ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

каф. ЕОМ

Модель федеративного навчання для виявлення вторгнень в IoT

Ст. групи СПм-23-3
Риков В.А.

Керівник
Ляшенко О.С.

2025

Актуальність роботи

- 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 роботи 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 – призначення голови кластера кожному кластеру, який діє від імені клієнтів FL.

Мета та задачі кваліфікаційної роботи

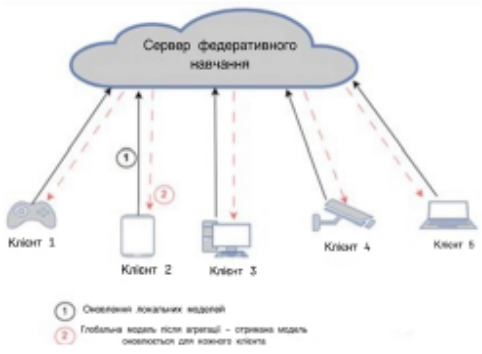
Мета кваліфікаційної роботи розробка моделі федеративного навчання для виявлення вторгнень в IoT

Задачі роботи:

- Провести аналіз архітектур федеративного навчання в системах виявлення вторгнень
- Запропонувати напівдецентралізовану модель FL
- Створити програмну реалізацію та протестувати запропоновану модель на наборі даних CICIOT2023

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.
- Тренувальний процес складається з кількох раундів. 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

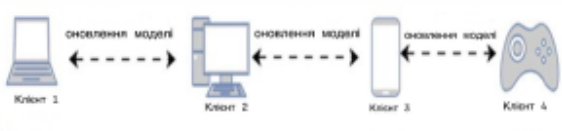


Архітектура централізованого FL (клієнт-сервер)

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.

У мережах 6G очікується, що пристрої будуть з'єднані між машинами (M2M), що робить зв'язок D2D перспективною технологією для децентралізованої FL.
На рисунку показано децентралізовану архітектуру FL.



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. Голова кластера (ГК) вибирається для кожного кластера на основі усередненого балу.

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.

В роботі представлено реалізований алгорим на Python, який узагальнює напівдецентралізовану модель.

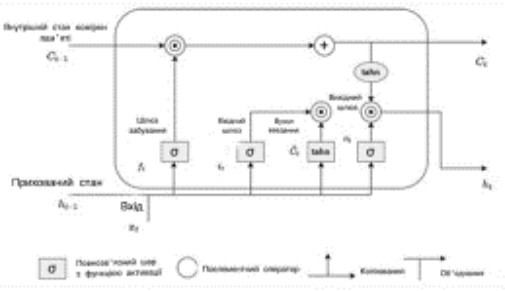
Напівдецентралізована архітектура FL

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 тривалішних 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.

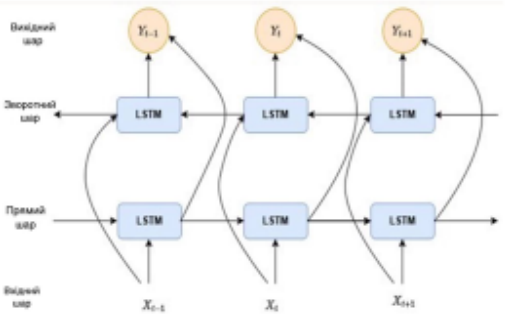
Рішення про збереження або видалення інформації залежить від ваг, призначених під час навчання, що дозволяє моделі дізнатися, які деталі варто зберегти або відкинути. Модель LSTM зазвичай складається з трьох вентилів: вентиля забуття, вхідного вентиля та вихідного вентиля, як показано на рисунку.



Двонаправлена довготривала короткочасна пам'ять (BiLSTM)

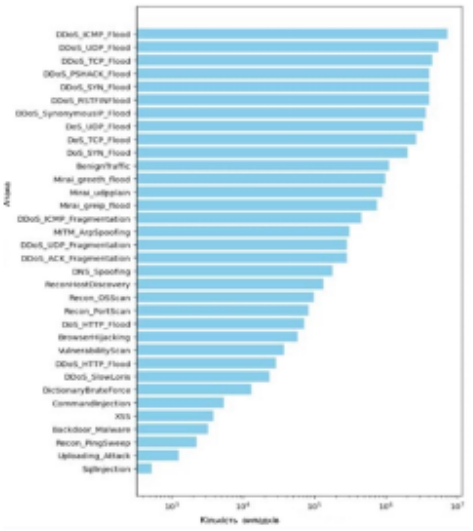
BiLSTM – це розширення моделі LSTM, де до вхідних даних застосовуються дві моделі LSTM. На відміну від стандартної мережі LSTM, яка використовує лише інформацію, з якою вона вже зіткнулася в послідовності, архітектура BiLSTM включає два шари LSTM – один обробляє вхідну послідовність вперед (прямий LSTM), а інший обробляє її назад (зворотний LSTM), як показано на рисунку.

Таке подвійне застосування LSTM значно розширює можливості моделі вивчати довгострокові залежності, що зрештою призводить до покращення продуктивності.



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 показано кількість екземплярів для кожної категорії атак.



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.

Типи атак та кількість зразків у наборі даних BoT-IoT

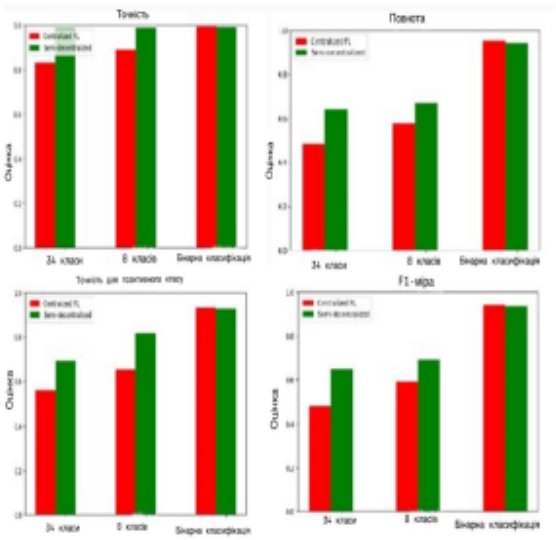
Типи атак	Кількість зразків
DoS-HTTP	1485
DoS-TCP	615800
DoS-UDP	1 032 975
DDoS-HTTP	989
DDoS-TCP	977380
DDoS-UDP	948255
Зняття відбитків ОС	17914
Сканування сервера	75168
Кейлоггер	73
Крадіжка даних	6
Звичайний	477

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	3200626
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

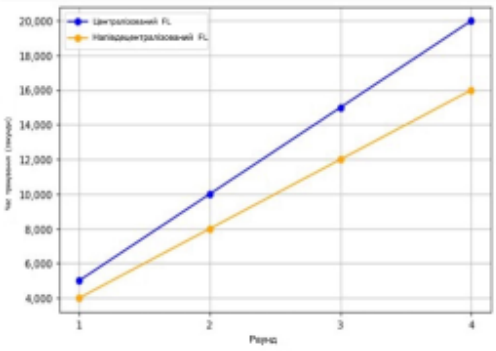
Централізоване та напівдецентралізоване FL з використанням LSTM

- 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.
- 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.



Час навчання на раунд FL для централізованого та напівдецентралізованого підходу.

- 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 показано порівняння часу навчання на раунд FL для централізованого та напівдецентралізованого підходів, знаючи, що обидві моделі збігаються на раунді 4.



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.
- 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 дві моделі глибокого навчання (DL): LSTM, BiLSTM.
- В таблиці показано результати застосування двох моделей DL: LSTM, BiLSTM, як локальних моделей для напівдецентралізованого підходу FL. Результати показують, що BiLSTM має найвищі показники продуктивності.

Метрики	Заняття	LSTM	BiLSTM
Accuracy	34-класи	0,9845	0,9855
	8-класи	0,9902	0,9909
	Бінарні класи	0,9942	0,9943
Recall	34-класи	0,6409	0,6602
	8-класи	0,6701	0,6805
	Бінарні класи	0,9467	0,9474
Precision	34-класи	0,6959	0,7227
	8-класи	0,8158	0,7948
	Бінарні класи	0,9295	0,9307
F1-Score	34-класи	0,6492	0,6751
	8-класи	0,6931	0,7054
	Бінарні класи	0,9578	0,9389

Порівняння класифікаційних метрик для напівдецентралізованих моделей FL

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.

- Результати дослідження подані в фаховий журнал категорії Б «Таврійський науковий вісник. Серія: Технічні науки» у вигляді наукової статті Ляшенко О.С., Знайдюк В.Г., Журило О.Д., Риков В.А. Напівдецентралізована модель федеративного навчання для виявлення вторгнень в IoT (дата подачі 08.06.2025)

ДОДАТОК Б
РЕАЛІЗОВАНИЙ АЛГОРИМ, ЯКИЙ УЗАГАЛЬНЮЄ НАПІВДЕЦЕНТРАЛІЗОВАНУ МОДЕЛЬ

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

class ClusterBasedFL:
    def __init__(self, initial_model, n_rounds=10, n_epochs=3):
        """
        Initialize Cluster-Based Federated Learning.

        Args:
            initial_model: Compiled Keras model to use as base
            n_rounds: Maximum number of FL rounds
            n_epochs: Number of local epochs per round
        """
        self.global_model = initial_model
        self.W_G = initial_model.get_weights()
        self.n_rounds = n_rounds
        self.n_epochs = n_epochs
        self.cluster_heads = []
        self.cluster_weights = []
        self.performance_history = []

    def train(self, clients: List[Dict[str, Any]], validate_set:
tuple):
        """
        Run cluster-based federated learning.

        Args:
            clients: List of clients with 'features' and
'labels'
            validate_set: Validation data (X_val, y_val)
        """
        # Step 2: Cluster clients (using previous
implementation)
        clusters = self._cluster_clients(clients)
        n_clusters = len(clusters)

        # Initialize cluster heads and weights
        self.cluster_heads = [None] * n_clusters
        self.cluster_weights = [None] * n_clusters

        # Step 4-15: Select cluster heads based on silhouette
scores
        self._select_cluster_heads(clients, clusters)
```

```
        # Federated training rounds
        for round in range(self.n_rounds):
            print(f"Starting FL round
{round+1}/{self.n_rounds}")

            # Reset cluster weights for new round
            self.cluster_weights = [None] * n_clusters

            # Step 17-21: Train each cluster head and aggregate
weights
            for cluster_idx in range(n_clusters):
                head_client_idx =
self.cluster_heads[cluster_idx]
                head_client = clients[head_client_idx]

                # Clone and train model on cluster head
                cluster_model = clone_model(self.global_model)
                cluster_model.set_weights(self.W_G)
                cluster_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy')

                cluster_model.fit(
                    head_client['features'],
                    head_client['labels'],
                    epochs=self.n_epochs,
                    verbose=0
                )

                # Store updated weights
                self.cluster_weights[cluster_idx] =
cluster_model.get_weights()

            # Step 23-25: Aggregate cluster weights
            self._aggregate_weights(n_clusters)

            # Update global model
            self.global_model.set_weights(self.W_G)

            # Step 27: Evaluate performance
            X_val, y_val = validate_set
            loss, accuracy = self.global_model.evaluate(X_val,
y_val, verbose=0)
            self.performance_history.append(accuracy)

            print(f"Round {round+1} - Validation Accuracy:
{accuracy:.4f}")

            # Step 28: Early stopping check
            if self._check_convergence():
                print("Model converged - stopping training")
                break

        return self.W_G
```

```
    def _cluster_clients(self, clients):
        """Cluster clients using previous implementation"""
        # Here we would use the clustering algorithm from
alg2.py
        # For simplicity, we'll do random clustering
        n_clients = len(clients)
        n_clusters = max(2, n_clients // 5)  # Simple heuristic
        clusters = [[] for _ in range(n_clusters)]

        for client_idx in range(n_clients):
            cluster_idx = client_idx % n_clusters
            clusters[cluster_idx].append(client_idx)

        return clusters

    def _select_cluster_heads(self, clients, clusters):
        """Select cluster heads based on silhouette scores"""
        # Extract features for silhouette calculation
        features = np.array([client['features'].mean(axis=0) for
client in clients])

        for cluster_idx, cluster in enumerate(clusters):
            if len(cluster) == 0:
                continue

            # Calculate silhouette scores for clients in this
cluster
            cluster_features = features[cluster]
            if len(cluster) > 1:
                distances = pairwise_distances(cluster_features,
metric='euclidean')
                scores = silhouette_samples(distances,
np.zeros(len(cluster)))  # All in same cluster
            else:
                scores = [0.0]  # Single client case

            avg_score = np.mean(scores)

            # Find client closest to average score
            closest_idx = np.argmin(np.abs(scores - avg_score))
            self.cluster_heads[cluster_idx] =
cluster[closest_idx]

    def _aggregate_weights(self, n_clusters):
        """Aggregate cluster weights to update global model"""
        # Simple average aggregation
        avg_weights = []

        for layer_idx in range(len(self.W_G)):
            layer_weights = []
            for cluster_w in self.cluster_weights:
                if cluster_w is not None:
```

```
                    layer_weights.append(cluster_w[layer_idx])

            # Average weights for this layer across clusters
            avg_layer = np.mean(layer_weights, axis=0)
            avg_weights.append(avg_layer)

        self.W_G = avg_weights

    def _check_convergence(self):
        """Check if model performance has converged"""
        if len(self.performance_history) < 2:
            return False

        # Check if improvement is below threshold
        improvement = abs(self.performance_history[-1] -
self.performance_history[-2])
        return improvement < 1e-5  # 0.00001 threshold
```