МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ПАВЛОВ МИКОЛА СЕРГІЙОВИЧ

Допускається до захисту:
в. о. завідувача кафедри
прикладної математики та
кібербезпеки,
д-р філ. з матем.
_____ Луценко А. В.
«____» _____ 20__ р.

РОЗРОБКА СПОСОБУ РОЗПІЗНАВАННЯ КІБЕРЗАГРОЗ У ВЕЛИКОМАСШТАБНИХ МЕРЕЖЕВИХ СИСТЕМАХ НА ОСНОВІ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ З ДОВГОЮ КОРОТКОТРИВАЛОЮ ПАМ'ЯТТЮ

Спеціальність 113 Прикладна математика

Кваліфікаційна (магістерська) робота

Науковий керівник:
Загоруйко Л.В., доцент кафедри
прикладної математики та кібербезпеки,
к.т.н, доцент
_____
підпис

Оцінка:___/ ___/ _____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2024

АНОТАЦІЯ

Павлов М. С. У цій магістерській роботі розглядається проблематика розробки та застосування рекурентної нейронної мережі з довгою короткочасною пам'яттю для ідентифікації кіберзагроз у великомасштабних адаптивних мережевих топологіях. В роботі проведено глибокий аналіз адаптивних мережевих топологій та використання рекурентних нейронних мереж у контексті кібербезпеки.

Розроблена концептуальна модель включає в себе створення, тренування та тестування прототипу нейронної мережі, орієнтованої на виявлення та класифікацію різних типів змін у мережевих топологіях, включаючи стандартні адаптації та кібератаки. Реалізація прототипу виконана з використанням Python та набору бібліотек, таких як TensorFlow та Keras, у середовищі Google Colab, що забезпечує доступ до потужних обчислювальних ресурсів.

Важливою частиною роботи є підготовка та аналіз дата сету, який складається з 20 000 json файлів, представляючи різні сценарії в мережевій топології. Модель була протестована та валідована, показавши середній показник точності приблизно 77%, що вказує на її ефективність у виявленні різних типів кіберзагроз.

Результати дослідження підкреслюють потенціал рекурентних нейронних мереж для забезпечення кібербезпеки великомасштабних мережевих систем. Робота надає детальний огляд процесу розробки та аналізу нейронної мережі, а також обговорює можливості подальшого розвитку та поліпшення моделі.

Ключові слова: рекурентні нейронні мережі, кібербезпека, адаптивні мережеві топології, LSTM, дата сет, машинне навчання, кіберзагрози, Python, TensorFlow, Keras, Google Colab.

66 с., 11 рис., 3 дод., 30 джерел.

ABSTRACT

Pavlov M. S. This master's thesis examines the issue of developing and applying a recurrent neural network with long short-term memory for the identification of cyber threats in large-scale adaptive network topologies. The work conducts an in-depth analysis of adaptive network topologies and the use of recurrent neural networks in the context of cybersecurity.

The developed conceptual model includes the creation, training, and testing of a neural network prototype, focused on detecting and classifying various types of changes in network topologies, including standard adaptations and cyberattacks. The implementation of the prototype was carried out using Python and a set of libraries such as TensorFlow and Keras, in the Google Colab environment, which provides access to powerful computational resources.

An important part of the work is the preparation and analysis of the dataset, which consists of 20,000 json files, representing various scenarios in the network topology. The model was tested and validated, showing an average accuracy of approximately 77%, indicating its effectiveness in detecting different types of cyber threats.

The research results highlight the potential of recurrent neural networks in ensuring cybersecurity of large-scale network systems. The work provides a detailed overview of the process of developing and analyzing the neural network, and also discusses the possibilities for further development and improvement of the model.

Keywords: recurrent neural networks, cybersecurity, adaptive network topologies, LSTM, dataset, machine learning, cyber threats, Python, TensorFlow, Keras, Google Colab.

66 pp., 11 figures, 3 appendices, 30 sources.

ЗМІСТ

ВСТУП 6
РОЗДІЛ І. ТЕОРЕТИЧНІ АСПЕКТИ СПОСОБУ РОЗПІЗНАВАННЯ КІБЕРЗАГРОЗ У ВЕЛИКОМАСШТАБНИХ МЕРЕЖЕВИХ СИСТЕМАХ НА ОСНОВІ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ З ДОВГОЮ КОРОТКОТРИВАЛОЮ ПАМ'ЯТТЮ 10
1.1 Вступ до проблематики кіберзагроз великомасштабних систем. 10
1.1.1 Захист мережевих топологій від кіберзагроз 11
1.2 Адаптивні мережеві топології великомасштабних систем 12
1.2.1 Огляд існуючих архітектур і топологій 13
1.3 Рекурентні нейронні мережі та їх застосування для розпізнавання кіберзагроз 17
1.4 Порівняльний аналіз існуючих рішень захисту мережевих топологій за допомогою нейронних мереж 18
РОЗДІЛ ІІ. РОЗРОБКА КОНЦЕПЦІЇ АДАПТИВНОЇ МЕРЕЖЕВОЇ ТОПОЛОГІЇ ТА РЕКУРРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ 22
2.1. Опис адаптивної мережевої топології. 22
2.2 Розробка концептуальної моделі рекурентної нейронної мережі 25
2.2.1 Пропонований підхід для розпізнавання кіберзагроз на адаптивну мережеву топологію 26
2.2.2 Структура рекурентної нейронної мережі з довгою короткотривалою пам'яттю 28
2.2.3 Опис методу навчання нейронної мережі 32
РОЗДІЛ ІІІ. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОТОТИПУ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ 36
3.1 Підготовка дата сету 36
3.1.1 Структура дата сету 37
3.1.2 Алгоритм генерації 38
3.2 Розробка програмного прототипу 49
3.2.1 Вибір програмних інструментів 49

3.2.2 Опис алгоритмів та обчислювального процесу 51
3.3 Тестування та валідація моделі 57
3.4 Висновки та пропозиції щодо подальшого вдосконалення 58
3.4.1 Перспективи розвитку та можливі поліпшення 58
3.4.2 Застосування моделі в реальних умовах 59
ВИСНОВКИ 62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ 64
ДОДАТКИ 67
Додаток А. Приклади дата сету 67
Додаток Б. Лістинг програми генерації дата сету 74
Додаток В. Лістинг нейромережі 80

ВСТУП

Актуальність: У сучасному світі, де кіберпростір стає все більш складним і взаємопов'язаним, захист великомасштабних адаптивних мережевих топологій є надзвичайно актуальним. Ці мережі, що часто використовуються у великому бізнесі, державних сервісах та військовій інфраструктурі, є динамічними та складними для моніторингу людиною через постійно змінюючу структуру та велику кількість вузлів. Традиційні алгоритми не можуть ефективно виявляти нові закономірності або адаптуватися до швидко змінних шаблонів поведінки, особливо в контексті постійно прогресуючих кібератак. Рекурентні нейронні мережі з довгою короткотривалою пам'яттю (LSTM) стають вирішенням цих викликів, оскільки вони можуть зберігати інформацію про стан мережі протягом тривалого часу, дозволяючи аналізувати динамічні зміни та виявляти складні шаблони кіберзагроз. Як зазначено в роботах [1] та [3], незважаючи на те, що існуючі нейромережеві підходи показали перспективність, їх адаптованість до нових і непередбачених типів кіберзагроз залишається ключовою сферою для подальшого розвитку.

Огляд [2] підкреслює проблеми, пов’язані з обробкою даних у реальному часі та масштабованістю нейронних мереж у великих і динамічних мережевих середовищах. Розробка рішень, які можуть ефективно обробляти величезні обсяги даних у режимі реального часу та адаптуватися до постійно мінливих топологій мереж, має вирішальне значення для вдосконалення заходів кібербезпеки.

Потреба у використанні нейронних мереж обумовлена важливістю швидкого та ефективного виявлення та реагування на кіберзагрози, що можуть завдати значної шкоди важливим інфраструктурам. На відміну від звичайних алгоритмів, LSTM мережі здатні вчасно адаптуватися до нових видів кібератак, забезпечуючи таким чином більш надійний захист від потенційних збитків та зберігаючи стабільність критичних систем.

Отже, розробка та впровадження ефективних LSTM нейронних мереж для захисту великомасштабних мереж є вкрай важливою у сучасному динамічному кіберпросторі. Нейронні мережі не тільки покращують рівень безпеки, але й відкривають нові перспективи у виявленні та протидії кіберзагрозам, завдяки своїй здатності аналізувати великі об'єми даних та виявляти складні патерни поведінки в реальному часі.

Об’єкт: Використання рекурентних нейронних мереж для аналізу та розпізнавання кіберзагроз у адаптивних мережевих топологіях.

Предмет: Розробка та аналіз LSTM нейронної мережі, що спроможна класифікувати та прогнозувати кіберзагрозу на основі аналізу даних мережевої топології.

Мета: Створення LSTM нейронної мережі, яка здатна ефективно ідентифікувати різні сценарії в мережевій топології, включаючи нормальну діяльність, обслуговування та кібератаки.

Завдання: Дослідження сучасних підходів до використання нейронних мереж у сфері кібербезпеки: Аналіз останніх розробок та методів у галузі кібербезпеки, зокрема вивчення використання рекурентних нейронних мереж для ідентифікації та протидії кіберзагрозам у складних мережевих системах.

Формалізація математичної моделі адаптивної мережевої топології: Розробка та визначення математичних параметрів, які відображають характеристики та поведінку адаптивних мережевих топологій, щоб забезпечити адекватну основу для тренування нейронної мережі.

Генерація та аналіз датасету: Визначення структури та параметрів датасету, який імітує різні сценарії поведінки мережевої топології, включаючи нормальну роботу, обслуговування та кібератаки. Цей крок включає збір або генерацію реалістичних даних для тренування моделі.

Розробка алгоритмів для LSTM мережі: Визначення та реалізація алгоритмічних підходів для ефективної роботи LSTM мережі з обраним датасетом, щоб забезпечити точне виявлення і класифікацію кіберзагроз.

Тренування та валідація моделі: Виконання процесів навчання та перевірки моделі на основі підготовлених даних. Це включає в себе аналіз результатів для оцінки точності, надійності та ефективності моделі в різних сценаріях.

Аналіз ефективності моделі в різних сценаріях: Оцінка здатності моделі розпізнавати та класифікувати різноманітні типи кіберзагроз та зміни в мережевих топологіях, щоб визначити її практичну застосовність у реальних умовах кібербезпеки.

Методи дослідження:

- Теоретичний аналіз
- Комп'ютерне моделювання та програмування
- Аналіз даних
- Машинне навчання

Наукова новизна: Наукова новизна цієї магістерської роботи полягає в детальному розробленні та описі функціонування рекурентної нейронної мережі з довгою короткочасною пам'яттю (LSTM), призначеної для ідентифікації кіберзагроз у великомасштабних адаптивних мережевих топологіях.

На відміну від попередніх досліджень, які лише концептуально розглядали можливість застосування подібних мереж для кібербезпеки, ця робота вперше представляє практичну реалізацію та детальний аналіз такої мережі, включаючи її структуру, алгоритми обробки даних та механізми навчання.

Головною інновацією є створення комплексної моделі, що інтегрує LSTM у контексті адаптивних мережевих систем, забезпечуючи розширене розуміння та аналітику станів кожного вузла в мережі. Це включає в себе здатність мережі виявляти нові закономірності в поведінці мережі, що важливо у контексті постійно прогресуючих кібератак.

Особливістю розробленої моделі є її спроможність зберігати інформацію про стан мережі на значній кількості попередніх тактів, що дозволяє робити більш точні та обґрунтовані прогнози щодо потенційних кіберзагроз.

Ця робота не лише надає новий внесок у сферу кібербезпеки, але й відкриває шлях для подальших досліджень та розвитку в цій динамічній і важливій області.

Практичне значення отриманих результатів: Розроблена модель LSTM має широке практичне застосування, що відкриває нові горизонти у сфері кібербезпеки:

- Моніторинг та Кіберзахист Великомасштабних Мереж
- Оптимізація Ресурсів в IT-Інфраструктурі
- Підтримка Прийняття Рішень у Кібербезпеці
- Захист Критичної Інфраструктури

Незважаючи на показану ефективність, модель потребує подальшого вдосконалення. Важливим кроком є її тренування на реальних датасетах, що дозволить підвищити точність та адаптивність моделі до різних сценаріїв кіберзагроз. Таке тренування допоможе моделі краще узагальнювати та прогнозувати кіберзагрози, підвищуючи її практичну цінність у реальних умовах.

Структура кваліфікаційної роботи: Робота включає теоретичний огляд, розробку концепції, реалізацію прототипу та його тестування.

Розділ I: Було детально розглянуто теоретичні аспекти кіберзагроз, властивості адаптивних мережевих топологій, а також роль та потенціал рекурентних нейронних мереж у розпізнаванні таких загроз. Проведено порівняльний аналіз існуючих рішень, який підкреслив значимість інноваційних підходів у сфері кібербезпеки.

Розділ II: Розроблено концепцію адаптивної мережевої топології та пропоновану модель LSTM. Описано метод навчання нейронної мережі, що включає підготовку даних, їх передобробку, створення послідовностей, та тренування моделі.

Розділ III: Здійснено реалізацію та тестування прототипу нейронної мережі. Підготовка дата-сету з використанням алгоритму генерації.

РОЗДІЛ І. ТЕОРЕТИЧНІ АСПЕКТИ СПОСОБУ РОЗПІЗНАВАННЯ КІБЕРЗАГРОЗ У ВЕЛИКОМАСШТАБНИХ МЕРЕЖЕВИХ СИСТЕМАХ НА ОСНОВІ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ З ДОВГОЮ КОРОТКОТРИВАЛОЮ ПАМ'ЯТТЮ

1.1 Вступ до проблематики кіберзагроз великомасштабних систем.

У сучасну цифрову епоху великомасштабні мережеві системи є невід'ємною частиною функціональності критичних операцій у різноманітних секторах. Ці складні системи, що характеризуються широкою можливістю підключення та оперативною ефективністю, одночасно піддаються впливу безлічі кіберзагроз. Такі загрози визначаються як потенційні зловмисні атаки, спрямовані на незаконне проникнення, порушення, виведення з ладу або використання мережевих інфраструктур.

Наслідки кіберзагроз для великомасштабних систем є значними, виходять за рамки простих технічних збоїв і впливають на економічні, соціальні та політичні аспекти залежних організацій. Випадок успішної кібератаки може призвести до значних фінансових збитків, скомпрометувати конфіденційні дані, перервати основні послуги та заплямувати репутацію залучених організацій. Через взаємозв'язок, притаманний цим системам, наслідки таких загроз часто спричиняють ефект доміно, кульмінацією якого є масові, каскадні збої системи.

Кіберзагрози характеризуються своєю динамічною та швидкою еволюцією, яка часто випереджає темпи прогресу оборонних технологій. Цьому прогресу сприяють дедалі складніші мережеві системи, зростаюча кількість взаємопов'язаних пристроїв і постійний прогрес у методологіях та наборах інструментів агресивного злому. Глибоке розуміння цієї еволюційної траєкторії є обов'язковим для формулювання ефективних стратегій безпеки.

Звичайні методології виявлення кіберзагроз часто недостатні для вирішення складності та еволюційного характеру цих загроз. Це викликало зростаючий інтерес до розробки більш просунутих, адаптивних стратегій, які можуть динамічно відповідати мінливим парадигмам ландшафту кіберзагроз.

Примітно, що впровадження штучного інтелекту, зокрема довготривалої короткочасної пам’яті (LSTM), рекурентних нейронних мереж, стало життєздатним та інноваційним підходом. У наступних розділах ми заглибимося в механізм цих нейронних мереж і з’ясуємо їх потенційну роль у зміцненні інфраструктури кібербезпеки великомасштабних мережевих систем.

1.1.1 Захист мережевих топологій від кіберзагроз

Важливо розуміти топологію мережі, щоб ефективно захистити її від кіберзагроз. Загальні топології включають шину, зірку, кільце, сітку та гібридні конструкції. Кожна топологія має свої унікальні вразливості та вимагає спеціальних захисних заходів.

Впровадження заходів безпеки:

1. Регулярна оцінка вразливості: це передбачає виявлення, класифікацію та визначення пріоритетів уразливостей у мережевій інфраструктурі, комп’ютерних системах і програмах. У цьому процесі [4] часто використовуються автоматизовані інструменти, такі як сканери безпеки мережі. Регулярні оцінки мають вирішальне значення, оскільки постійно виявляються нові вразливості, і ними потрібно ефективно керувати.

2. Надійна фізична безпека: цей аспект безпеки мережі часто ігнорується, але життєво важливо захистити фізичний доступ до мережевих компонентів, таких як комп’ютери, сервери та маршрутизатори. Поширені загрози включають людську помилку, навмисний саботаж і стихійні лиха. Стратегії підвищення фізичної безпеки включають контроль доступу до даних, забезпечення безпечних місць зберігання даних, оцінку цінності даних, а також увімкнення сповіщень у реальному часі та постійного моніторингу.

3. Брандмауери та системи виявлення вторгнень: брандмауери використовуються для контролю потоку мережевого трафіку та запобігання несанкціонованому доступу. Системи виявлення вторгнень (IDS) контролюють і виявляють несанкціонований доступ до системи або маніпуляції. Обидва

інструменти працюють за принципом багаторівневого захисту та мають вирішальне значення для безпеки мережі.

4. Сегментація мережі [5] та контроль доступу: сегментація мережі на різні домени обмежує зв’язок між цими доменами та контролює трафік у певних місцях за допомогою інструментів безпеки, таких як глибока перевірка пакетів, виявлення вторгнень і брандмауери. Мікросегментація ще більше вдосконалює це, застосовуючи політики на деталізованому рівні, наприклад, за програмою чи пристроєм.

5. Шифрування. Шифрування даних [6] під час передачі, наприклад за допомогою SSL/TLS, має важливе значення для захисту від підслуховування та атак типу "людина посередині". Організаціям важливо захищати не лише свій інтернет-трафік, але й внутрішні мережі, корпоративні магістральні мережі та VPN. Однак це має бути частиною комплексної стратегії, яка також враховує ризики для даних у місцях їх походження та призначення.

6. Передові заходи: ШІ та машинне навчання

Використання штучного інтелекту та алгоритмів машинного навчання може значно підвищити безпеку мережі. Ці технології можуть передбачати та ідентифікувати незвичні шаблони, які вказують на кіберзагрозу, забезпечуючи додатковий рівень безпеки.

Захист мережевих топологій від кіберзагроз передбачає повне розуміння структури мережі, регулярні оцінки, заходи фізичної та цифрової безпеки та використання передових технологій, як-от AI. Це безперервний процес, який розвивається разом із мережею та ландшафтом кіберзагроз.

1.2 Адаптивні мережеві топології великомасштабних систем

Мережна архітектура, по суті, є планом, який визначає макет, протоколи зв’язку та схеми з’єднання мережевих систем. Він розроблений для задоволення конкретних вимог підключення, з різними архітектурами, які пропонують унікальні переваги для конкретних потреб [7]. Наприклад, мережі центрів обробки даних оптимізовані для доступу до даних і розміщення додатків, тоді як

глобальні мережі (WAN) дозволяють з’єднуватися на більшій відстані. Кожен тип мережевої архітектури має власний набір міркувань безпеки та вимог до підключення, що підкреслює необхідність різноманітних архітектур для задоволення різних операційних вимог.

Сучасні мережі значно еволюціонували від традиційних топологій до більш складних і ефективних проектів. Ця еволюція обумовлена потребою в надійності, резервуванні та гнучкості. Наприклад, сітчасті мережі забезпечують кілька шляхів для переміщення даних, підвищуючи надійність мережі. Подібним чином, програмно-визначена мережа (SDN) пропонує більший контроль і простіше керування завдяки розділенню рівня керування та даних. Різноманітність топологій, таких як P2P, деревовидна, віртуальна, хмарна та багатохмарна топології, задовольняє конкретні вимоги, такі як децентралізація, масштабованість та інтеграція з хмарними службами.

Вибір конкретної топології мережі часто залежить від конкретних потреб і обмежень мережі, таких як вартість, продуктивність, безпека та масштабованість. Наприклад, mesh-мережі вибираються через їх надлишковість, тоді як топології гібридної хмари вибираються через їх баланс масштабованості та безпеки. Різноманітність доступних мережевих топологій дозволяє створювати індивідуальні рішення, які найкраще відповідають робочим вимогам і обмеженням різних мережевих середовищ.

Таким чином, безліч мережевих архітектур і топологій є відповіддю на різноманітні вимоги та проблеми, з якими стикаються різні мережеві середовища. Ця різноманітність дозволяє створювати індивідуальні рішення, які оптимізують продуктивність, безпеку та економічну ефективність для різних операційних контекстів.

1.2.1 Огляд існуючих архітектур і топологій

Огляд існуючих мережевих архітектур і топологій [8] включає:

Топологія шини: один центральний кабель, до якого безпосередньо підключені всі вузли мережі. В основному використовується в невеликих мережах.

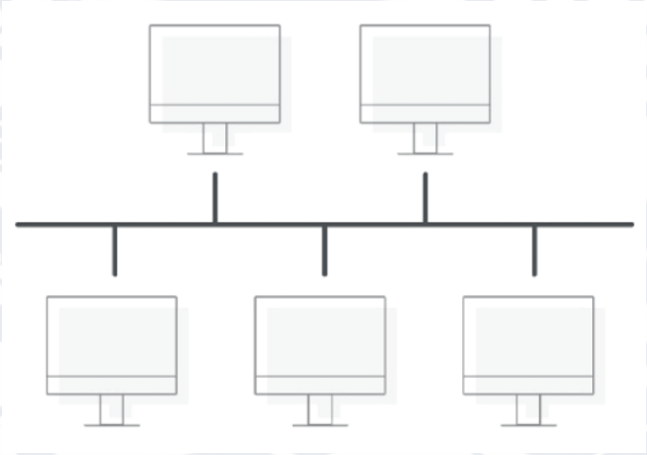

Рис. 1.1. Топологія шини [8].

Зіркова топологія: має центральний концентратор або комутатор, до якого підключаються всі мережеві пристрої. Це поширене явище в домашніх мережах.

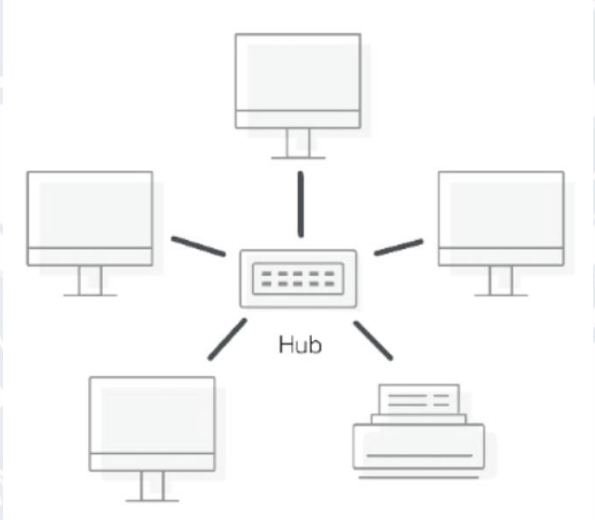

Рис. 1.2. Зіркова топологія [8].

Кільцева топологія: кожен мережевий пристрій підключено до двох інших пристроїв, утворюючи кільце. Використовується в деяких локальних мережах.

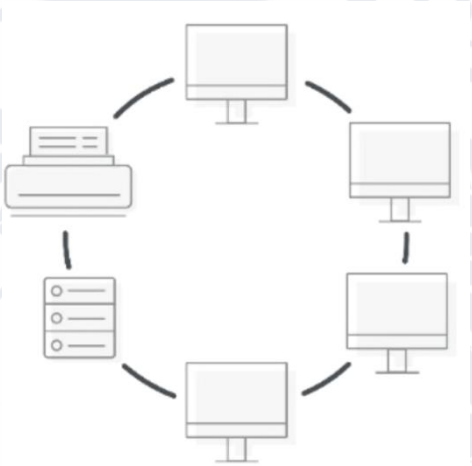

Рис. 1.3. Кільцева топологія [8].

Топологія сітки: кожен вузол підключений до будь-якого іншого вузла в мережі, що забезпечує високий рівень резервування.

Рис. 1.4. Топологія сітки [8].

Деревоподібна топологія: поєднує характеристики зіркоподібної та шинної топології. Ієрархічний за своєю природою, він використовується у великих мережах.

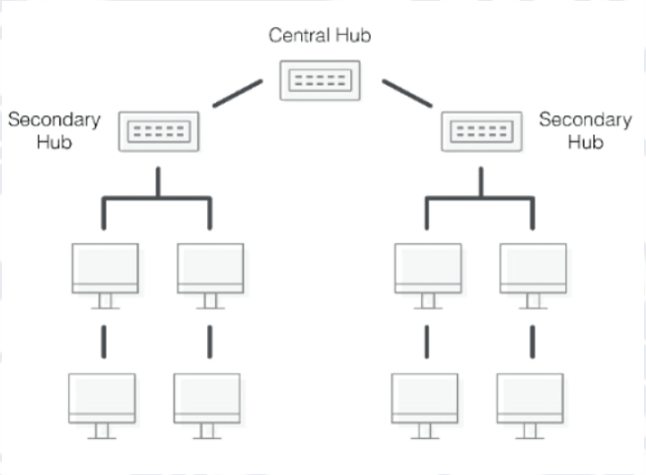

Рис. 1.5. Деревоподібна топологія [8].

Гібридна топологія: поєднання двох або більше різних типів топологій.

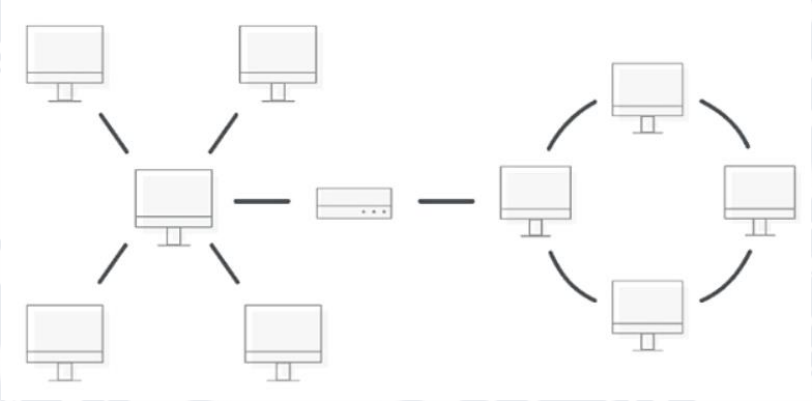

Рис. 1.6. Гібридна топологія [8].

Однорангова топологія (P2P): кожен вузол однаково розділяє роль клієнта та сервера. Поширений у файлообмінних мережах.

Програмно-визначена мережа (SDN): відокремлює площину керування від площини даних для більш гнучкого керування мережею.

Топологія віртуальної (накладної) мережі: створюється на основі існуючих фізичних мереж, часто використовується в VPN і SD-WAN.

Хмарні та мультихмарні топології: у хмарних топологіях ресурси розміщуються в хмарі, тоді як мультихмарні стратегії використовують послуги від кількох хмарних постачальників.

Кожна з цих топологій має свої унікальні переваги, обмеження та випадки використання, а вибір залежить від конкретних вимог до мережі, таких як розмір, масштабованість, гнучкість і вартість.

1.3 Рекурентні нейронні мережі та їх застосування для розпізнавання кіберзагроз

Повторювані нейронні мережі [9] (RNN) — це клас штучних нейронних мереж, у яких зв’язки між вузлами утворюють орієнтований граф у часовій послідовності. Ця архітектура дозволяє їм демонструвати часову динамічну поведінку та обробляти послідовності даних, що робить їх придатними для таких програм, як розпізнавання мовлення, моделювання мови та прогнозування часових рядів.

У контексті кібербезпеки RNN особливо ефективні завдяки своїй здатності обробляти послідовні та часові ряди даних, що є загальною характеристикою мережевого трафіку. Кіберзагрози часто проявляються як аномалії або закономірності в мережевих даних, які розкриваються з часом. RNN можуть навчитися розпізнавати ці закономірності та передбачати потенційні загрози, аналізуючи історичні дані [10].

Виявлення аномалій у мережевому трафіку: мережі RNN вміють аналізувати дані часових рядів, такі як мережеві пакети або журнали, виявляючи відхилення від типових шаблонів. Ця здатність має вирішальне значення для виявлення аномалій у режимі реального часу для позначення потенційних загроз, таких як DDoS-атаки або спроби викрадання даних, адаптації до мінливих шаблонів трафіку та нових загроз.

Аналіз поведінки користувачів і виявлення внутрішніх загроз: RNN у співпраці з нейронними мережами Graph (GNN) можуть моделювати й аналізувати моделі поведінки користувачів в організації. Вони виявляють

відхилення від нормальної поведінки, допомагаючи у виявленні внутрішніх загроз. Аналізуючи послідовності дій користувача (наприклад, доступ до файлів, спроби входу), ці мережі можуть позначати аномальну поведінку, яка може вказувати на потенційні ризики безпеці.

Виявлення фішингових електронних листів: RNN у поєднанні зі згортковими нейронними мережами (CNN) можуть ефективно виявляти фішингові електронні листи. RNN аналізують текстовий вміст на наявність шаблонів, що вказують на фішинг, тоді як CNN оцінюють візуальні елементи, як-от вбудовані зображення чи логотипи. Цей комбінований підхід підвищує точність виявлення та захисту фішингової електронної пошти.

Автоматичне виявлення вразливостей і виправлення: RNN у поєднанні з CNN можуть аналізувати вихідний код і файли конфігурації для виявлення вразливостей безпеки. RNN обробляють послідовність інструкцій у коді, виявляючи шаблони, пов’язані з відомими вразливими місцями, тоді як CNN аналізують конфігураційні файли на наявність неправильних конфігурацій або небезпечних налаштувань.

Загальне застосування в кібербезпеці: унікальна архітектура RNN, яка включає підтримку прихованого стану пам’яті, дозволяє їм запам’ятовувати та використовувати минулі вхідні дані для прогнозування майбутніх подій. Ця функція є безцінною для виявлення моделей і тенденцій у потоках даних, сприяючи ранньому виявленню кібератак і проактивному пом’якшенню загроз.

Ці програми підкреслюють універсальність і ефективність мереж RNN у різних аспектах кібербезпеки, від виявлення загроз у реальному часі до прогнозної аналітики та оцінки вразливості системи.

1.4 Порівняльний аналіз існуючих рішень захисту мережевих топологій за допомогою нейронних мереж

Загальний порівняльний аналіз рішень на основі нейронних мереж для захисту мережевих топологій [11 - 13]:

Повторювані нейронні мережі (RNN)

Випадок використання: RNN особливо підходять для даних часових рядів або послідовної обробки даних, що є типовим для мережевого трафіку та системних журналів.

Переваги:

Розпізнавання часових шаблонів: RNN можуть ідентифікувати шаблони в даних, де послідовність і час подій є вирішальними.

Виявлення аномалій: ефективне виявлення незвичайних послідовностей або відхилень з часом, корисне для виявлення вторгнень.

Обмеження:

Проблема із зникаючим градієнтом: труднощі з вивченням довготривалих залежностей у послідовностях даних.

Обчислювальна інтенсивність: вимагає значної обчислювальної потужності, що може бути проблемою в програмах реального часу.

Застосування:

Системи виявлення вторгнень (IDS): для виявлення та прогнозування спроб вторгнення на основі минулих тенденцій даних.

Аналіз поведінки мережі: для аналізу та прогнозування аномалій мережі на основі історичних моделей трафіку.

Згорточні нейронні мережі (CNN)

Випадок використання: CNN, які зазвичай використовуються для обробки зображень, можуть бути адаптовані для мережевих даних, представлених візуально або просторово.

Переваги:

Розпізнавання шаблонів у просторових даних: ефективне визначення шаблонів у даних, структурованих у форматі сітки (як зображення).

Зменшення розмірності: здатність зменшувати багатовимірні дані до більш керованих форм без втрати важливої інформації.

Обмеження:

Вимоги до попередньої обробки: мережеві дані можуть потребувати трансформації для ефективної обробки CNN.

Менш ефективний для послідовних даних: не призначений для аналізу часових рядів або послідовних даних.

Застосування:

Візуальний аналіз мережевого трафіку: для аналізу мережевого трафіку, представленого у візуальному або просторовому форматі.

Виділення функцій у даних безпеки: вилучення ключових функцій зі складних наборів даних безпеки для подальшого аналізу.

Глибинні нейронні мережі (DNN)

Випадок використання: підходить для широкого спектру типів даних, включаючи складні та багатовимірні набори даних.

Переваги:

Вивчення складних шаблонів: здатність вивчати складні шаблони та зв’язки у великих наборах даних.

Загальна застосовність: універсальність для різних типів даних і програм.

Обмеження:

Великі потреби в даних: потрібні значні обсяги навчальних даних.

Ризик переналагодження: може переналаштувати тренувальні дані та погано працювати на невидимих даних.

Застосування:

Виявлення зловмисного програмного забезпечення: ідентифікація та класифікація зловмисного програмного забезпечення з величезних наборів даних.

Класифікація мережевого трафіку: розрізнення звичайного та потенційно шкідливого трафіку.

Гібридні моделі нейронних мереж

Випадок використання: поєднує різні типи нейронних мереж, щоб отримати вигоду з їхніх сильних сторін.

Переваги:

Комплексний аналіз: може аналізувати дані з різних точок зору.

Підвищена надійність: усуває обмеження окремих моделей, щоб забезпечити більш ефективне рішення.

Обмеження:

Складність моделі: поєднання різних архітектур може призвести до складних моделей, які важко навчити та оптимізувати.

Ресурсовитратність: може вимагати більше обчислювальних ресурсів і часу для навчання та розгортання.

Застосування:

Комплексне виявлення кіберзагроз: коли необхідно проаналізувати кілька характеристик даних, наприклад, у розширених постійних загрозах (APT).

Інтегровані рішення безпеки: пропонують всебічний аналіз безпеки з урахуванням різних аспектів даних.

Кожна з цих архітектур нейронних мереж має свої унікальні переваги та обмеження, і вибір того, яку використовувати, часто залежить від конкретних вимог топології мережі, яку вони захищають. На практиці комбінація цих архітектур (гібридні моделі) часто використовується для забезпечення більш комплексного рішення безпеки.

РОЗДІЛ ІІ. РОЗРОБКА КОНЦЕПЦІЇ АДАПТИВНОЇ МЕРЕЖЕВОЇ ТОПОЛОГІЇ ТА РЕКУРРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ

2.1Опис адаптивної мережевої топології.

Адаптивна мережева топологія — це гнучка структура мережі, яка може змінюватися та адаптуватися до різних умов і потреб. Основною характеристикою адаптивної топології є її здатність динамічно реорганізувати себе у відповідь на зміни в мережі, такі як відключення компонентів, зміна шляхів передачі даних, коливання в обсязі трафіку, або кібератаки.

Компонентами мережевої топології є кінцеві точки або точки пересилання в мережі та зв’язки між цими кінцевими точками. Кінцевими точками можуть бути сервери, роутери, комутатори, якісь інші кінцеві користувачі (наприклад, комп'ютери або мобільні пристрої). А зв’язки між кінцевими точками — це провідні (інтернет-кабелі) або безпровідні (Wi-Fi, Bluetooth) способи зв’язку.

Адаптивну мережеву топологію представимо у вигляді орієнтованого графа [14], де кожен вузол символізує компонент мережі, а ребро – звязки між компонентами мережі. Кожен вузол має вагу — атрибут критичності. Наприклад, атаки відмови в обслуговуванні (DoS) [15] на систему можна розглядати як видалення певних вершин і подальшу перебудову графа.

Формалізовано задача виглядає наступним чином: мережева топологія представлена у вигляді

$G=(V,E)$

де $V = \{v_1, v_2, \dots, v_n\}$ – множина вузлів мережі, $E == \{e_1, e_2, \dots, e_n\}$ – кількість дуг, $E \subseteq (V \times V)$.

В цьому випадку, мережева топологія характеризується високим ступенем мінливості, тому початковий стан мережевий топології можна позначити як G^0, її стан в момент часу $p - G^p$ (для якого попереднім є $G^{p-1}, p \in (1,2, \dots,)$). Тоді, для опису адаптивної мережевої топології доцільно розглядати набір статистичних графів [16] $DG = (G^0, G^1, \dots)$, (рис. 2.1).

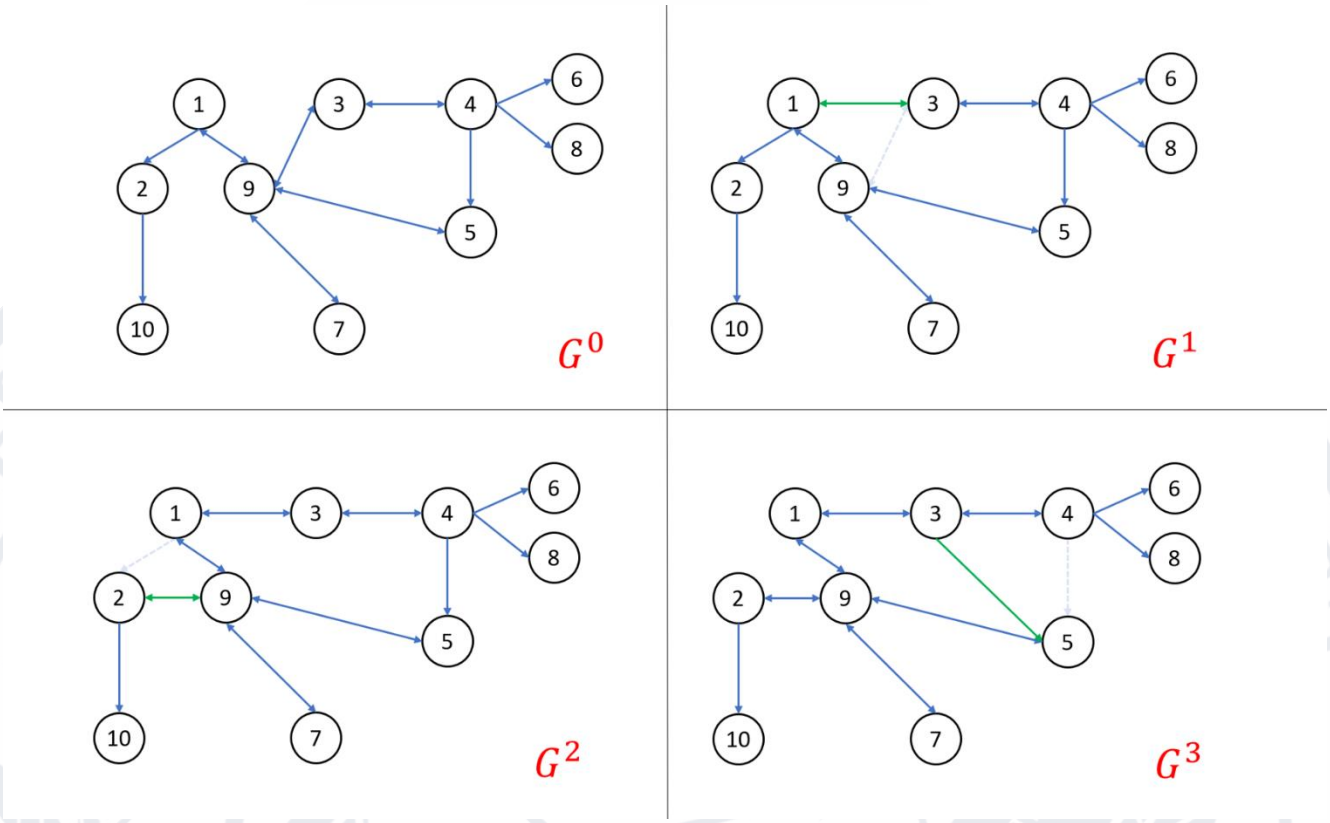

Рис. 2.1. Набір статичних графів (схема адаптивної мережевої топології).

Будь-який граф G^{i-1} асоціюється з набором наступних атрибутів: попередній стан G^{i-1}, набір операцій (властивих графам, а саме видалення / додавання дуг) Op^i при перетворенні G^{i-1} в G^i і часова мітка t. Стан мережевої топології частково залежить від критичності кожної вершини, що позначена як cr:

$$f: v^i \rightarrow \{cr_{v^i}\}$$

де f – оператор, що задає відповідне значення критичності, v – довільний вузол графа топології.

Критичність вузла у мережі визначається за його іншими характеристиками, такими як пропускна спроможність (англ. *bandwidth*), кількість транзитних вузлів (*num*) і затримка при передачі даних (*delay*).

Таким чином, рішення зводиться до визначення множини вершин $V': V' \in V$, таких, що:

– серед інших вершин, критичність цих вузлів є найвищою, $cr_V \rightarrow MAX, cr_{V'} = \{cr_{V'_1}, \dots, cr_{V'_m}\}$;

– зміни в характеристиках *bandwidth*, *num* і *delay* у вузлах з високою критичністю, що перевищують припустимі значення. Формально це виражається в тому, що застосування оператора Δ до множини критичних вузлів V' показує результат, який перевищує певний пороговий показник (*threshold*). При цьому значення порогу може бути перевищено як внаслідок надмірних змін одного з параметрів (пропускної здатності, кількості транзитних участків і затримки), так і внаслідок надмірних змін їх певної комбінації $\Delta(V') > treshold_{V'}$ [3, 4].

Проблеми на мережевих вузлах можуть характеризуватись. аномальними показниками активності мережевих вузлів або відмовою в обслуговуванні пов'язаних кластерів. Адаптивність можна розглядати як властивість мережі забезпечувати стійке функціонування при настанні негативних факторів, що впливають на функціональність вузлів. До таких негативних факторів можна віднести втрату зв'язку, зниження продуктивності, пропускної спроможності чи появу зайвих перешкод у каналі зв'язку.

Адаптивність – характеристика топології, незалежна від типу вузлів, що дозволяє узагальнити різноманітні можливі топології разом із механізмами забезпечення надійності. Зазвичай будь-яку мережеву топологію можна подати у вигляді матриці суміжності розміром $N \times N$, де N – кількість вузлів у мережі. При цьому для оцінки стійкості топології враховується лише наявність зв'язку між функціональними вузлами. Характеристики мережі, такі як середня пропускна здатність, затримки, наявність центрів керування, не враховуються. Крім того, передбачається, що вузли в мережі рівноправні, але мають атрибути пріоритету - значення критичності в межах функціонального навантаження.

Початкова матриця має вигляд відповідно до поточного стану мережевої топології:

$$A_0 = \begin{bmatrix} 1 & 0 & \cdots & 1 & 0 \\ & 1 & \ldots & & \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ & & \ldots & & \\ 1 & 0 & \ldots & 0 & 1 \end{bmatrix},$$

де A_0 – початкова матриця зв’язку.

Значення «1» позначає присутність зв’язку, «0» – відповідно навпаки. Отже, якщо елементи на головній діагоналі початкової матриці зв'язку мають значення "1", це означає, що вузли завжди мають підключення до самих себе, якщо вони не вийшли з ладу.

Для моделювання втілення кіберзагроз запропоновано використовувати перетворення матриці А, таке, що:

$$d: A_{t+1} \to A_{t+2} + d - 1,$$

де A_{t+1}, A_{t+2} – стани матриці в моменти часу $t + 1$ та $t + 2$.

Це означає, що після одного застосування матриця зв'язку може втратити лише один вузол, і адаптаційне перетворення – використання резервного маршруту – здійснюється тільки при порушенні роботи основного (найкоротшого) маршруту між цільовими вузлами. Якщо зміни в матриці зв'язку не впливають на основний маршрут, то адаптаційні перетворення не відбуваються.

2.2 Розробка концептуальної моделі рекурентної нейронної мережі

Адаптивна мережева топологія, яка описується у попередньому підрозділі, включає кінцеві точки або вузли мережі та їхні зв'язки. Важливим аспектом такої топології є її здатність до динамічної реорганізації у відповідь на різноманітні зміни, включаючи кібератаки. Це робить LSTM (Long Short-Term Memory) нейронну мережу ідеальною для аналізу та захисту такої структури.

2.2.1 Пропонований підхід для розпізнавання кіберзагроз на адаптивну мережеву топологію

Для ідентифікації кіберзагроз у адаптивних мережевих топологіях застосуємо таку методологію:

Призначення кожному вузлу в топології певного рівня критичності, обумовленого числом сусідніх вузлів. Високе число сусідніх вузлів вказує на вищу критичність. Існують три рівні критичності:

"high" - для вузлів з найбільшим показником,

"medium" - для вузлів із проміжним значенням,

"low" - для вузлів з низькою кількістю суміжних вузлів.

При зміні стану вузлів у маршруті визначається ступінь впливу змін на метрики топології (пропускну здатність, кількість транзитних ділянок та часову затримку).

У випадку, якщо цілісність топології порушена, тобто є ізольовані області та метрики для маршруту не задовольняють мінімально допустимим значенням, вважається, що зміни викликані внаслідок спроби реалізації кіберзагрози.

В інших випадках зміни вважаються стандартними. Застосування нейронної мережі з довгою короткочасною пам'яттю для класифікації змін у топології покращує точність, оскільки фіксуються попередні стани вузлів. При кіберзагрозах, адаптація топології може прийняти каскадний характер, на відміну від звичайної адаптації, де зміни відбуваються більш поступово.

Перед експериментом підготовлено дані про топологію мережі за 20 000 циклів. На кожному циклі можлива стандартна адаптація (якщо метрики вузлів опускаються нижче значень для нормального функціонування, але вищі ніж мінімально допустимі значення, крім зв'язків вузлів) або адаптація через кібератаку. Адаптація також може бути відсутня.

У тренувальному наборі чітко позначені стандартні та зміни при кібератаці. Під час формування набору даних встановлено, що критичні вузли менше схильні до стандартних змін, а більшість адаптацій після зміни критичних вузлів є аварійними.

Дані містять такі поля:

"timestamp": Це мітка часу, що вказує на конкретний момент часу, коли були зафіксовані дані.

"process_type": Це поле ідентифікує тип процесу або стану системи в момент часу, вказаний у "timestamp", може містити такі значення:

"normal" - система функціонує в нормальному режимі.

"maintenance" - система функціонує в режимі обслуговування (стандартні зміни).

"cyber_attack" – на систему здійснена кібератака.

"nodes": Це масив об'єктів, кожен з яких представляє вузол мережі. Кожен вузол має такі характеристики:

"node_ID": Унікальний ідентифікатор вузла.

"criticality_mark": Оцінка критичності вузла, може бути "high", "medium" або "low", вказує на рівень важливості вузла в мережі.

"metrics": Метрики вузла, які включають:

"bandwidth": Пропускна здатність вузла.

"connections_number": Кількість з'єднань, які має вузол.

"signal_delay": Затримка сигналу вузла.

"connections": Це об'єкт, що містить інформацію про зв'язки між вузлами. Ключі об'єкта відповідають ідентифікаторам вузлів, а значення - це об'єкти, які показують, чи існує з'єднання між вузлами (1 означає наявність з'єднання, 0 - його відсутність).

Реалізація включає три етапи:

1. Створення моделі адаптивної масштабної мережевої топології.

2. Навчання нейронної мережі на даних топології мережі.

3. Симуляція кіберзагроз, що спричиняють збої вузлів та стандартних адаптацій мережі. Навчання та тестування виконується на моделі з шістьма вузлами. Розглядаються окремі та каскадні збої. Окремий збій виключає один вузол, каскадний - велику частину топології. Обидва сценарії впливають на зміну топології.

2.2.2 Структура рекурентної нейронної мережі з довгою короткотривалою пам’яттю

Мережева архітектура LSTM складається з трьох частин, як показано на малюнку нижче (рис. 2.2), і кожна частина виконує окрему функцію [17].

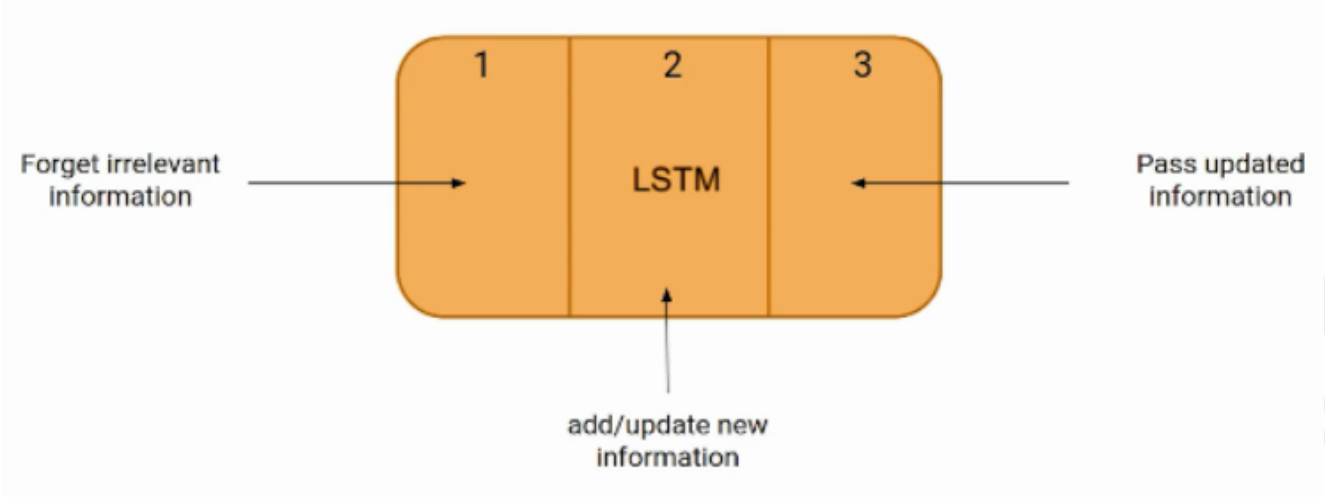

Рис. 2.2. Частини мережевої архітектури LSTM [17].

На початковому етапі визначається необхідність утримання інформації з минулого моменту часу або її відкидання як непотрібної. Далі клітина старається освоїти та інтегрувати свіжу інформацію, що приходить. Останній етап полягає в передачі оновленої інформації до наступного моменту часу, завершуючи таким чином один цикл роботи LSTM, що являється окремим часовим інтервалом.

Три ключові компоненти блоку LSTM, відомі як ворота, регулюють рух інформації внутрішньо та зовні клітини пам'яті або LSTM клітини. Ворота забуття роблять вибір про необхідність збереження даних, вхідні ворота визначають, які нові дані приймати, а вихідні ворота керують видачею результатів з клітини. Така структура LSTM, яка складається з трьох воріт та пам'яті клітини, може бути уявлена як нейронний шар у звичайній нейронній мережі, де кожен нейрон обробляє як приховані дані, так і актуальну інформацію.

Так само, як у базової моделі RNN, LSTM містить прихований стан: H(t-1) відображає стан попереднього моменту часу, тоді як Ht представляє поточний стан. Крім того, LSTM характеризується наявністю стану клітини, зазначеного як C(t-1) для попереднього часового відрізка і C(t) для поточного.

У цьому контексті прихований стан слугує роллю короткотермінової пам'яті, тоді як стан клітини виступає як довготермінова пам'ять. За допомогою наступного зображення (рис. 2.3) це можна краще зрозуміти.

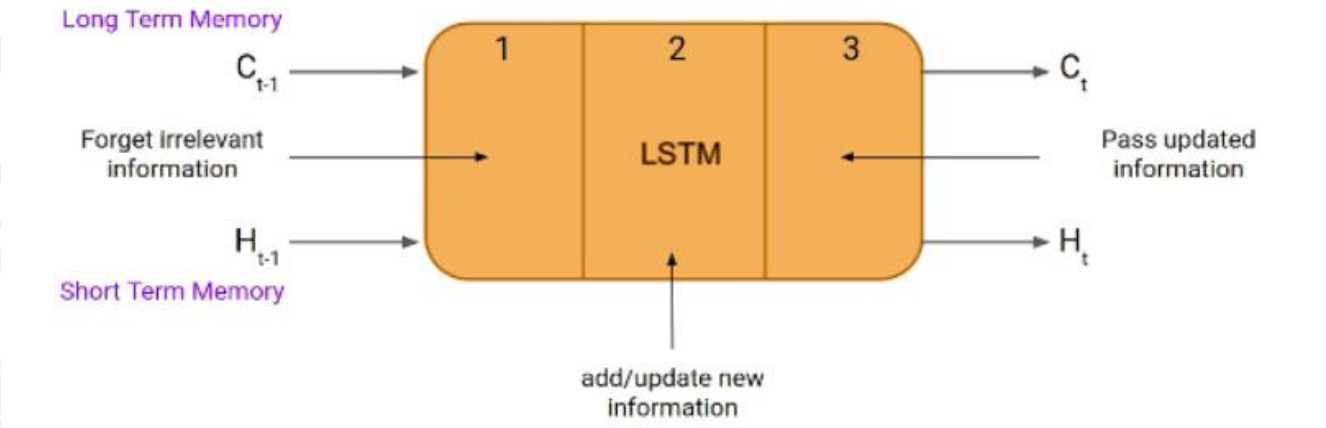

Рис. 2.3. Схема станів LSTM [17].

Розглянемо наступний приклад, щоб зрозуміти, як функціонує LSTM. Уявімо, що перед нами два речення, розділені крапкою. Перше речення звучить так: "Аліса - чудова вчителька", а друге - "Том, навпаки, не найкращий учень". Ясно, що у першому реченні йдеться про Алісу, але як тільки ми доходимо до крапки (.), ми починаємо говорити про Тома.

Під час переходу від першого речення до другого наша мережа має зрозуміти, що ми більше не згадуємо про Алісу. Тепер головним об'єктом нашої уваги є Том.

Першочергове завдання в елементі LSTM нейронної мережі полягає у визначенні, чи потрібно зберігати інформацію з минулого часового кроку або ж відмовитися від неї. Наступне рівняння демонструє механізм воріт забування.

$$f_t = \sigma(x_t \cdot U_f + H_{t-1} \cdot W_f)$$

Xt: введення поточної мітки часу.

Uf: вага, пов'язана з входом

Ht-1: прихований стан попередньої мітки.

Wf: Це вагова матриця, пов'язана із прихованим станом.

Згодом до нього застосовують функцію сигмоїди. Відповідно, ft перетворюється на число в діапазоні між 0 та 1. Цей коефіцієнт ft подалі використовується для множення на стан клітини з попереднього кроку часу, як це демонструється далі.

Якщо $f_t=0$ (забуваємо все),

$$C_{t-1}\times f_t=0$$

Якщо $f_t=1$ (не забуваємо нічого),

$$C_{t-1}\times f_t=C_{t-1}$$

Звернемося до альтернативного прикладу.

"Ліза відома своїми художніми талантами. Вона поділилася зі мною, що навчалася в академії мистецтв протягом трьох захоплюючих років".

Таким чином, в обох цих висловлюваннях ми обговорюємо Лізу. Проте кожне з них надає унікальну інформацію про неї. З першого ми дізнаємось про її здібності до малювання. У той час, друге повідомляє, що вона отримала освіту в академії мистецтв.

Тепер, роздумуючи над контекстом, наданим у першому висловлюванні, який елемент інформації у другому висловлюванні є критичним? По суті, не настільки важливо, яким чином вона поділилася цією інформацією — чи це було усно, чи вона написала листа. Важливим є факт її навчання в академії мистецтв, і це те, що ми хочемо, щоб наша модель зберегла для майбутніх висновків. Це саме завдання, яке виконують Вхідні ворота.

Вхідні ворота оцінюють значимість нової інформації, яка надходить, щоб вирішити, скільки уваги вона має отримати у подальших обчисленнях. Ось рівняння, що описує дію вхідних воріт.

$i_t = \sigma(x_t \cdot U_i + H_{t-1} \cdot W_i)$

Xt: введення з поточною міткою t.

Ui: вагова матриця введення

Ht-1: прихований стан на попередній мітці.

Wi: Вагова матриця введення, пов'язана із прихованим станом.

Ми знову застосували до нього сигмоподібну функцію. В результаті значення I в момент часу t буде між 0 та 1.

Нові дані, що мають бути передані в стан комірки, залежать від прихованого стану на попередньому часовому кроці t-1 та вхідних даних x на поточному часовому кроці t. Функцією активації у цьому випадку слугує tanh.

$N_t = \tanh(x_t \cdot U_c + H_{t-1} \cdot W_c)$ (нова інформація)

Завдяки функції tanh значення нової інформації перебуватиме в діапазоні від -1 до 1. Якщо значення Nt негативне, інформація віднімається від стану комірки, а якщо значення позитивне, інформація додається до комірки. стан на поточній часовій мітці.

Однак Nt не буде додано безпосередньо у стан комірки. Ось оновлене рівняння:

$C_t = f_t * C_{t-1} + i_t * N_t$

Тут Ct-1 - це стан осередку в поточну тимчасову мітку, а решта - це значення, які ми вирахували раніше.

Розглянемо інший контекст.

"Ліза, працюючи в лабораторії, розробила новий лікарський засіб, який має потенціал врятувати мільйони життів. За це відкриття, винахідлива _____".

Під час виконання цього завдання ми повинні доповнити друге речення. Отже, коли ми бачимо слово "винахідлива", ми розуміємо, що йдеться про особу. В контексті цього речення винахідливою є лише Ліза, ми не можемо стверджувати, що лабораторія чи новий засіб винахідливі. Таким чином, виходячи з наших очікувань, ми повинні підібрати відповідне слово для

заповнення пропуску. Це слово стане нашим результатом, і це завдання для нашого виходного вентиля.

Ось рівняння виходного вентиля, яке функціонує за аналогією з двома попередніми вентилями.

$$o_t = \sigma(x_t \cdot U_o + H_{t-1} \cdot W_o)$$

Його величина також знаходитиметься в межах від 0 до 1 через застосування функції сигмоїди. Далі, для визначення поточного прихованого стану, ми застосовуємо Ot та tanh оновленого стану клітинки, як це демонструється далі.

$$H_t = o_t \times \tanh(C_t)$$

Виявляється, прихований стан є функцією довготривалої пам'яті (Ct) та поточного виходу. Якщо вам потрібно отримати висновок поточної мітки часу, просто застосуйте активацію SoftMax до прихованого стану Ht.

$$\text{Output} = \text{Softmax}(H_t)$$

Тут токен із максимальним балом на виході є прогнозом.

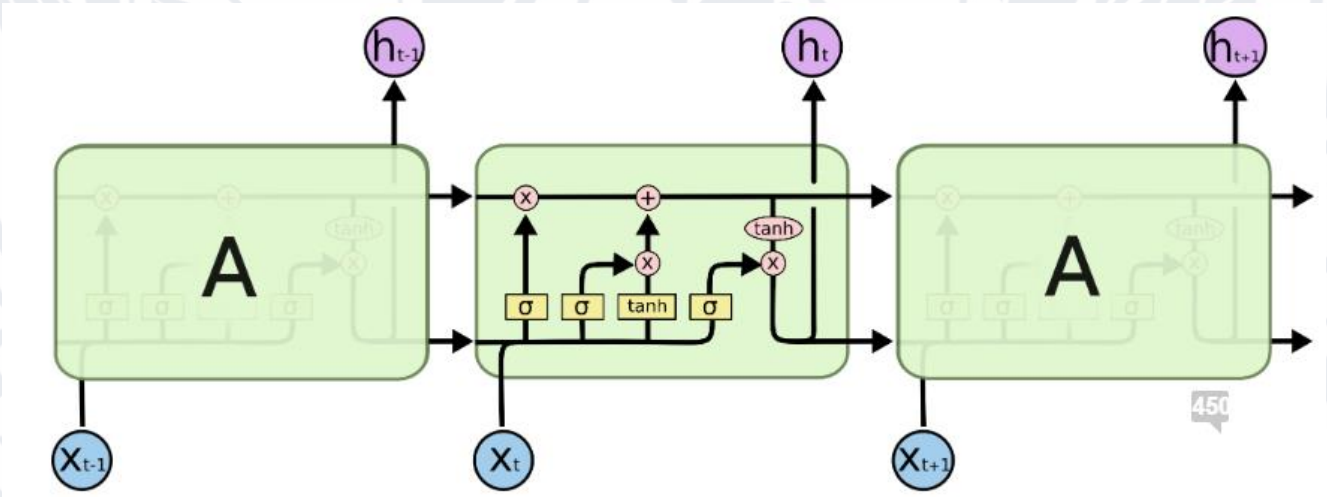

Рис. 2.4. Діаграма мережі LSTM [18].

2.2.3 Опис методу навчання нейронної мережі

У цьому підрозділі ми детально розглянемо ключовий аспект розробки нейронних мереж — метод навчання. Для навчання нейронної мережі був

вибраний метод надзворотнього поширення помилки [19] у поєднанні з оптимізатором Adam [20], який є фундаментальним для ефективності і точності LSTM нейронних мереж. Цей підхід дозволяє не тільки адекватно налаштувати вагові коефіцієнти мережі, але й забезпечує можливість уникнути типових проблем, таких як перенавчання або недостатнє навчання.

Метод надзворотнього поширення помилки у поєднанні з оптимізатором Adam для навчання LSTM нейронної мережі представляє собою ітеративний процес, який має на меті оптимізацію вагових коефіцієнтів у мережі для мінімізації помилки прогнозування. Процес можна розділити на наступні етапи:

Ініціалізація ваг: Ваги нейронів у мережі ініціалізуються випадковими невеликими числами.

Пряме поширення (Forward Propagation): На цьому етапі вхідні дані подаються у мережу та просуваються через приховані шари до вихідного. На кожному нейроні відбувається сумування вхідних сигналів з врахуванням ваг, та активація за допомогою відповідної функції активації.

Обчислення помилки: Помилка визначається як різниця між цільовими значеннями та тими, що були отримані на виході мережі. Ця різниця зазвичай обчислюється за допомогою функції втрат, наприклад, середньоквадратичної помилки.

Надзворотнє поширення (Backward Propagation): На цьому етапі відбувається обчислення градієнтів функції втрат по відношенню до кожного вагового коефіцієнта мережі. Градієнти обчислюються з використанням правила ланцюгового диференціювання, яке дозволяє визначити вплив кожної ваги на кінцеву помилку.

Оновлення ваг за допомогою оптимізатора Adam: Adam (Adaptive Moment Estimation) використовує оцінки перших (моменти) та других (невирівноважені варіанти) моментів градієнтів для адаптації швидкості навчання для кожного параметру. Adam обчислює адаптивні швидкості навчання на основі оцінок моментів низького порядку та квадратичних середніх значень градієнтів, що

дозволяє уникнути занадто великих або малих оновлень ваг і сприяє більш стабільній та швидкій конвергенції в процесі навчання.

Ітерації: Вищеописані кроки повторюються багаторазово в ітеративному процесі навчання. Кожна ітерація включає в себе пряме поширення, обчислення помилки, надзворотнє поширення та оновлення ваг.

Метод надзворотнього поширення помилки був обраний для виявлення кіберзагроз в адаптивній мережевій топології з кількох причин:

Секвенційна природа даних: Оскільки дані мають послідовну природу (кожен файл представляє стан мережі у певний момент часу), LSTM нейронна мережа є ідеальною для обробки таких типів даних. Метод надзворотнього поширення дозволяє LSTM мережі ефективно навчатися на основі цих послідовностей, зберігаючи інформацію про попередні стани і використовуючи її для виправлення помилок прогнозування.

Можливість мережі враховувати, як різні точки даних взаємопов'язані між собою з часом: Надзворотнє поширення дозволяє мережі враховувати залежності між різними точками даних у часі, що є критично важливим для виявлення кіберзагроз, які можуть розвиватися або змінюватися з плином часу.

Здатність до самоадаптації: Алгоритм надзворотнього поширення помилки у поєднанні з оптимізатором Adam дозволяє мережі адаптуватися до змін у мережевій топології, що є важливим для динамічних мережевих середовищ, де кіберзагрози постійно еволюціонують.

Оптимізація швидкості навчання: Оптимізатор Adam автоматично регулює швидкість навчання для кожного параметру в мережі, що допомагає прискорити навчання і покращити його якість. Це особливо корисно при роботі з великими датасетами, як у випадку з датасетом, що складається з 20 000 файлів.

Здатність до узагальнення: Метод надзворотнього поширення допомагає LSTM нейронній мережі уникнути перенавчання, що дає змогу краще узагальнювати знання і ефективніше виявляти невідомі кіберзагрози.

Ефективність з точки зору ресурсів: Незважаючи на те, що існують інші алгоритми навчання, такі як градієнтний спуск з моментом або RMSprop, оптимізатор Adam зазвичай вважається більш ефективним у багатьох задачах, оскільки він поєднує переваги обох цих методів, а також автоматично налаштовує швидкість навчання, що робить його ідеальним вибором для великих датасетів і складних задач, таких як виявлення кіберзагроз.

Вибір методу надзворотнього поширення помилки та оптимізатора Adam для LSTM був зроблений на основі цих аргументів, які роблять їх ідеально підходящими для даної задачі з виявлення кіберзагроз.

РОЗДІЛ ІІІ. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОТОТИПУ РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ

3.1 Підготовка дата сету

Розробка ефективної системи для ідентифікації кіберзагроз у адаптивних мережевих топологіях вимагає глибокого розуміння динаміки мережі та здатності точно класифікувати її стани. Основою успішного вирішення цієї задачі є наявність високоякісних даних, які точно відображають різні сценарії та можливі стани системи. З огляду на унікальність та специфічність цього дослідження, важливо мати датасет, який максимально відповідає реальним умовам та вимогам. На жаль, знайти готовий датасет, який би відповідав цим критеріям, виявилося неможливим. Тому було прийнято рішення створити власний датасет за допомогою спеціально розробленого скрипта на мові програмування Python [21].

Таке рішення має низку переваг:

Індивідуальне налаштування: Можливість точно налаштувати параметри генерації даних, що відповідають специфіці досліджуваної мережевої топології.

Гнучкість: Здатність швидко адаптувати та модифікувати скрипт для експериментування з різними сценаріями та умовами.

Відтворення реальних умов: Можливість створення сценаріїв, які імітують реальні умови функціонування мережі, включаючи різні види кіберзагроз.

Контроль якості даних: Повний контроль над якістю та відповідністю даних, які будуть використовуватися для навчання та тестування моделі.

Масштабованість: Здатність генерувати датасети різного розміру та складності, що дозволяє ефективно тестувати та налагоджувати нейронну мережу.

Враховуючи ці переваги, створення власного датасету є найбільш оптимальним рішенням для забезпечення високої точності та надійності результатів дослідження.

3.1.1 Структура дата сету

Навчальний датасет, створений для дослідження, має унікальну структуру та обсяг. Він складається з 20,000 файлів у форматі JSON, кожен з яких детально описує стан однієї мережевої топології в конкретний проміжок часу. Кожен файл являє собою окремий знімок стану мережі, що дозволяє досліджувати динаміку її поведінки та відповіді на різні зовнішні впливи, включаючи потенційні кібератаки (Додаток А).

Кожен JSON файл [22] містить наступні ключові елементи:

«timestamp» (String): Це мітка часу, яка вказує на конкретний момент часу, коли були зафіксовані дані. Формат мітки часу - "дд-мм-рррр гг:хх:сс".

«process_type» (String): Це поле ідентифікує тип процесу або стану системи в момент часу, вказаний у "timestamp". Можливі значення: "normal", "maintenance", "cyber_attack".

«nodes» (Array of Objects): Масив об'єктів, кожен з яких представляє вузол мережі зі своїми характеристиками.

«connections» (Object): Об'єкт, що містить інформацію про зв'язки між вузлами.

Вузли (Nodes)

Кожен вузол у масиві "nodes" має наступну структуру:

«**node_ID**» (String): Унікальний ідентифікатор вузла.

«**criticality_mark**» (String): Оцінка критичності вузла. Можливі значення: "high", "medium", "low".

«**metrics**» (Object): Об'єкт з метриками вузла, що включає:

«**bandwidth**» (Float): Пропускна здатність вузла в мегабайтах.

«**connections_number**» (Integer): Кількість з'єднань, які має вузол.

«**signal_delay**» (Float): Затримка сигналу вузла в мілісекундах.

Зв'язки (Connections)

Об'єкт "connections" має ключі, що відповідають ідентифікаторам вузлів ("node_1", "node_2", ...), і кожен такий ключ має внутрішній об'єкт, де:

Ключі внутрішнього об'єкта також відповідають ідентифікаторам вузлів.

Значення (Integer): Число, яке показує наявність (1) або відсутність (0) з'єднання між вузлами.

Типи Даних

String: Рядок тексту, використовується для представлення текстової інформації (наприклад, "timestamp", "node_ID").

Integer: Ціле число, використовується для кількісних показників, що не мають дробової частини (наприклад, "connections_number").

Float: Дійсне число, використовується для кількісних показників з дробовою частиною (наприклад, "bandwidth", "signal_delay").

Array of Objects: Масив об'єктів, використовується для представлення списку однотипних елементів (наприклад, масив вузлів).

Object: Об'єкт, що містить пари ключ-значення, використовується для структурованого представлення даних (наприклад, "metrics", "connections").

3.1.2 Алгоритм генерації

Ключові Компоненти:

Імпорти та Глобальні Змінні: Використовуються для імпорту необхідних бібліотек та визначення глобальних змінних, таких як директорія даних та індикатори нормальних значень метрик.

Функції Для Роботи з JSON: Включають read_json та write_json для читання та запису файлів JSON відповідно.

Функції Аналізу та Оновлення Даних: Включають check_metrics, define_label, update_criticality, update_connections, update_metrics, randomize_data, check_network_integrity та допоміжні функції для аналізу, модифікації та оновлення даних в датасеті.

Генерація Даних: Процес, що включає використання функцій для створення модифікованих записів даних.

Етапи Обробки Даних:

Читання Початкових Даних: Використання функції read_json для отримання даних з початкового файлу JSON.

Аналіз Метрик Вузлів: Функція check_metrics визначає вузли з проблемними метриками, які потребують оновлення або відновлення.

Визначення Типу Процесу: Функція define_label визначає тип процесу (нормальний, обслуговування, кібератака) на основі стану вузлів та інших параметрів.

Оновлення Критичності Вузлів: Функція update_criticality оновлює рівень критичності вузлів на основі кількості з'єднань.

Оновлення Зв'язків та Метрик: Функції update_connections та update_metrics модифікують зв'язки між вузлами та метрики, відповідно до типу процесу.

Генерація Нових Даних: Функція randomize_data генерує нові дані на основі аналізу та оновлень.

Запис Оновлених Даних: Змінені дані записуються у новий файл JSON.

Повторення Процесу: Циклічний процес, де нові дані стають вхідними для наступної ітерації обробки даних.

Алгоритм:

У змінній «base_dir» міститься шлях до директорії, де розміщена необхідна структура дата сету. В директорії, перед початком запуску програми повинна бути така структура:

- base_dir_name // *директорія де буде розміщений дата сет*
  - input // *директорія яка містить початковий фал*
    - input.json // *файл який слугує шаблоном для генерації наступних станів топології*
  - output // *директорія яка міститиме згенерований датасет*
    - input.json // *дублюємо файл з «input»*
    - …

Файл «input.json» повинен містити базову початкову структуру (див. Додаток А та розділ 3.1.1) топології у нормальному стані функціонування (тобто поле «process_type» має значення «normal»), к-сть вузлів – не менше ніж три, але може бути більшою. Для тестового набору даних, було обрано 6 вузлів. Назва файлу може бути вибрана за власним бажанням, на генерацію наступних файлів вона не впливає. Початковий файл повинен бути продубльованим у директорію «output». Обидві директорії повинні не містити файлів (окрім «input.json» у «input»).

Робота алгоритму починається з запуску циклу, який генерує файли дата сету, к-сть ітерацій циклу – к-сть файлів в дата сеті.

В циклі за допомогою функції «find_first_json_file» в змінну «file_path» записується шлях до файлу шаблону «input.json» для генерації наступних станів топології. Далі в змінну «modified_data» записується результат функції «randomize_data», параметр цієї функції – це значення змінної «file_path», значення – це готова структура для наступного файлу в дата сеті. В змінній «output_file_name» формуємо ім’я вихідного файлу (строкове значення дати, наступного стану мережевої топології у форматі "pppp-мм-дд-гг-хх", не теж саме, що і час створення файлу). За допомогою функції «write_json» записуємо данні у «.json» файл в директорію «output». В цю функцію передаємо аргументи в наступній послідовності: «modified_data», «output_file_name», 'output'. Потім відбувається видалення файлу на основі якого був згенерований новий файл із директорії «input», далі записуємо щойно згенерований файл у директорію «input», для того щоб наступний файл був згенерований на основі вже нового файлу. Таким чином кожен файл дата сету буде згенерований на основі попереднього значення топології.

У функції «randomize_data» спочатку відбувається запис даних в змінну «entry» за допомогою функції «read_json», потім у змінну «problem_nodes» записується результат роботи функції «check_metrics» (передаємо аргументом змінну «entry»).

Метод «check_metrics» призначений для аналізу стану вузлів у мережевій топології, що представлені в об'єкті data_entry. Він використовується для ідентифікації вузлів, які можуть мати потенційні проблеми, на основі їх метрик. Ось детальний опис того, що відбувається в цьому методі:

Ініціалізація Списку Проблемних Вузлів: Створюється порожній список problem_nodes, який буде використовуватися для зберігання ідентифікаторів (ID) вузлів з проблемами.

Перебір Вузлів: Метод перебирає кожен вузол в масиві nodes, який є частиною об'єкта data_entry.

Перевірка Метрик Кожного Вузла: Для кожного вузла метод перевіряє наступні умови:

Якщо connections_number (кількість з'єднань) дорівнює нулю, це вказує на ізоляцію вузла. У такому випадку ID вузла додається до списку problem_nodes.

Якщо bandwidth (пропускна здатність) вузла менше або дорівнює значенню rebuild_limit у словнику normal_indicators для bandwidth, це вказує на критично низьку пропускну здатність. ID такого вузла також додається до списку problem_nodes.

Якщо signal_delay (затримка сигналу) вузла більше або дорівнює значенню rebuild_limit у словнику normal_indicators для signal_delay, це вказує на неприйнятно високу затримку сигналу. В такому випадку ID вузла додається до списку problem_nodes.

Повернення Результату: Після перевірки всіх вузлів, метод повертає список problem_nodes, який містить ідентифікатори всіх вузлів з виявленими проблемами.

Словник normal_indicators (рис. 3.1) визначає порогові значення для bandwidth та signal_delay, які використовуються для ідентифікації потенційних проблемних вузлів. Особливо важливими є значення rebuild_limit, які вказують на критичний поріг, нижче або вище якого вузол може потребувати додаткової уваги або втручання.

```
normal_indicators = {
        # values in megabytes
        "bandwidth": {
            "upper_limit": 500,
            "normal_limit": 100,
            "rebuild_limit": 10,
            "lower_limit": 0
        },
        # values in milliseconds
        "signal_delay": {
            "upper_limit": 10,
            "normal_limit": 30,
            "rebuild_limit": 1000,
            "lower_limit": 10000
        }
    }
```

Рис. 3.1. Словник «normal_indicators».

Після відпрацювання методу check_metrics у методі «randomize_data» відбувається запис результату роботи функції «check_network_integrity» (параметр «entry») у змінну «need_connections».

Метод «check_network_integrity» використовується для перевірки цілісності мережевої топології (перевіряє, чи немає ізольованих ділянок топології). Ось детальний опис його логіки роботи:

Ініціалізація Словника main_connections: Створюється словник для зберігання основних з'єднань між вузлами.

Перебір З'єднань Кожного Вузла:

Метод перебирає кожен вузол у data_entry['connections'].

Для кожного вузла (node_key) створюється словник connections, де node_key має значення True (означає, що вузол перевірено), а всі з'єднані з ним вузли (item_key) мають значення False (означає, що вузол не перевірено).

Виклик check_network_integrity_helper:

Використовується метод check_network_integrity_helper для аналізу з'єднань вузлів і оновлення стану перевірки в словнику connections.

Цикл while True продовжується до тих пір, поки не будуть перевірені всі з'єднання.

Зберігання Результатів Перевірки:

Отримані результати перевірки з'єднань зберігаються в словнику main_connections для кожного вузла.

Аналіз Необхідності Оновлення З'єднань:

Метод порівнює всі можливі з'єднання (data_entry['connections'].keys()) зі з'єднаннями кожного вузла (value у main_connections).

Якщо існують унікальні з'єднання, які не включені в value, вони додаються до списку need_updates.

Повернення Результату:

Якщо список need_updates порожній, метод повертає False, означаючи, що оновлення з'єднань не потрібне.

Якщо список need_updates містить елементи, метод повертає випадково обрану пару вузлів з need_updates для оновлення з'єднання між ними.

Загалом, check_network_integrity перевіряє цілісність мережевої топології та визначає, чи потрібно оновлювати з'єднання між вузлами для підтримки стабільності мережі.

Далі в методі «randomize_data» відбувається визначення типу процесу за допомогою функції «define_label» (параметри «problem_nodes», «entry['process_type']» - тип процесу з даних попереднього стану топології, «need_connections»), результат роботи записується в змінну «process_type».

Метод define_label використовується для визначення мітки процесу (або стану) в мережевій топології. Цей метод виконує наступні дії:

Перевірка Поточного Мітки:

Якщо поточна мітка label дорівнює 'cyber_attack', то існує 50% шанс (визначений за допомогою random.random() < 0.5), що мітка залишиться 'cyber_attack'. Якщо умова виконується, метод повертає 'cyber_attack'.

Перевірка Відсутності Проблемних Вузлів і Потреби в З'єднаннях:

Якщо немає проблемних вузлів (problem_nodes) і немає потреби в оновленні з'єднань (need_connections), то існує 5% шанс, що стан системи зміниться на 'cyber_attack' (визначено random.random() < 0.05).

Якщо ця умова не виконується, метод повертає 'normal', оскільки вважається, що система працює в нормальному режимі.

Обробка Інших Сценаріїв:

Якщо існують проблемні вузли або потреба в з'єднаннях, і поточна мітка 'maintenance', то існує 2% шанс (визначений random.random() < 0.02), що стан системи зміниться на 'cyber_attack'.

Якщо ця умова не виконується, метод повертає 'maintenance', оскільки вважається, що система працює в режимі обслуговування.

Загалом, метод define_label використовує випадковість та стан системи для визначення, чи система знаходиться під впливом кібератаки, працює нормально або перебуває в стані обслуговування.

Наступна дія в «randomize_data» - сформувати вже інші данні для нового елементу дата сету, на основі попередньо отриманих даних, та даних із попереднього стану мережі. Дані записуються в змінну «updated_entry», визначає ці дані функція «update_connections» (параметри «entry», «process_type», «problem_nodes», «need_connections»).

Метод update_connections отримує наступні параметри:

data_entry: Об'єкт JSON, що представляє поточний стан мережі, у тому числі вузли та їх з'єднання.

process_type: Тип процесу, який може бути 'maintenance', 'cyber_attack' або 'normal'.

problem_nodes: Список вузлів з проблемними метриками.

need_connections: Пара вузлів, між якими потрібно встановити з'єднання.

Обробка Стану 'Maintenance'

Якщо існують потреби у з'єднаннях (need_connections):

Для вузлів, визначених у need_connections, збільшується кількість з'єднань (connections_number).

Відповідні з'єднання між цими вузлами встановлюються як активні (значення 1).

Після оновлення з'єднань, викликається update_criticality, щоб переоцінити рівні критичності (criticality_mark) всіх вузлів на основі їхньої кількості з'єднань.

Обробка Стану 'Cyber_Attack'

Якщо process_type є 'cyber_attack', метод симулює кібератаку на вузли мережі.

Визначення Типу Атаки:

Випадкове число від 0 до 1 визначає тип атаки (attack_type).

Якщо attack_type <= 0.45 (45% шанс), імітується одиночна атака.

Якщо attack_type > 0.45 (55% шанс), імітується атака на кілька вузлів.

Вибір Вузлів для Атаки:

Для одиночної атаки:

Визначається критичність вузла (80% шанс, що вузол має високу критичність, 15% - середню, 5% - низьку).

Вибирається випадковий вузол із заданою критичністю.

Для атаки на кілька вузлів:

Визначається кількість атакованих вузлів (від 2 до 4).

Вузли вибираються на основі їх критичності з різними шансами (80% для вузлів із високою критичністю, 20% - для середньої, 10% - для низької).

Усунення З'єднань

Вибір Вузлів для Видалення З'єднань:

Спочатку вибираються атаковані вузли на основі їх критичності, як описано вище.

Для кожного атакованого вузла (node['node_ID'] in attacked_nodes):

Визначаються всі вузли, з якими є з'єднання.

Шанси для видалення з'єднань визначаються випадково, використовуючи метод random.sample. Це означає, що кількість і конкретні з'єднання для видалення вибираються випадковим чином, з урахуванням загальної кількості наявних з'єднань.

Якщо є кілька з'єднань, то вибирається випадкова кількість з'єднань для видалення, від одного до всіх наявних.

Видалення З'єднань:

Для кожного вибраного з'єднання, воно встановлюється як неактивне (значення 0).

Кількість з'єднань (connections_number) для кожного відповідного вузла зменшується на одиницю.

Шанси для усунення конкретних з'єднань не фіксовані й залежать від загальної кількості з'єднань вузла та випадкової вибірки.

Такий підхід дозволяє симулювати різні сценарії атак, де деякі вузли можуть втратити одне або кілька з'єднань, а інші - можуть втратити всі свої з'єднання.

Оновлення Метрик Вузлів

Викликається метод update_metrics, який оновлює метрики (bandwidth та signal_delay) кожного вузла в залежності від типу процесу (process_type) та ідентифікованих проблемних вузлів чи вузлів, атакованих під час кібератаки.

Maintenance (Обслуговування)

Вузли, що потребують відновлення (rebuild_nodes):

90% шанс збільшення bandwidth на випадкове число в діапазоні від поточного значення до 125% від цього значення.

90% шанс зменшення signal_delay на випадкове число в діапазоні від поточного значення до 75% від цього значення.

Інші вузли:

90% шанс збільшення bandwidth до значення, вищого за мінімальний ліміт відновлення.

90% шанс зменшення signal_delay до значення, нижчого за максимальний ліміт відновлення.

Normal (Нормальний стан)

Усі вузли:

90% шанс загальної зміни метрик:

Якщо шанс зміни bandwidth активовано (60%):

bandwidth змінюється на випадкове число в діапазоні від 75% до 125% від поточного значення.

Якщо нове значення bandwidth перевищує верхній ліміт, воно обмежується діапазоном між нормальним і верхнім лімітом.

Якщо шанс зміни signal_delay активовано (60%):

signal_delay змінюється на випадкове число в діапазоні від 75% до 125% від поточного значення.

Якщо нове значення signal_delay менше нижнього ліміту, воно обмежується діапазоном між нижнім і нормальним лімітом.

Cyber Attack (Кібератака)

Усі вузли:

bandwidth зменшується на випадкове число в діапазоні від поточного значення до 70% від цього значення.

signal_delay збільшується на випадкове число в діапазоні від поточного значення до 125% від цього значення.

Ці шанси та методи змін відображають різні сценарії, що можуть виникати під час обслуговування, нормальної експлуатації або під час кібератак, дозволяючи симулювати реалістичні зміни в мережевій топології.

Повернення Оновленого Стану Мережі

Оновлений об'єкт data_entry повертається як результат виконання методу.

Метод update_connections є ключовим у алгоритмі, оскільки він відповідає за моделювання різних станів мережі та динамічне оновлення структури та метрик мережевих вузлів.

Наступне, що відбувається в «randomize_data» - оновлення часової мітки (timestamp) для кожного запису в датасеті.

Встановлення process_type для оновленого запису (updated_entry):

updated_entry['process_type'] = process_type: Цей рядок коду встановлює тип процесу (process_type) для оновленого запису. Значення process_type може бути "normal", "maintenance" або "cyber_attack", в залежності від результатів попередньої обробки.

Отримання і оновлення часової мітки:

last_timestamp = datetime.strptime(updated_entry['timestamp'], '%d-%m-%Y %H:%M:%S'): Тут відбувається конвертація часової мітки зі стрічки у форматі '%d-%m-%Y %H:%M:%S' в об'єкт datetime. Це дає можливість виконувати арифметичні операції з датами і часом.

random_duration = timedelta(seconds=random.randint(180, 1800)): Генерується випадковий проміжок часу від 180 секунд (3 хвилини) до 1800 секунд (30 хвилин).

time = (last_timestamp + random_duration): До останньої часової мітки додається випадковий проміжок часу, що імітує реальний проміжок між подіями в датасеті.

new_time = time.strftime('%d-%m-%Y %H:%M:%S'): Оновлена часова мітка форматується назад у стрічку за допомогою strftime, щоб вона відповідала потрібному формату.

updated_entry['timestamp'] = new_time: Оновлена часова мітка зберігається у вхідному запису (updated_entry).

Форматування часової мітки для імені файлу:

new_time = time.strftime('%Y-%m-%d-%H-%M'): Тут часова мітка форматується в інший формат, який зазвичай використовується для іменування файлів (без секунд і з заміною дефісів на двокрапки).

Ці дії дозволяють створювати послідовність записів в датасеті, де кожен наступний запис відбувається через реалістичний проміжок часу після попереднього, імітуючи реальні умови зміни стану мережевої топології.

3.2 Розробка програмного прототипу

3.2.1 Вибір програмних інструментів

У процесі розробки прототипу рекурентної нейронної мережі для аналізу та класифікації станів мережевих топологій, ключове значення має вибір відповідних програмних інструментів. Цей вибір впливає на ефективність розробки, гнучкість в експериментах та якість кінцевих результатів. З огляду на складність задачі та необхідність обробки великої кількості даних, було прийнято рішення використати комбінацію перевірених та високопродуктивних інструментів, що відповідають сучасним стандартам у галузі штучного інтелекту та машинного навчання.

Середовище Розробки: Google Colab

Google Colab [23] було вибрано як основне середовище для розробки через кілька ключових переваг:

Доступність: Colab є безкоштовним і доступним онлайн, що забезпечує легкий доступ до потужних обчислювальних ресурсів без необхідності налаштування локального середовища.

Обчислювальні Ресурси: Colab пропонує безкоштовний доступ до GPU та TPU [24], що значно підвищує швидкість тренування моделей глибокого навчання.

Сумісність із Jupyter Notebook: Colab базується на Jupyter Notebook [25], який забезпечує інтуїтивно зрозумілу і гнучку робочу область для розробки та експериментів (рис. 3.2).

Мова Програмування: Python

Python обрано через його широке використання у сфері науки про дані та машинного навчання:

Широка Підтримка Бібліотек: Python підтримує велику кількість бібліотек для обробки даних, машинного навчання та глибокого навчання, що спрощує розробку.

Легкість використання і читання: Python відомий своїм чистим синтаксисом, що робить код легшим для написання та розуміння.

Спільнота та Документація: Мова має велику і активну спільноту, що забезпечує розгорнуту документацію та підтримку.

Основні Бібліотеки та Їх Роль

NumPy та Pandas

NumPy використовується для ефективної роботи з багатовимірними масивами. Це ключове значення в області науки про дані для маніпуляцій та аналізу чисельних даних.

Pandas застосовується для обробки та аналізу даних. Його функціональність для роботи з табличними даними, зокрема з JSON-файлами, є критичною для попередньої обробки даних.

TensorFlow та Keras

TensorFlow є відкритою платформою для машинного навчання, яка дозволяє ефективно створювати і тренувати моделі глибокого навчання.

Keras є високорівневим API, який працює поверх TensorFlow. Він забезпечує більш зручний інтерфейс для створення нейронних мереж, зокрема LSTM.

Scikit-learn

Scikit-learn використовується для попередньої обробки даних (наприклад, нормалізації) та для різноманітних завдань машинного навчання, включаючи кодування категорійних даних.

Keras Utilities та Layers

Keras Utilities (to_categorical): Допомагає перетворювати цільові змінні у формат, придатний для класифікаційних моделей.

Keras Layers (LSTM, Dense, Dropout): Ці компоненти використовуються для побудови архітектури LSTM мережі, де LSTM-шари є ключовими для врахування часових залежностей, а Dense та Dropout шари використовуються для побудови та оптимізації вихідного шару моделі.

TimeseriesGenerator

TimeseriesGenerator з бібліотеки Keras спрощує процес створення пакетів даних для навчання на послідовних даних, що є критичним для LSTM мереж.

Загалом, ці інструменти були обрані через їхню ефективність, гнучкість та широке прийняття в спільноті розробників, особливо у сфері машинного навчання та глибокого навчання.

3.2.2 Опис алгоритмів та обчислювального процесу

Розробка нейронної мережі для аналізу мережевих топологій вимагає ретельного підходу до вибору та опису алгоритмів та обчислювального процесу. Важливим аспектом є визначення логічних частин мережі, їх функцій, а також методів обробки та аналізу даних. У цьому підрозділі ми розглянемо ключові компоненти нашої нейронної мережі, її архітектуру та особливості роботи.

Завантаження та Агрегація Даних

Для роботи з датасетом з 20 000 json файлів ми використовуємо бібліотеки Python, такі як os, json, numpy та pandas. Ці інструменти дозволяють нам ефективно завантажувати та обробляти дані, готуючи їх до подальшої обробки нейронною мережею. Використання numpy та pandas є оптимальним для обробки великих обсягів даних і їх структурування у зручному для машинного навчання форматі.

Бібліотеки:

os: Для роботи з файловою системою. Використання цієї бібліотеки дозволяє нам здійснювати навігацію по директоріям та визначати шляхи до файлів.

json: Для читання та обробки JSON файлів. Використання цієї бібліотеки є ключовим, оскільки наші дані зберігаються у форматі JSON.

numpy та pandas: Для обробки та аналізу даних. numpy використовується для роботи з числовими масивами, а pandas - для ефективної обробки табличних даних.

tensorflow.keras.utils та sklearn.preprocessing: Для підготовки даних до навчання нейронної мережі, зокрема, для кодування категоріальних ознак та нормалізації числових даних.

Функція load_data

Функція load_data відповідає за завантаження та попередню обробку даних:

Читання файлів: Функція перебирає файли у вказаній директорії, фільтруючи тільки JSON файли (filename.endswith('.json')). Це забезпечує гнучкість у виборі джерела даних.

Збір даних у пакети: Дані з кожного файла додаються до списку batch_data. Використання пакетного підходу (batch_size=1000) дозволяє ефективно управляти пам'яттю та обчислювальними ресурсами, особливо при роботі з великими датасетами.

Попередня обробка даних

Основною частиною підготовки даних є кодування категоріальних ознак (node_ID, criticality_mark, process_type) та нормалізація числових даних (bandwidth, connections_number, signal_delay). Це забезпечує однорідність даних, що є критично важливим для ефективності навчання нейромережі. Використання LabelEncoder і MinMaxScaler з sklearn дозволяє нам трансформувати ці дані у формат, придатний для обробки нейронною мережею.

Обробка Часової Мітки: Для кожного запису у датасеті, часова мітка (timestamp) конвертується з текстового формату у числовий за допомогою функції datetime.strptime(), що забезпечує однорідність даних для аналізу.

Збір Даних за Вузлами: Для кожного вузла (node) у записі, створюється словник з ключовою інформацією: ідентифікатор вузла (node_ID), рівень критичності (criticality_mark), метрики вузла (bandwidth, connections_number, signal_delay), а також зв'язки з іншими вузлами.

Кодування Категоріальних Даних: Категоріальні ознаки, такі як node_ID і criticality_mark, кодуються за допомогою one-hot encoding або label encoding,

залежно від специфіки даних. Це перетворення необхідне, оскільки більшість алгоритмів машинного навчання краще працюють з числовими даними.

Застосування Min-Max Нормалізації: Числові метрики вузлів нормалізуються за допомогою Min-Max scaler, який трансформує дані так, що вони мають задані мінімальні та максимальні значення. Це покращує ефективність навчання моделі, оскільки усі числові ознаки будуть міститися в одному масштабі.

Створення Послідовностей

Оскільки ми працюємо з послідовними даними, важливим етапом є створення послідовностей для подачі в LSTM-мережу.

Імпорт необхідних Бібліотек: Використовується TimeseriesGenerator з бібліотеки TensorFlow, який є спеціалізованим інструментом для створення послідовностей часових рядів.

Визначення Функції для Створення Послідовностей:

df: Вхідні дані у форматі DataFrame.

target_column: Колонка, яка містить цільові значення для прогнозування.

sequence_length: Довжина кожної послідовності.

Формування Послідовностей:

Ітеруємо через DataFrame df, створюючи послідовності фіксованої довжини (sequence_length).

Кожна послідовність складається з sequence_length послідовних рядків даних.

X зберігає вхідні дані для кожної послідовності, а y - відповідні цільові значення.

Вибірка Характеристик та Цільових Значень:

X.append(...): Додавання в X послідовностей вхідних даних, ігноруючи останній стовпець (який зазвичай є цільовою змінною).

y.append(...): Додавання у y значення цільової змінної для останнього елемента в кожній послідовності.

Конвертація в NumPy Масиви:

Перетворення списків X та y в NumPy масиви, які є оптимальним форматом для обробки у нейронних мережах.

Результат

Функція повертає два масиви: X (вхідні дані для LSTM мережі) та y (цільові значення для кожної послідовності). Ці масиви використовуються для навчання LSTM мережі, де мережа намагається передбачити y, знаючи X.

Обробка пакетів даних та Поділ на Навчальну та Тестову Вибірки

У цій частині алгоритму дані розділяються на пакети та подаються для навчання та тестування моделі. Використання пакетного підходу до обробки даних та їх подальший поділ на навчальну та тестову вибірки є стандартною практикою в машинному навчанні. Це дозволяє перевірити, наскільки добре модель може узагальнювати свої знання на нових, невідомих раніше даних.

Процес Розділення

Імпорт необхідних бібліотек:

Використовується train_test_split з бібліотеки sklearn.model_selection для розділення даних.

Виконання Функції train_test_split:

X_train, X_test: Ці змінні містять вхідні дані для навчальної (train) та тестової (test) вибірок відповідно.

y_train, y_test: Ці змінні містять цільові значення для навчальної та тестової вибірок.

test_size=0.2: Визначає, що 20% даних буде використано для тестової вибірки, а решта 80% - для навчальної.

random_state=42: Забезпечує відтворюваність результатів шляхом використання фіксованого «зерна» випадковості.

Результат:

Навчальна вибірка (X_train, y_train) використовується для навчання моделі.

Тестова вибірка (X_test, y_test) використовується для перевірки ефективності моделі на невідомих даних.

Розділення даних на навчальну та тестову вибірки забезпечує неупереджену оцінку ефективності моделі. Це допомагає виявити перенавчання (overfitting), коли модель добре працює на навчальних даних, але погано - на нових даних.

Створення та Навчання LSTM Моделі

Архітектура LSTM моделі, реалізована з використанням Sequential з tensorflow.keras, включає в себе декілька шарів LSTM і Dense, а також шари Dropout для запобігання перенавчання. Вибір LSTM обумовлений їх здатністю ефективно обробляти послідовні дані, що є ключовим для нашої задачі. Використання активаційної функції relu та sigmoid забезпечує необхідну нелінійність у моделі.

Структура Моделі

Визначення Вхідних Даних:

input_shape = (X_train.shape[1], X_train.shape[2]): Визначає форму даних, що подаються на вхід LSTM шару.

Побудова Моделі:

Використовується Sequential для створення послідовного стеку шарів.

Додаються два шари LSTM з 50 нейронами кожен та активаційною функцією relu.

Dropout(0.2): Запобігає перенавчанню, випадково відключаючи 20% нейронів під час навчання.

Dense(1, activation='sigmoid'): Вихідний шар із одним нейроном для класифікації (наприклад, кібератака чи нормальна діяльність).

Компіляція Моделі:

Оптимізатор adam: Популярний вибір для глибокого навчання через його ефективність у роботі з великими датасетами.

Функція втрат sparse_categorical_crossentropy: Використовується для багатокласової класифікації.

Метрика accuracy: Дозволяє оцінити точність класифікації моделі.

Навчання Моделі:

epochs=50: Вказує, що модель буде тренуватися на 50 ітераціях по всьому датасету.

batch_size=64: Кількість зразків, які обробляються перед оновленням моделі.

validation_data=(X_test, y_test): Дані для валідації моделі під час навчання.

verbose=2: Вказує на рівень деталізації виведення інформації під час тренування.

Вибір Методології

Обраний підхід з використанням LSTM та вищевказаних параметрів навчання є оптимальним для цієї задачі з кількох причин:

Обробка Послідовностей: LSTM ефективно впорається з послідовними даними, які є характерними для мережевої топології.

Запобігання Перенавчанню: Використання слоїв Dropout допомагає уникнути перенавчання моделі.

Адаптивність: Оптимізатор Adam автоматично налаштовує швидкість навчання, що покращує процес тренування.

Ефективність: Параметри epochs та batch_size були вибрані для ефективного балансу між швидкістю тренування та точністю моделі.

Вхідні та Вихідні Дані LSTM Нейромережі

Вхідні Дані

Формат Даних:

Вхідні дані представлені у вигляді векторів, які генеруються на основі JSON файлів. Кожен файл описує стан мережевої топології в певний момент часу.

Розмір та Структура:

Кожен вектор включає часову мітку (timestamp), кодовані категоріальні ознаки (node_ID, criticality_mark, process_type), нормалізовані числові дані (bandwidth, connections_number, signal_delay) та дані про з'єднання між вузлами (conn_{node_ID}).

Розмір вхідних даних залежить від кількості вузлів в мережі та визначеної довжини послідовності (sequence_length).

Подача Даних:

Дані подаються у форматі послідовностей, де кожна послідовність містить sequence_length векторів. Це дозволяє LSTM мережі враховувати часові залежності між станами мережі.

Вихідні Дані

Формат Виводу:

Модель повертає вектор ймовірностей, що відображає можливість належності кожної послідовності до одного із класів (наприклад, normal, maintenance, cyber_attack).

Тип Даних:

Вивід представлений у вигляді одновимірного масиву, де кожен елемент є числом від 0 до 1, вказуючи на ймовірність належності до відповідного класу.

Ймовірність Класифікації:

Наприклад, для послідовності, модель може повернути [0.05, 0.90, 0.05], що означає 5% ймовірність normal, 90% ймовірність maintenance і 5% ймовірність cyber_attack.

Повернення Результату:

Результати подаються в послідовностях, аналогічно вхідним даним. Кожна послідовність містить прогнозовані ймовірності для кожного часового кроку в межах послідовності.

3.3 Тестування та валідація моделі

Підготовка Даних

Датасет було розділено на навчальний (80%) та тестовий (20%) набори даних. Важливою частиною підготовки була перевірка на репрезентативність даних у обох наборах.

Валідація Моделі

Валідація моделі включала:

Перехресна Перевірка: Для оцінки ефективності моделі використовувалась перехресна перевірка.

Аналіз Помилок: Проведено детальний аналіз помилок для виявлення випадків невірної класифікації.

Оцінка Точності: Використовувались метрики точності, матриці невідповідностей, та F1-скору.

Тестування Моделі

Використання Тестового Набору: Модель була протестована на тестовому наборі.

Оцінка Прогнозів: Аналіз прогнозів моделі на точність визначення станів мережі.

Сценарії Кібератак: Особлива увага була приділена випадкам кібератак.

Результати Точності

Модель показала наступні результати точності:

- Точність визначення "нормального" стану: 78%
- Точність визначення "обслуговування": 74%
- Точність виявлення "кібератак": 80%

Середній показник точності моделі склав приблизно 77%.

3.4 Висновки та пропозиції щодо подальшого вдосконалення

3.4.1 Перспективи розвитку та можливі поліпшення

Розроблена LSTM модель для ідентифікації станів мережевої топології має значний потенціал для подальшого розвитку та удосконалення. Декілька ключових напрямків можуть бути визначені для підвищення ефективності та точності моделі.

Збільшення Розміру Датасету

Більший обсяг даних: Збільшення обсягу датасету може покращити здатність моделі до генералізації та адаптації до різноманітних сценаріїв.

Відкорегування Вагів

Тонка настройка параметрів: Детальне відкорегування ваг і гіперпараметрів моделі може призвести до кращих результатів. Експерименти з

різними конфігураціями мережі, включаючи кількість шарів та нейронів, можуть значно вплинути на точність прогнозування.

Розширення Топології

Складніша топологія: Розширення моделі до включення декілька сотень або тисяч вершин може сприяти більш реалістичному моделюванню та аналізу складних мереж.

Додавання Метрик

Більше метрик: Включення додаткових метрик, таких як якість сервісу, навантаження на вузли, може дати більше інформації для аналізу та класифікації станів мережі.

Покращення Алгоритму Генерації

Ускладнення поведінки: Розширення алгоритму генерації датасету для відображення більш складних сценаріїв поведінки мережі, що включає в себе каскадні збої та адаптивні зміни.

Використання Реальних Даних

Збір реальних даних: Хоча синтетичний датасет є корисним для первинного навчання та тестування, збір реальних даних з мережевих систем може значно покращити реалістичність та актуальність моделі.

Заключні Думки

Враховуючи потенціал LSTM мереж у аналізі часових рядів, ці поліпшення можуть значно розширити застосування та ефективність моделі. Подальші дослідження та експерименти дозволять виявити найбільш оптимальні стратегії для удосконалення роботи моделі у контексті аналізу мережевих топологій.

3.4.2 Застосування моделі в реальних умовах

Розроблена LSTM модель для ідентифікації та класифікації станів мережевої топології має широкий спектр потенційних застосувань у реальних умовах. Цей підрозділ описує ключові сфери, де модель може бути використана для підвищення ефективності мережевих систем.

Моніторинг та Прогнозування в IT-Інфраструктурі

Проактивне виявлення збоїв: Модель може бути інтегрована в системи моніторингу IT-інфраструктури для раннього виявлення потенційних збоїв або нештатних ситуацій, дозволяючи адміністраторам систем швидко реагувати на проблеми.

Прогнозування навантаження на мережу: Застосування моделі для прогнозування майбутніх тенденцій в навантаженні на мережу дозволить планувати ресурси більш ефективно.

Кібербезпека

Виявлення кібератак: Модель може бути використана для виявлення аномалій у поведінці мережі, що можуть вказувати на кібератаки, такі як DDoS або проникнення в систему.

Аналіз загроз: Аналізуючи історію змін у мережі, модель допоможе виявити потенційні вразливі вузли в мережевій структурі.

Оптимізація Мережевих Ресурсів

Адаптивна оптимізація мережі: Модель може бути використана для адаптивного управління ресурсами мережі, реагуючи на зміни в трафіку та використанні ресурсів.

Інтеграція з Іншими Системами

Співпраця з IoT та Edge Computing: Інтеграція моделі з IoT пристроями та edge обчислювальними платформами для ефективного управління даними та мережевими ресурсами.

Автоматизація та Підтримка Рішень

Підтримка прийняття рішень: Модель може бути використана як частина більш широкої системи підтримки прийняття рішень для мережевих інженерів та адміністраторів.

Застосування цієї LSTM моделі в реальних умовах відкриває широкі можливості для підвищення надійності, безпеки та ефективності мережевих систем. Інтеграція моделі з сучасними IT-інфраструктурами та її адаптація до

конкретних вимог бізнесу та технічних умов може значно сприяти оптимізації роботи мереж та забезпеченню їх стабільності.

ВИСНОВКИ

Ця магістерська робота присвячена розробці та дослідженню рекурентної нейронної мережі з довгою короткотривалою пам'яттю (LSTM) для розпізнавання кіберзагроз у великомасштабних мережевих системах. Робота включає теоретичний огляд, розробку концепції, реалізацію прототипу та його тестування.

Розділ I: Було детально розглянуто теоретичні аспекти кіберзагроз, властивості адаптивних мережевих топологій, а також роль та потенціал рекурентних нейронних мереж у розпізнаванні таких загроз. Проведено порівняльний аналіз існуючих рішень, який підкреслив значимість інноваційних підходів у сфері кібербезпеки.

Розділ II: Розроблено концепцію адаптивної мережевої топології та пропоновану модель LSTM. Описано метод навчання нейронної мережі, що включає підготовку даних, їх передобробку, створення послідовностей, та тренування моделі.

Розділ III: Здійснено реалізацію та тестування прототипу нейронної мережі. Підготовка дата-сету з використанням алгоритму генерації була критично важливою для забезпечення якості тренування та тестування моделі. Результати тестування та валідації моделі демонструють її ефективність та надійність у розпізнаванні кіберзагроз.

Основна увага була приділена розробці та аналізу алгоритмів, спрямованих на розпізнавання кіберзагроз за допомогою LSTM нейронної мережі.

Використано сучасні інструменти та технології, включаючи Google Colab, Python, бібліотеки NumPy, Pandas, TensorFlow, Keras та Scikit-learn.

Модель показала задовільні результати точності: 78% для "нормального" стану, 74% для "обслуговування", та 80% для виявлення "кібератак", з середнім показником точності близько 77%.

Середовище Розробки (Google Colab): Вибір цього середовища зумовлений його доступністю, наявністю безкоштовних GPU/TPU та сумісністю з Jupyter Notebook.

Мова Програмування (Python): Обрано за її універсальність, велику підтримку бібліотек та спільноту. NumPy і Pandas для ефективної обробки даних. TensorFlow і Keras для створення та тренування глибоких нейронних мереж. Scikit-learn для попередньої обробки даних та кодування категорійних ознак. Keras Utilities та Layers для оптимізації вихідного шару моделі. TimeseriesGenerator для створення пакетів даних для LSTM.

В роботі окреслено можливі шляхи для подальшого вдосконалення моделі, включаючи збільшення обсягу дата-сету, детальніше налаштування ваг моделі, розширення числа метрик для аналізу, а також збір реальних даних.

Модель має значний потенціал для використання в реальних умовах, зокрема у сферах моніторингу та прогнозування в IT-інфраструктурі, кібербезпеці, оптимізації мережевих ресурсів, а також у якості інструменту підтримки прийняття рішень.

В цілому, розроблена LSTM модель є важливим кроком у напрямку створення ефективних та адаптивних рішень для розпізнавання кіберзагроз у мережевих системах. Результати дослідження та тестування підтверджують її потенціал та відкривають нові перспективи для подальших досліджень та застосування в реальному світі. Робота вносить важливий вклад у розвиток методів захисту мережевих систем та кібербезпеки загалом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Sarker, I. H. (2021). Deep Cybersecurity: A Comprehensive Overview from Neural Network and Deep Learning Perspective. Springer. веб-сайт. URL: https://link.springer.com/article/10.1007/s42979-021-00535-6

2. Podder, P., Bharati, S., Mondal, M. R. H., Paul, P. K., & Kose, U. (2021). Artificial Neural Network for Cybersecurity: A Comprehensive Review. arXiv. веб-сайт. URL: https://arxiv.org/ftp/arxiv/papers/2107/2107.01185.pdf

3. Kan, X., Fan, Y., Fang, Z., et al. (2021). A novel IoT network intrusion detection approach based on Adaptive Particle Swarm Optimization Convolutional Neural Network. Information Sciences, 568, 147-162. DOI: 10.1016/j.ins.2021.03.060. веб-сайт. URL: https://www.sciencedirect.com/science/article/abs/pii/S1063520323000660

4. HackerOne. What is Vulnerability Assessment: Benefits, Tools, and Process. веб-сайт. URL: https://www.hackerone.com/knowledge-center/what-vulnerability-assessment-benefits-tools-and-process

5. TechTarget. How to Implement Network Segmentation for Better Security. веб-сайт. URL: https://www.techtarget.com/searchnetworking/tip/How-to-implement-network-segmentation-for-better-security

6. Thales Group. What is Network Encryption? веб-сайт. URL: https://www.thalesgroup.com/faq/encryption/what-network-encryption

7. Kentik. Network Architecture. веб-сайт. URL: https://www.kentik.com/kentipedia/network-architecture/

8. ZenArmor. What is Network Topology? веб-сайт. URL: https://www.zenarmor.com/docs/network-basics/what-is-network-topology

9. Analytics Vidhya. A Brief Overview of Recurrent Neural Networks (RNN). веб-сайт. URL: https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/

10. Mad Devs. Artificial Intelligence in Cybersecurity. веб-сайт. URL: https://maddevs.io/blog/artificial-intelligence-in-cybersecurity/

11. International Journal of Creative Research Thoughts. веб-сайт. URL: https://ijcrt.org/papers/IJCRT2107706.pdf

12. Neurocomputing. веб-сайт. URL: https://www.sciencedirect.com/science/article/abs/pii/S092523120400044X

13. SAGE Journals. веб-сайт. URL: https://journals.sagepub.com/doi/full/10.1177/1550147720971517

14. Wikipedia. Орієнтований граф. веб-сайт. URL: https://uk.wikipedia.org/wiki/Орієнтований_граф

15. CISA. Understanding Denial of Service Attacks. веб-сайт. URL: https://www.cisa.gov/news-events/news/understanding-denial-service-attacks

16. ScienceDirect. веб-сайт. URL: https://www.sciencedirect.com/science/article/abs/pii/S1063520323000660

17. Analytics Vidhya. Introduction to Long Short-Term Memory (LSTM). веб-сайт. URL: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

18. Christopher Olah's Blog. Understanding LSTM Networks. веб-сайт. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

19. Towards Data Science. Understanding Backpropagation Algorithm. веб-сайт. URL: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

20. Keras API. Adam Optimizer. веб-сайт. URL: https://keras.io/api/optimizers/adam/

21. Wikipedia. Python Programming Language. веб-сайт. URL: https://en.wikipedia.org/wiki/Python_(programming_language)

22. JSON.org. веб-сайт. URL: https://www.json.org/json-en.html

23. Google Colab. веб-сайт. URL: https://colab.google/

24. Analytics Vidhya. Evolution of TPUs and GPUs in Deep Learning Applications. веб-сайт. URL: https://www.analyticsvidhya.com/blog/2022/08/evolution-of-tpus-and-gpus-in-deep-learning-applications/

25. Jupyter Notebook Beginner Guide. веб-сайт. URL: https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html

26. TensorFlow. Introduction to TensorFlow. веб-сайт. URL: https://www.tensorflow.org/learn

27. TensorFlow. TensorFlow is an end-to-end open source platform for machine learning. Веб-сайт. URL: https://www.tensorflow.org/overview

28. Analytics Vidhya. A Comprehensive Guide to Google Colab. веб-сайт. URL: https://www.analyticsvidhya.com/blog/2020/03/google-colab-machine-learning-deep-learning/

29. Python.Land. Python And AI: Why Learning Python Might Be A Good Idea. веб-сайт. URL: https://python.land/python-and-ai

30. ScienceDaily. Future of brain-inspired AI as Python code library passes major milestone. веб-сайт. URL: https://www.sciencedaily.com/releases/2023/11/231117102505.htm

ДОДАТКИ

Додаток А. Приклади дата сету

Нормальне функціонування мережі:


```
{
  "timestamp": "17-12-2023 22:38:49",
  "process_type": "normal",
  "nodes": [
    {
      "node_ID": "node_1",
      "criticality_mark": "high",
      "metrics": {
        "bandwidth": 36.747514427440535,
        "connections_number": 3,
        "signal_delay": 11.266174766952116
      }
    },
    {
      "node_ID": "node_2",
      "criticality_mark": "low",
      "metrics": {
        "bandwidth": 108.7260727938836,
        "connections_number": 1,
        "signal_delay": 25.08741119198253
      }
    },
    {
      "node_ID": "node_3",
      "criticality_mark": "high",
      "metrics": {
        "bandwidth": 49.045823775044234,
        "connections_number": 3,
        "signal_delay": 24.92406399139379
      }
    },
    {
      "node_ID": "node_4",
      "criticality_mark": "medium",
      "metrics": {
        "bandwidth": 21.07049818043833,
        "connections_number": 2,
        "signal_delay": 50.15552033235315
      }
    },
    {
      "node_ID": "node_5",
      "criticality_mark": "medium",
      "metrics": {
        "bandwidth": 65.31099280799066,
```

```
            "connections_number": 2,
            "signal_delay": 55.28452042404891
        }
    },
    {
        "node_ID": "node_6",
        "criticality_mark": "low",
        "metrics": {
            "bandwidth": 125.12650056960685,
            "connections_number": 1,
            "signal_delay": 42.40127384642765
        }
    }
],
"connections": {
    "node_1": {
        "node_1": 1,
        "node_2": 1,
        "node_3": 1,
        "node_4": 1,
        "node_5": 0,
        "node_6": 0
    },
    "node_2": {
        "node_1": 1,
        "node_2": 1,
        "node_3": 0,
        "node_4": 0,
        "node_5": 0,
        "node_6": 0
    },
    "node_3": {
        "node_1": 1,
        "node_2": 0,
        "node_3": 1,
        "node_4": 1,
        "node_5": 1,
        "node_6": 0
    },
    "node_4": {
        "node_1": 1,
        "node_2": 0,
        "node_3": 1,
        "node_4": 1,
        "node_5": 0,
        "node_6": 0
    },
    "node_5": {
        "node_1": 0,
        "node_2": 0,
        "node_3": 1,
        "node_4": 0,
```

```
            "node_5": 1,
            "node_6": 1
        },
        "node_6": {
            "node_1": 0,
            "node_2": 0,
            "node_3": 0,
            "node_4": 0,
            "node_5": 1,
            "node_6": 1
        }
    }
}
```


Кібератака:


```
{
    "timestamp": "17-12-2023 22:55:49",
    "process_type": "cyber_attack",
    "nodes": [
        {
            "node_ID": "node_1",
            "criticality_mark": "high",
            "metrics": {
                "bandwidth": 36.028587919907444,
                "connections_number": 1,
                "signal_delay": 13.285909981961428
            }
        },
        {
            "node_ID": "node_2",
            "criticality_mark": "low",
            "metrics": {
                "bandwidth": 90.52266683252157,
                "connections_number": 1,
                "signal_delay": 25.591719269864832
            }
        },
        {
            "node_ID": "node_3",
            "criticality_mark": "high",
            "metrics": {
                "bandwidth": 34.738128503018885,
                "connections_number": 2,
                "signal_delay": 25.477179188472274
            }
        },
        {
            "node_ID": "node_4",
            "criticality_mark": "medium",
            "metrics": {
                "bandwidth": 16.74134205670046,
                "connections_number": 1,
```

```
            "signal_delay": 50.837334440882834
        }
    },
    {
        "node_ID": "node_5",
        "criticality_mark": "medium",
        "metrics": {
            "bandwidth": 57.59387364516274,
            "connections_number": 2,
            "signal_delay": 59.64173791319674
        }
    },
    {
        "node_ID": "node_6",
        "criticality_mark": "low",
        "metrics": {
            "bandwidth": 118.88157701240044,
            "connections_number": 1,
            "signal_delay": 42.801547939039374
        }
    }
],
"connections": {
    "node_1": {
        "node_1": 1,
        "node_2": 1,
        "node_3": 0,
        "node_4": 0,
        "node_5": 0,
        "node_6": 0
    },
    "node_2": {
        "node_1": 1,
        "node_2": 1,
        "node_3": 0,
        "node_4": 0,
        "node_5": 0,
        "node_6": 0
    },
    "node_3": {
        "node_1": 0,
        "node_2": 0,
        "node_3": 1,
        "node_4": 1,
        "node_5": 1,
        "node_6": 0
    },
    "node_4": {
        "node_1": 0,
        "node_2": 0,
        "node_3": 1,
        "node_4": 1,
```

```
            "node_5": 0,
            "node_6": 0
        },
        "node_5": {
            "node_1": 0,
            "node_2": 0,
            "node_3": 1,
            "node_4": 0,
            "node_5": 1,
            "node_6": 1
        },
        "node_6": {
            "node_1": 0,
            "node_2": 0,
            "node_3": 0,
            "node_4": 0,
            "node_5": 1,
            "node_6": 1
        }
    }
}
```


Процес відновлення мережі:


```
{
    "timestamp": "17-12-2023 23:07:09",
    "process_type": "maintenance",
    "nodes": [
        {
            "node_ID": "node_1",
            "criticality_mark": "low",
            "metrics": {
                "bandwidth": 27.351886672687982,
                "connections_number": 1,
                "signal_delay": 502.9662822055271
            }
        },
        {
            "node_ID": "node_2",
            "criticality_mark": "medium",
            "metrics": {
                "bandwidth": 22.87829329915648,
                "connections_number": 2,
                "signal_delay": 416.56063072105854
            }
        },
        {
            "node_ID": "node_3",
            "criticality_mark": "medium",
            "metrics": {
                "bandwidth": 15.277622077739906,
                "connections_number": 2,
                "signal_delay": 897.8055895832865
```

```
        }
      },
      {
        "node_ID": "node_4",
        "criticality_mark": "low",
        "metrics": {
          "bandwidth": 12.943361294747396,
          "connections_number": 1,
          "signal_delay": 50.837334440882834
        }
      },
      {
        "node_ID": "node_5",
        "criticality_mark": "high",
        "metrics": {
          "bandwidth": 57.361353987816734,
          "connections_number": 3,
          "signal_delay": 175.81469284702823
        }
      },
      {
        "node_ID": "node_6",
        "criticality_mark": "low",
        "metrics": {
          "bandwidth": 107.27258142324042,
          "connections_number": 1,
          "signal_delay": 993.1109176398321
        }
      }
    ],
    "connections": {
      "node_1": {
        "node_1": 1,
        "node_2": 1,
        "node_3": 0,
        "node_4": 0,
        "node_5": 0,
        "node_6": 0
      },
      "node_2": {
        "node_1": 1,
        "node_2": 1,
        "node_3": 0,
        "node_4": 0,
        "node_5": 1,
        "node_6": 0
      },
      "node_3": {
        "node_1": 0,
        "node_2": 0,
        "node_3": 1,
        "node_4": 1,
```

```
            "node_5": 1,
            "node_6": 0
        },
        "node_4": {
            "node_1": 0,
            "node_2": 0,
            "node_3": 1,
            "node_4": 1,
            "node_5": 0,
            "node_6": 0
        },
        "node_5": {
            "node_1": 0,
            "node_2": 1,
            "node_3": 1,
            "node_4": 0,
            "node_5": 1,
            "node_6": 1
        },
        "node_6": {
            "node_1": 0,
            "node_2": 0,
            "node_3": 0,
            "node_4": 0,
            "node_5": 1,
            "node_6": 1
        }
    }
}
```

Додаток Б. Лістинг програми генерації дата сету

```
import json
import os
import random
from datetime import datetime, timedelta

base_dir = " directory//path"
new_time = "
normal_indicators = {
        # values in megabytes
        "bandwidth": {
            "upper_limit": 500,
            "normal_limit": 100,
            "rebuild_limit": 10,
            "lower_limit": 0
        },
        # values in milliseconds
        "signal_delay": {
            "upper_limit": 10,
            "normal_limit": 30,
            "rebuild_limit": 1000,
            "lower_limit": 10000
        }
    }

def read_json(file_path):
    with open(file_path, 'r') as file:
        return json.load(file)

def write_json(data, file_path, tag):
    global base_dir
    if tag == 'input':
        full_path = os.path.join(base_dir, 'input', file_path)
    else:
        full_path = os.path.join(base_dir, 'output', file_path)

    with open(full_path, 'w') as file:
        json.dump(data, file, indent=4)

def check_metrics(data_entry):
    problem_nodes = []

    for node in data_entry['nodes']:
        if node['metrics']['connections_number'] == 0:
            problem_nodes.append(node['node_ID'])
            continue
        if node['metrics']['bandwidth'] <= normal_indicators['bandwidth']['rebuild_limit']:
            problem_nodes.append(node['node_ID'])
            continue
```

```
            if node['metrics']['signal_delay'] >= normal_indicators['signal_delay']['rebuild_limit']:
                problem_nodes.append(node['node_ID'])
                continue
        return problem_nodes

    def define_label(problem_nodes, label, need_connections):
        if (label == 'cyber_attack') and (random.random() < 0.5):
            return 'cyber_attack'
        else:
            if not problem_nodes and not need_connections:
                if random.random() < 0.05:
                    return 'cyber_attack'
                return 'normal'
            else:
                if (label == 'maintenance') and (random.random() < 0.02):
                    return 'cyber_attack'
                return 'maintenance'

    def update_criticality(data_entry):
        connections_numbers = [node['metrics']['connections_number'] for node in
data_entry['nodes']]
        max_connections = max(connections_numbers)
        min_connections = min(connections_numbers)

        # Count how many times the maximum value occurs
        max_count = connections_numbers.count(max_connections)

        # Update criticality_mark for nodes with the highest connections_number value
        for node in data_entry['nodes']:
            if node['metrics']['connections_number'] == max_connections:
                if max_count > len(data_entry['nodes']) / 2:
                    node['criticality_mark'] = "medium"
                else:
                    node['criticality_mark'] = "high"

        # Find intermediate values of connections_number
        intermediate_values = set(connections_numbers) - {max_connections, min_connections}

        # Update criticality_mark for nodes with the lowest and intermediate values of
connections_number
        for node in data_entry['nodes']:
            if node['metrics']['connections_number'] == min_connections:
                node['criticality_mark'] = "low"
            elif node['metrics']['connections_number'] in intermediate_values:
                node['criticality_mark'] = "medium"
        return data_entry

    def update_connections(data_entry, process_type, problem_nodes, need_connections):
        if process_type == 'maintenance':
            if need_connections:
                for node in data_entry['nodes']:
```

```
                if node['node_ID'] == need_connections[0] or node['node_ID'] == need_connections[1]:
                    node['metrics']['connections_number'] += 1

            data_entry['connections'][need_connections[0]][need_connections[1]] = 1
            data_entry['connections'][need_connections[1]][need_connections[0]] = 1

        data_entry = update_criticality(data_entry)

    attacked_nodes = []
    if process_type == 'cyber_attack':
        attack_type = random.random()
        criticality = ''

        # single attack simulation
        if attack_type <= 0.45:
            random_node_id = None
            while random_node_id == None:
                criticality_mark = random.random()
                if criticality_mark <= 0.8:
                    criticality = "high"
                elif 0.8 < criticality_mark < 0.95:
                    criticality = "medium"
                else:
                    criticality = "low"
                criticality_nodes = [node['node_ID'] for node in data_entry['nodes'] if node['criticality_mark'] == criticality]
                random_node_id = random.choice(criticality_nodes) if criticality_nodes else None
            attacked_nodes.append(random_node_id)

        # simulating an attack on multiple nodes
        else:
            count = random.randint(2, 4)
            added = 0
            while count > added:
                for node in data_entry['nodes']:
                    if added == count:
                        break
                    elif node['criticality_mark'] == "high" and random.random() <= 0.8:
                        attacked_nodes.append(node['node_ID'])
                        added += 1
                    elif node['criticality_mark'] == "medium" and random.random() <= 0.2:
                        attacked_nodes.append(node['node_ID'])
                        added += 1
                    elif node['criticality_mark'] == "low" and random.random() <= 0.1:
                        attacked_nodes.append(node['node_ID'])
                        added += 1

        # removing connections
        for node in data_entry['nodes']:
            if node['node_ID'] in attacked_nodes:
                selected_node_id = node['node_ID']
```

```
                target_node_ids = []
                for key, value in data_entry['connections'][selected_node_id].items():
                    if value == 1 and key != selected_node_id:
                        target_node_ids.append(key)
                if not target_node_ids:
                    continue
                target_node_ids = random.sample(target_node_ids, random.randint(1,
len(target_node_ids)))
                for target_node in target_node_ids:
                    data_entry['connections'][selected_node_id][target_node] = 0
                    data_entry['connections'][target_node][selected_node_id] = 0
                    for updated_node in data_entry['nodes']:
                        if updated_node['node_ID'] == selected_node_id or updated_node['node_ID']
== target_node:
                            updated_node['metrics']['connections_number'] -= 1


        data_entry = update_metrics(data_entry, process_type, problem_nodes, attacked_nodes)
        return data_entry

    def update_metrics(data_entry, process_type, rebuild_nodes, attacked_nodes):
        # rebuild_nodes nodes whose metrics have reached values for restructuring
        global normal_indicators

        if(process_type == 'maintenance'):
            for node in data_entry['nodes']:
                if node['node_ID'] in rebuild_nodes:
                    if random.random() <= 0.9:
                        node['metrics']['bandwidth'] = random.uniform(node['metrics']['bandwidth'],
node['metrics']['bandwidth'] * 1.25)
                    if random.random() <= 0.9:
                        node['metrics']['signal_delay'] =
random.uniform(node['metrics']['signal_delay'],node['metrics']['signal_delay'] * 0.75)
                else:
                    if random.random() <= 0.9:
                        node['metrics']['bandwidth'] =
random.uniform(normal_indicators['bandwidth']['rebuild_limit'], node['metrics']['bandwidth'])
                    if random.random() <= 0.9:
                        node['metrics']['signal_delay'] = random.uniform(node['metrics']['signal_delay'],
normal_indicators['signal_delay']['rebuild_limit'])
            return data_entry

        if (process_type == 'normal'):
            for node in data_entry['nodes']:
                if random.random() <= 0.9:
                    if random.random() <= 0.6:
                        tempo_val = node['metrics']['bandwidth'] * random.uniform(0.75, 1.25)
                        if tempo_val >= normal_indicators['bandwidth']['upper_limit']:
                            tempo_val = random.uniform(normal_indicators['bandwidth']['normal_limit'],
normal_indicators['bandwidth']['upper_limit'])
                        node['metrics']['bandwidth'] = tempo_val
```

```
                if random.random() <= 0.6:
                    tempo_val = node['metrics']['signal_delay'] * random.uniform(0.75, 1.25)
                    if tempo_val <= normal_indicators['signal_delay']['upper_limit']:
                        tempo_val = random.uniform(normal_indicators['signal_delay']['upper_limit'], normal_indicators['signal_delay']['normal_limit'])
                    node['metrics']['signal_delay'] = tempo_val
            return data_entry

        if (process_type == 'cyber_attack'):
            for node in data_entry['nodes']:
                node['metrics']['bandwidth'] = random.uniform(node['metrics']['bandwidth'], node['metrics']['bandwidth'] * 0.7)
                node['metrics']['signal_delay'] = random.uniform(node['metrics']['signal_delay'], node['metrics']['signal_delay'] * 1.25)
            return data_entry

    def randomize_data(file_path):
        entry = read_json(file_path)
        problem_nodes = check_metrics(entry)
        need_connections = check_network_integrity(entry)
        process_type = define_label(problem_nodes, entry['process_type'], need_connections)
        updated_entry = update_connections(entry, process_type, problem_nodes, need_connections)

        # Update process_type and timestamp
        updated_entry['process_type'] = process_type
        last_timestamp = datetime.strptime(updated_entry['timestamp'], '%d-%m-%Y %H:%M:%S')
        random_duration = timedelta(seconds=random.randint(180, 1800))  # 3 mins to 30 mins

        global new_time
        time = (last_timestamp + random_duration)
        new_time = time.strftime('%d-%m-%Y %H:%M:%S')
        updated_entry['timestamp'] = new_time
        new_time = time.strftime('%Y-%m-%d-%H-%M')

        return updated_entry

    def check_network_integrity(data_entry):
        main_connections = {}
        for node_key, node_value in data_entry['connections'].items():
            connections = {}
            connections[node_key] = True
            for item_key, connect in node_value.items():
                if connect == 1 and item_key != node_key:
                    connections[item_key] = False

            while True:
                result = check_network_integrity_helper(connections, data_entry)
                if not result:
                    break
```

```
                connections.update(result)

            main_connections[node_key] = sorted(connections.keys())

        all_connectinos = data_entry['connections'].keys()
        need_updates = []
        for key, value in main_connections.items():
            unique_keys = set(all_connectinos) - set(value)
            if unique_keys:
                need_updates.append((random.choice(list(unique_keys)), random.choice(value)))

        if not need_updates:
            return False
        else:
            return random.choice(need_updates)

    def check_network_integrity_helper(connections, data_entry):
        not_chacked = {}
        while not all(connections.values()):
            for nested_node_key, nested_node_value in connections.items():
                if not nested_node_value:
                    connections[nested_node_key] = True
                    for check_nested_node_key, check_nested_node_value in
data_entry['connections'][
                        nested_node_key].items():
                        if check_nested_node_key not in connections and check_nested_node_value ==
1 and check_nested_node_key not in not_chacked:
                            not_chacked[check_nested_node_key] = False
        return not_chacked

    def find_first_json_file(directory):
        global base_dir
        full_path = os.path.join(base_dir, directory)

        for file in os.listdir(full_path):
            if file.endswith('.json'):
                return os.path.join(full_path, file)
        return None

    # Example usage
    for i in range(20000):
        file_path = find_first_json_file('input')
        modified_data = randomize_data(file_path)

        # Forming the name of the output file
        output_file_name = new_time + '.json'
        write_json(modified_data, output_file_name, 'output')

        # Removing source file from input
        if os.path.exists(file_path):
            os.remove(file_path)
```

```
    # Adding a file to input for next iteration
    write_json(modified_data, output_file_name, 'input')
    print(f'The file {output_file_name} - has been created!')
```

Додаток В. Лістинг нейромережі

Розархівовуємо данні

[ ]

```
!unzip /content/dataset.zip -d /content/dataset
```

Якщо потрібно видалити датасет

[ ]

Завантаження та Агрегація Даних

[ ]

```
import os
import json
import numpy as np
import pandas as pd
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder, MinMaxScaler

# Function for reading data from JSON files
def load_data(directory, batch_size=1000):
    batch_data = []
    for filename in sorted(os.listdir(directory)):
        if filename.endswith('.json'):
            with open(os.path.join(directory, filename), 'r') as file:
                file_data = json.load(file)
                batch_data.append(file_data)
                if len(batch_data) == batch_size:
                    yield batch_data
                    batch_data = []
    if batch_data:
        yield batch_data
```

Попередня обробка даних

Кодування Категоріальних Ознак: 'node_ID', 'criticality_mark', 'process_type'

Нормалізація Числових Даних: 'bandwidth', 'connections_number', 'signal_delay'

[ ]

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from datetime import datetime

# Function for data conversion
def preprocess_data(batch):
    preprocessed = []
    for entry in batch:
```

```
            timestamp = datetime.strptime(entry['timestamp'], '%d-%m-%Y %H:%M:%S').timestamp()
            process_type = entry['process_type']
            for node in entry['nodes']:
                node_data = {
                    'timestamp': timestamp,
                    'process_type': process_type,
                    'node_ID': node['node_ID'],
                    'criticality_mark': node['criticality_mark'],
                    'bandwidth': node['metrics']['bandwidth'],
                    'connections_number': node['metrics']['connections_number'],
                    'signal_delay': node['metrics']['signal_delay']
                }
                # Додавання даних про звязки
                connections = entry['connections'][node['node_ID']]
                for conn_node, conn_value in connections.items():
                    node_data[f'conn_{conn_node}'] = conn_value
                preprocessed.append(node_data)
        return pd.DataFrame(preprocessed)
```

Створення Послідовностей: LSTM-мережі потребують послідовності даних. Потрібно буде визначити, як створити послідовності.

```
[ ]
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator

# Function to create sequences for LSTM
def create_sequences(df, target_column, sequence_length):
    X, y = [], []
    for i in range(len(df) - sequence_length):
        X.append(df.iloc[i:(i + sequence_length), :-1].values)
        y.append(df.iloc[i + sequence_length - 1][target_column])
    return np.array(X), np.array(y)
```

Обробка пакетів даних

```
[ ]
# Loading data
data_directory = '/content/dataset'
batch_size = 1000
dataset = load_data(data_directory, batch_size)

# Determining Sequence Length
sequence_length = 10

# Initialization for overlapping data
overlap_data = pd.DataFrame()

# Convert all data
all_X, all_y = [], []
for batch in dataset:
    processed_data = preprocess_data(batch)
```

```
            if not overlap_data.empty:
                processed_data.reset_index(drop=True, inplace=True)
                overlap_data.reset_index(drop=True, inplace=True)
                combined_df = pd.concat([overlap_data, processed_data], ignore_index=True)
            else:
                combined_df = processed_data

            # One-hot encoding for 'node_ID' and 'criticality_mark'
            one_hot_encoded_features = pd.get_dummies(combined_df[['node_ID', 'criticality_mark']], drop_first=True)
            combined_df = pd.concat([combined_df, one_hot_encoded_features], axis=1)

            # Coding of categorical features
            label_encoder = LabelEncoder()
            combined_df['process_type_encoded'] = label_encoder.fit_transform(combined_df['process_type'])

            # Видалення стовбців після кодування
            combined_df.drop(['node_ID', 'criticality_mark', 'process_type'], axis=1, inplace=True)

            # Normalizing Numeric Data
            scaler = MinMaxScaler()
            combined_df[['bandwidth', 'connections_number', 'signal_delay']] = scaler.fit_transform(
                combined_df[['bandwidth', 'connections_number', 'signal_delay']]
            )

            # Save latest data for overlap
            overlap_data = combined_df.iloc[-sequence_length:]

            print(Початок: combined_df')
            print(combined_df)
            print('Кінець: combined_df')
            print('Початок: overlap_data')
            print(overlap_data)

            # Create sequences for training
            X, y = create_sequences(combined_df, 'process_type_encoded', sequence_length)
            all_X.extend(X)
            all_y.extend(y)

        X_array = np.array(all_X)
        y_array = np.array(all_y)
        Поділ Даних на Навчальну та Тестову Вибірки
        [ ]
        from sklearn.model_selection import train_test_split

        # Splitting the dataset into training and test sets
        X_train, X_test, y_train, y_test = train_test_split(X_array, y_array, test_size=0.2, random_state=42)

        print(type(X_train), X_train.dtype, X_train.shape)
        print(type(y_train), y_train.dtype, y_train.shape)
```

```
print(type(X_test), X_test.dtype, X_test.shape)
print(type(y_test), y_test.dtype, y_test.shape)

Створення та Навчання LSTM Моделі
[ ]
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Визначення розміру вхідних данних
input_shape = (X_train.shape[1], X_train.shape[2])

# Створення моделі
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=input_shape, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(50, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))  # Только один выходной нейрон

# Компіляція моделі
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Навчання моделі
history = model.fit(X_train, y_train, epochs=50, batch_size=64, validation_data=(X_test, y_test), verbose=2)

Оцінка моделі
[ ]
# Evaluating model performance on test data
model.evaluate(X_test, y_test)
Використання Моделі для передбачень
[ ]
# Prediction based on new data
predictions = model.predict(X_test)

# Convert predictions to class labels
predicted_classes = np.argmax(predictions, axis=1)
```