників встановлена на 10, а навчальний набір розділений між ними, що гарантує, що кожен клієнт має зразки з усіх типів атак, як показано в Листингу 2.1. Оскільки розподіл виконується випадковим чином, ми генеруємо п'ять різних розподілів клієнтів або «запусків». Всі експерименти виконуються на цих п'яти запусках, а результати усереднюються, щоб мінімізувати вплив випадковості. Встановлюємо кількість раундів FL на 10 і використовуємо FedAvg як алгоритм агрегації.

3.3.3 Алгоритм кластеризації

Як згадувалося вище, кластеризація виконується за допомогою k-середніх. Кількість кластерів встановлюється на 3 (k=3) на основі методу ліктя. На рисунку 3.2 відображено метод ліктя для визначення оптимальної кількості кластерів. Внутрішньо кластерна помилка (WCE) різко зменшується до k=3, після чого покращення сповільнюється, що свідчить k=3 як оптимальна кількість кластерів.

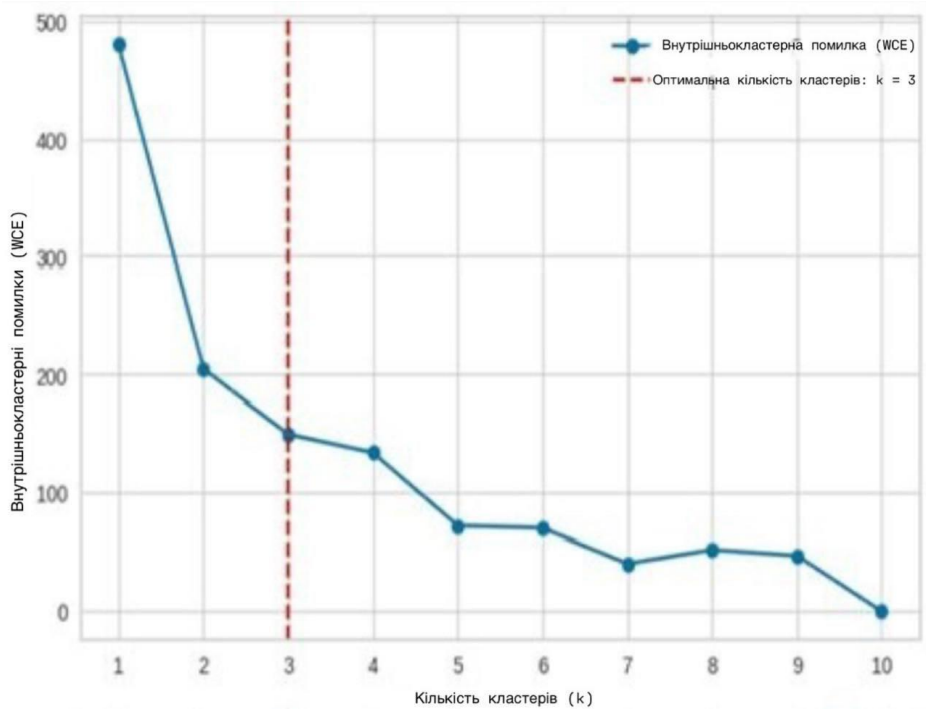

Рисунок 3.2 – Графік методу ліктя, що показує залежність внутрішньокластерної помилки (WCE) від кількості кластерів (k)

Кластеризація за методом k-середніх базується на мірах відстані; таким чином, їй потрібно знайти найкращу міру відстані, яка підходить для нашого випадку. Обираємо чотири метрики відстані: евклідову, середньоевклідову, манхеттенську та коефіцієнт дивергенції. Евклідову обрано, оскільки вона є метрикою за замовчуванням для кластеризації за методом k-середніх, незважаючи на її чутливість до викидів. Тому середньоевклідова використовується для управління викидами, тоді як манхеттенська міра відстані демонструє мінімальне спотворення. З іншого боку, коефіцієнт дивергенції виявляється найефективнішим методом для обробки високовимірних даних. Кластеризація за методом k-середніх застосовується з використанням кожної з чотирьох мір відстані; потім результати порівнюються за допомогою показника Silhouette, щоб вибрати найкращу міру відстані. Оскільки кластеризація за методом k-середніх є випадковим алгоритмом, де кожен прогін дає новий кластер, кластеризація за методом k-середніх застосовується 10 разів, а потім обчислюється середній показник Silhouette для порівняння між кожною з мір відстані. Згідно з експериментальними результатами, k-середніх з відстанню Манхеттена досягає найвищого балу за шкалою Silhouette. В таблиці 3.8 нижче наведено бали за шкалою силуету, отримані при застосуванні кластеризації k-середніх з різними мірами відстані.

Таблиця 3.8 – Бали силуету для різних вимірювань відстані

Вимірювання відстані	Силуетний бал
Евклідова	0,2654
Середній евклідіва	0,1823
Манхеттен	0,2777
Коефіцієнт дивергенції	0,14398

3.4 Оцінювання ефективності

У цьому розділі представлено результати оцінювання запропонованої моделі. Спочатку порівнюється напівдецентралізований підхід FL з централізованим підходом FL. Далі напівдецентралізований підхід FL оцінюється за допомогою трьох методів DL: LSTM, BiLSTM та WGAN.

3.4.1 Централізоване та напівдецентралізоване FL з використанням LSTM

Щоб продемонструвати ефективність напівдецентралізованої FL, ми порівнюємо її з централізованою FL. Обидва підходи налаштовані з LSTM як локальною моделлю. Обидва підходи застосовуються на п'яти прогонах, а потім результати усереднюються. Згідно з результатами, напівдецентралізована модель на основі FL перевершує централізовану модель на основі FL у 34 та 8 класах, як показано на рисунку 3.3. Однак централізована модель на основі FL має вищі показники продуктивності для бінарної класифікації. Це пояснюється тим, що в бінарній класифікації дисбаланс класів менший, ніж у мультикласифікації. Це також демонструє, що напівдецентралізований підхід є ефективнішим для обробки дисбалансу класів, ніж централізована FL.

Більше того, напівдецентралізована модель на основі FL зменшує накладні витрати на зв'язок, дозволяючи головному вузлу кластера взаємодіяти із сервером, що призводить до збереження споживання ресурсів (пропускної здатності), а також до покращення часу очікування. Запропонований підхід до кластеризації вибирає головний вузол кластера на основі середньої відстані до сусідніх вузлів у кластері, прагнучи ефективно отримувати ваги та параметри без затримки.

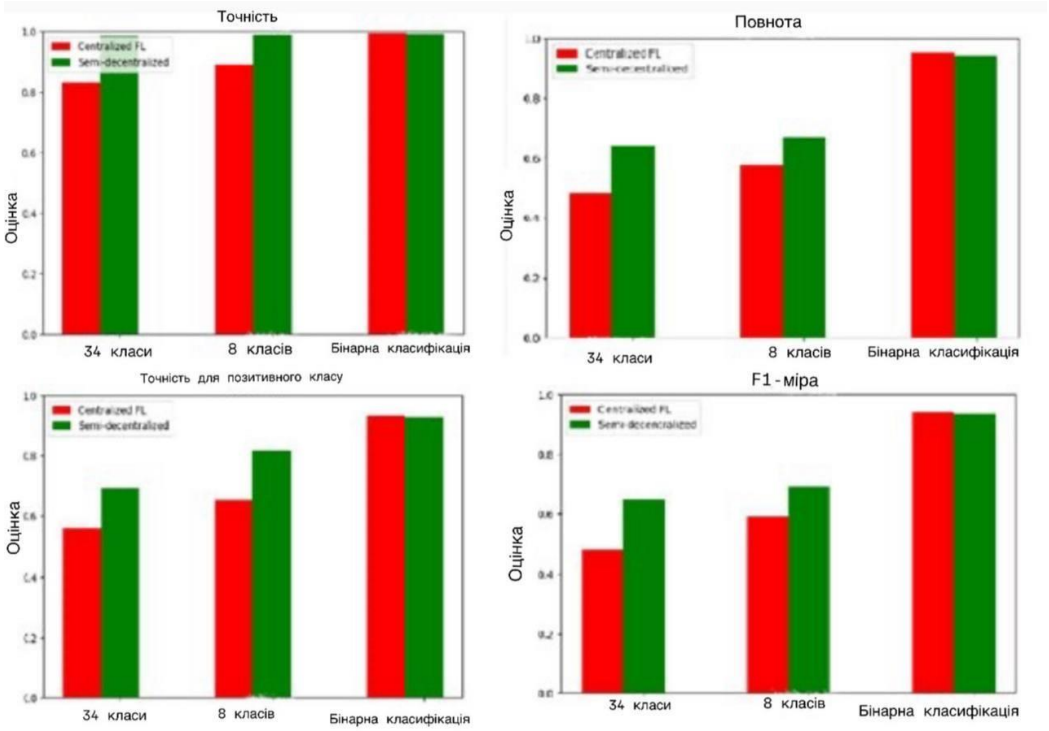

Рисунок 3.3 – Порівняння продуктивності централізованого та напівдецентралізованого підходу до FL

Споживання ресурсів можна виміряти часом навчання на раунд FL. Для централізованого підходу FL час навчання на один раунд становить 4398,4525 с, а для напівдецентралізованого підходу FL – 3399,985 с. Отже, напівдецентралізований підхід FL сприяє збереженню приблизно 1000 с часу навчання на раунд FL, тим самим зберігаючи загальний час навчання наприкінці процесу навчання FL. Це пов'язано з механізмом кластеризації, який дозволяє кожному кластеру надсилати свої оновлення на сервер, не чекаючи на інші кластери. В результаті час навчання мінімізується, що дійсно допомагає зменшити споживання ресурсів. На рисунку 3.4 показано порівняння часу навчання на раунд FL для централізованого та напівдецентралізованого підходів, знаючи, що обидві моделі збігаються на раунді 4.

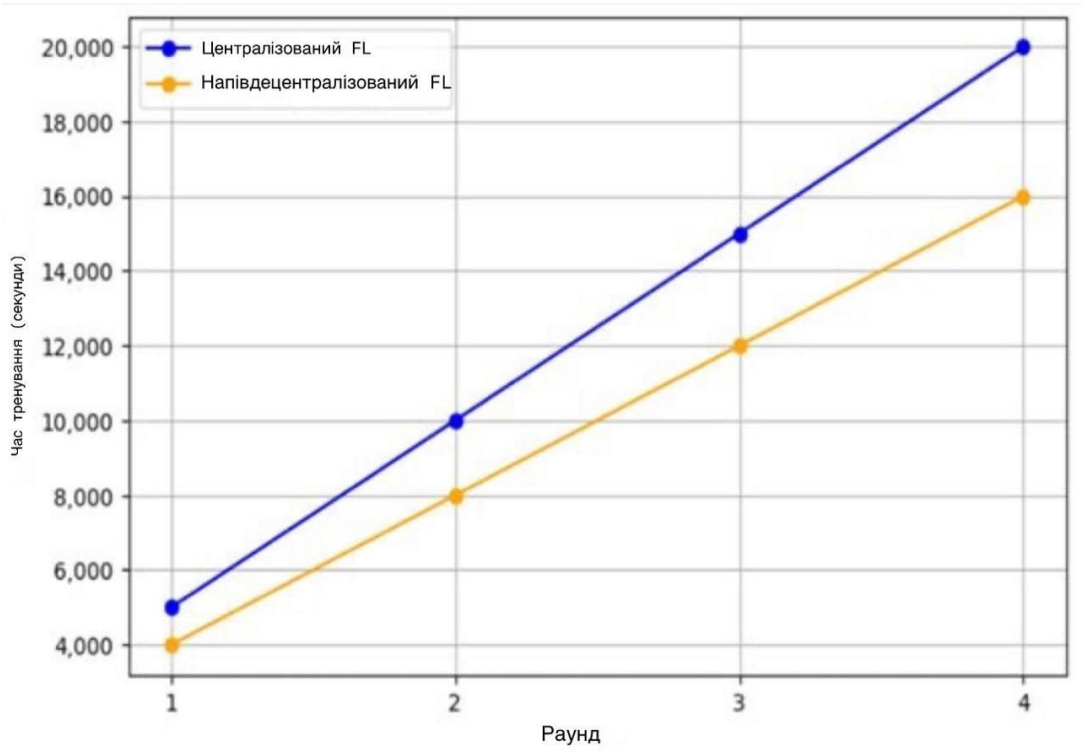

Рисунок 3.4 – Час навчання на раунд FL для централізованого та напівдецентралізованого підходу

3.4.2 Напівдецентралізована FL з різними моделями DL

Оскільки напівдецентралізоване FL перевершує централізоване FL, цей розділ зосередиться на покращенні продуктивності архітектури напівдецентралізованого FL шляхом спроби інших моделей DL. Як згадувалося раніше, існує п'ять різних розподілів клієнтів, відомих як прогони. Процес FL застосовується до всіх прогонів, результати усереднюються, а також обчислюється стандартне відхилення. Модель застосовується до завдань класифікації з 34 класами, 8 класами та бінарною класифікацією. Використовуються три моделі глибокого навчання (DL): LSTM, BiLSTM та WGAN. На рисунку 3.5 та таблиці 3.8 нижче показано результати застосування трьох моделей DL: LSTM, BiLSTM та WGAN як локальних моделей для напівдецентралізованого підходу FL. Результати показують, що BiLSTM має найвищі показники продуктивності.

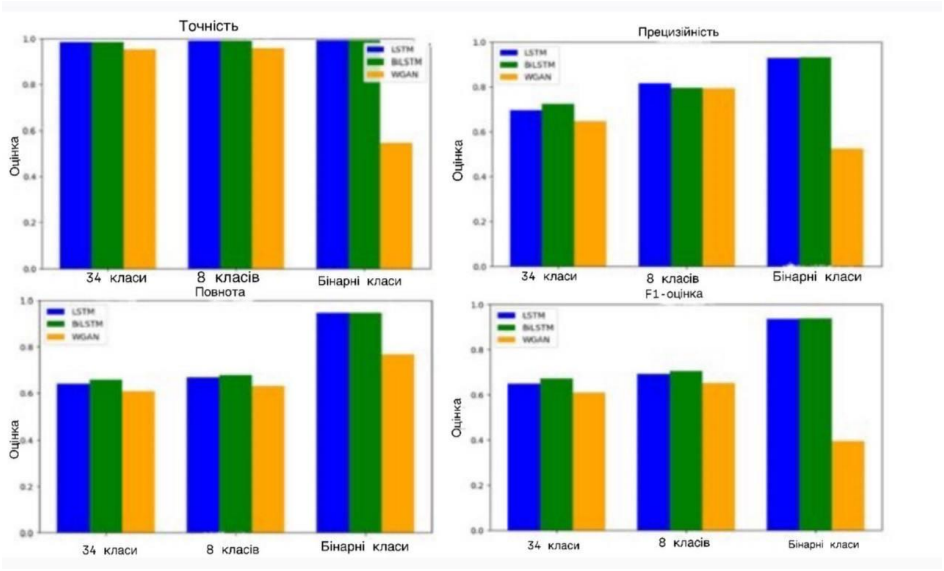

Рисунок 3.5 – Порівняння напівдецентралізованих моделей FL (LSTM, BiLSTM та WGAN) для різних категорій класів

Таблиця 3.8 – Порівняння класифікаційних метрик для напівдецентралізованих моделей FL

Метрики	Заняття	LSTM	BiLSTM	WGAN
Accuracy	34-класи	0,9843	0,9855	0,9516
	8-класи	0,9902	0,9909	0,9569
	Бінарні класи	0,9942	0,9943	0,5464
Recall	34-класи	0,6409	0,6602	0,6095
	8-класи	0,6701	0,6805	0,6309
	Бінарні класи	0,9467	0,9474	0,7673
Precision	34-класи	0,6959	0,7227	0,6474
	8-класи	0,8158	0,7948	0,7927
	Бінарні класи	0,9293	0,9307	0,5246
F1-Score	34-класи	0,6492	0,6731	0,6099
	8-класи	0,6931	0,7054	0,6511
	Бінарні класи	0,9378	0,9389	0,3957

Коли для дослідження статистично значущої різниці застосовується парний t-критерій, результати показують, що існує значна різниця в продуктивності між усіма моделями за всіма показниками, окрім точності. У

таблиці 3.9 наведено результати застосування парного t-критерію.

Таблиця 3.9 – Результати парного t-тесту всіх значень метрик на запропонованій моделі з використанням трьох методів DL

Метрика	Порівняння	*p*-значення	Значення
Accuracy	BiLSTM / LSTM	0,0054	Значний
	BiLSTM / WGAN	0,0443	Значний
	LSTM / WGAN	0,0470	Значний
Recall	BiLSTM / LSTM	0,0004	Значний
	BiLSTM / WGAN	0,0047	Значний
	LSTM / WGAN	0,0097	Значний
Precision	BiLSTM / LSTM	0,0562	Несуттєво
	BiLSTM / WGAN	0,8959	Несуттєво
	LSTM / WGAN	0,1430	Несуттєво
F1-Score	BiLSTM / LSTM	0,0019	Значний
	BiLSTM / WGAN	0,0017	Значний
	LSTM / WGAN	0,0032	Значний

ВИСНОВКИ

В роботі запропоновано напівдецентралізовану модель на основі FL, яка є легким механізмом виявлення вторгнень з урахуванням неоднорідності, який підходить для розгортання в мережах IoT. Запропонована модель базується на кластеризації клієнтів FL, що вирішує неоднорідність даних, що, у свою чергу, покращує процес навчання, а також зменшує накладні витрати на зв'язок завдяки роботі на рівні кластера замість рівня клієнта.

Запропонована модель оцінюється за допомогою набору даних CICIOT2023, оскільки вона містить велику кількість трас IoT і категорій атак, зокрема DDoS, оскільки це наша цільова атака. Три методи DL оцінюються як локальні моделі в напівдецентралізованому підході FL, а саме LSTM, BiLSTM і WGAN. Результати показують, що BiLSTM досягає найкращої продуктивності; таким чином, він обраний як локальна модель у запропонованій моделі. Після тривалих експериментів BiLSTM із двома шарами, кожен із 128 і 64 нейронами відповідно, забезпечує найкращі результати з точки зору продуктивності, а легкість робить його доступним для IoT з обмеженими ресурсами. Попередньо підготовлену напівдецентралізовану модель FL було додатково перевірено на трьох додаткових наборах даних IoT – BoT-IoT, WUSTL-IIoT-2021 і Edge-IIoTset – щоб підтвердити її можливість узагальнення. Результати показують, що запропонована модель досягає найвищих показників продуктивності в більшості класів, з винятковою продуктивністю у виявленні DDoS-атак.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Stallings, W.; Brown, L. Computer Security: Principles and Practice, 4th ed.; Pearson: London, UK, 2018; pp. 274–299.

2. Krontiris, I.; Giannetsos, T.; Dimitriou, T. LIDeA: A Distributed Lightweight Intrusion Detection Architecture for Sensor Networks. In Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, Istanbul, Turkey, 22–25 September 2008; pp. 1–10.

3. Zhurylo, O., Liashenko, O., & Avetisova, K. (2023). Hardware security overview of Fog computing end devices in the Internet of Things. Innovative technologies and scientific solutions for industries, (1 (23), 57–71. https://doi.org/10.30837/ITSSI.2023.23.057

4. Derhab, A.; Aldweesh, A.; Emam, A.Z.; Khan, F.A.; Wang, X. Intrusion Detection System for Internet of Things Based on Temporal Convolution Neural Network and Efficient Feature Engineering. Wirel. Commun. Mob. Comput. 2020, 2020, 6689134.

5. Ahmad, R.; Wazirali, R.; Abu-Ain, T. Machine Learning for Wireless Sensor Networks Security: An Overview of Challenges and Issues. Sensors 2022, 22, 4730.

6. О.С. Ляшенко, І.А. Великодний, В.Г. Знайдюк, О.Д. Журило. Модель та методи виявлення широкомасштабної атаки в середовищі IoT / Системи управління, навігації та зв'язку. Том 1 № 75 (2024), С.127-132

7. Campos, E.M.; Saura, P.F.; González-Vidal, A.; Hernández-Ramos, J.L.; Bernal Bernabé, J.; Baldini, G.; Skarmeta, A. Evaluating Federated Learning for Intrusion Detection in Internet of Things: Review and Challenges. Comput. Netw. 2022, 203, 108661.

8. Agrawal, S.; Sarkar, S.; Aouedi, O.; Yenduri, G.; Piamrat, K.; Alazab, M.; Bhattacharya, S.; Maddikunta, P.K.R.; Gadekallu, T.R. Federated Learning for Intrusion Detection System: Concepts, Challenges and Future Directions. Comput.

Commun. 2022, 195, 346–361.

9. Savazzi, S.; Nicoli, M.; Rampa, V. Federated Learning with Cooperating Devices: A Consensus Approach for Massive IoT Networks. IEEE Internet Things J. 2020, 7, 4641–4654.

10. Журило, О. і Ляшенко, О. (2024) «Архітектура та системи безпеки IoT на основі туманних обчислень», СУЧАСНИЙ СТАН НАУКОВИХ ДОСЛІДЖЕНЬ ТА ТЕХНОЛОГІЙ В ПРОМИСЛОВОСТІ, (1(27), с. 54–66. doi: 10.30837/ITSSI.2024.27.054.

11. Yang, Z.; Shi, Y.; Zhou, Y.; Wang, Z.; Yang, K. Trustworthy Federated Learning via Blockchain. IEEE Internet Things J. 2023, 10, 92–109

12. Sun, Y.; Shao, J.; Mao, Y.; Wang, J.H.; Zhang, J. Semi-Decentralized Federated Edge Learning for Fast Convergence on Non-IID Data. In Proceedings of the 2022 IEEE Wireless Communications and Networking Conference (WCNC), Austin, TX, USA, 10–13 April 2022; pp. 1898–1903.

13. Wu, W.; He, L.; Lin, W.; Mao, R.; Maple, C.; Jarvis, S. SAFA: A Semi-Asynchronous Protocol for Fast Federated Learning with Low Overhead. IEEE Trans. Comput. 2021, 70, 655–668.

14. Liu, L.; Zhang, J.; Song, S.H.; Letaief, K.B. Client-Edge-Cloud Hierarchical Federated Learning. In Proceedings of the ICC 2020—2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–6.

15. Ghimire, B.; Rawat, D.B. Recent Advances on Federated Learning for Cybersecurity and Cybersecurity for Federated Learning for Internet of Things. IEEE Internet Things J. 2022, 9, 8229–8249.

16. Khraisat, A.; Alazab, A. A Critical Review of Intrusion Detection Systems in the Internet of Things: Techniques, Deployment Strategy, Validation Strategy, Attacks, Public Datasets, and Challenges. Cybersecurity 2021, 4, 18.

17. Tran, D.H.; Nguyen, V.L.; Utama, I.B.K.Y.; Jang, Y.M. An Improved Sensor Anomaly Detection Method in IoT System using Federated Learning. In Proceedings of the International Conference on Ubiquitous and Future Networks

(ICUFN), Barcelona, Spain, 5–8 July 2022; pp. 466–469.

18. Tahir, M.; Ali, M.I. On the Performance of Federated Learning Algorithms for IoT. IoT 2022, 3, 273–284.

19. Lee, S.J.; Yoo, P.D.; Asyhari, A.T.; Jhi, Y.; Chermak, L.; Yeun, C.Y.; Taha, K. IMPACT: Impersonation Attack Detection via Edge Computing Using Deep Autoencoder and Feature Abstraction. IEEE Access 2020, 8, 65520–65529.

20. Nguyen, X.H.; Nguyen, X.D.; Huynh, H.H.; Le, K.H. Realguard: A Lightweight Network Intrusion Detection System for IoT Gateways. Sensors 2022, 22, 432.

21. Huong, T.T.; Bac, T.P.; Long, D.M.; Thang, B.D.; Binh, N.T.; Luong, T.D.; Phuc, T.K. LocKedge: Low-Complexity Cyberattack Detection in IoT Edge Computing. IEEE Access 2021, 9, 29696–29710.

22. Rashid, M.M.; Khan, S.U.; Eusufzai, F.; Redwan, M.A.; Sabuj, S.R.; Elsharief, M. A Federated Learning-Based Approach for Improving Intrusion Detection in Industrial Internet of Things Networks. Network 2023, 3, 158–179.

23. Hamdi, N. Federated Learning-Based Intrusion Detection System for Internet of Things. Int. J. Inf. Secur. 2023, 22, 1937–1948.

24. Zhang, T.; He, C.; Ma, T.; Gao, L.; Ma, M.; Avestimehr, S. Federated Learning for Internet of Things. In Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems, Coimbra, Portugal, 15–17 November 2021; pp. 413–419.

25. Regan, C.; Nasajpour, M.; Parizi, R.M.; Pouriyeh, S.; Dehghantanha, A.; Choo, K.-K.R. Federated IoT Attack Detection Using Decentralized Edge Data. Mach. Learn. Appl. 2022, 8, 100263.

26. Attota, D.C.; Mothukuri, V.; Parizi, R.M.; Pouriyeh, S. An Ensemble Multi-View Federated Learning Intrusion Detection for IoT. IEEE Access 2021, 9, 117734–117745.

27. Driss, M.; Almomani, I.; Huma, Z.E.; Ahmad, J. A Federated Learning Framework for Cyberattack Detection in Vehicular Sensor Networks. Complex Intell. Syst. 2022, 8, 4221–4235.

28. Ляшенко О.С., Знайдюк В.Г., Журило О.Д., Риков В.А. (2025) Напівдецентралізована модель федеративного навчання для виявлення вторгнень в IoT // Таврійський науковий вісник. Серія: Технічні науки. №2