МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ПРУС ІЛЛЯ ЛЕОНІДОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

# ВИЯВЛЕННЯ АНОМАЛЬНОЇ АКТИВНОСТІ У ЛОКАЛЬНІЙ МЕРЕЖІ ЗА ДОПОМОГОЮ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ

Спеціальність 122 Комп'ютерні науки
Кваліфікаційна (магістерська) робота

Науковий керівник:
Ю.С. Антонов, доцент кафедри
інформаційних технологій,
канд. фіз.-мат. наук, доцент
_____

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

Вінниця — 2024

# АНОТАЦІЯ

Пояснювальна записка: 73 сторінки, 12 рисунків, 55 джерел.

Об'єктом дослідження є аномалії у роботі локальної мережі.

Предметом дослідження є програмні засоби для проектування та реалізації системи виявлення аномалій у даних моніторингу локальної мережі.

Мета випускної кваліфікаційної роботи – розробка системи виявлення аномалій у даних моніторингу локальної мережі, використовуючи методи інтелектуального аналізу.

У випускній кваліфікаційній роботі розглянуто основні системи моніторингу та аналізу локальних мереж, визначено класифікацію засобів моніторингу та аналізу аномалій, розглянуто технології виявлення аномальної діяльності, проведено аналіз та вибір засобів реалізації поставленого завдання, налаштований моніторинг імітованої локальної мережі за допомогою системи zabbix для підготовки вихідних даних.

Розроблено систему виявлення аномалій з використанням методів інтелектуального аналізу, а саме ARIMA та LSTM.

Ключові слова: АНОМАЛІЇ, ЛОКАЛЬНА МЕРЕЖА, СИСТЕМИ МОНІТОРИНГУ, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ, ПРОГНОЗУВАННЯ, НЕЙРОНІ МЕРЕЖІ, МОВА ПРОГРАМУВАННЯ PYTHON.

# ABSTRACT

Explanatory note: 73 pages, 12 figures, 55 sources.

The object of the research is anomalies in the operation of the local network.

The subject of the research is software tools for designing and implementing a system for detecting anomalies in local network monitoring data

The goal of the final qualification work is to develop a system for detecting anomalies in local network monitoring data using methods of intellectual analysis.

In the final qualification work, the main systems of monitoring and analysis of local networks were considered, the classification of means of monitoring and analysis of anomalies was determined, technologies for detecting anomalous activity were considered, the analysis and selection of means of implementation of the task was carried out, monitoring of a simulated local network was configured using the zabbix system for the preparation of output data.

An anomaly detection system was developed using the methods of intellectual analysis, namely ARIMA and LSTM.

Keywords: ANOMALIES, LOCAL NETWORK, MONITORING SYSTEMS, INTELLIGENT ANALYSIS, FORECASTING, NEURAL NETWORKS, PYTHON PROGRAMMING LANGUAGE.

# Зміст

# ВСТУП

Одним із проявів процесу інформатизації суспільства є масштабний розвиток мережевих сервісів. У зв'язку з цим зростає необхідність такого завдання адміністрування інформаційних систем, як забезпечення надійності роботи системи. До цього відноситься своєчасна перевірка стану обладнання, підтримання оптимальної роботи програмного забезпечення, забезпечення безпеки даних. Це завдання вирішується у сферах комп'ютерної безпеки та системного адміністрування. Відчутний внесок у вирішення цих проблем роблять методи аналізу даних. Здійснюючи обробку службових даних про статистику, що надаються системами моніторингу, дані інструменти реалізують перевірку різних метрик роботи інформаційної системи на предмет досягнення аномальних значень.

Метою даної роботи є розробка системи виявлення аномалій у роботі локальної мережі. Для цього з поставлено такі завдання:

- Провести дослідження існуючих систем моніторингу та аналізу локальних мереж;
- Визначити класифікацію засобів моніторингу та аналізу аномалій;
- Розглянути технології виявлення аномальної діяльності;
- Провести аналіз та вибір засобів реалізації поставленого завдання;
- Обрати та описати алгоритми для виявлення аномалій за допомогою інтелектуального аналізу;
- Розробити програму, що реалізує виявлення аномалій в даних розробленої системи моніторингу.

**Об’єктом дослідження** є процеси моніторингу та аналізу діяльності локальних мереж.

**Предметом дослідження** є програмні засоби для проектування та реалізації системи виявлення аномалій у даних моніторингу локальної мережі.

**Методи дослідження**. У процесі роботи застосовувались

загальнонаукові та спеціальні методи, які дозволили вивчити предмет та об’єкт дослідження, дослідити напрями та шляхи оптимізації процесу проектування та розробки системи виявлення аномалій у даних моніторингу локальної мережі. Аналіз існуючих систем моніторингу локальних мереж, класифікація та аналіз засобів виявлення аномалій, дослідження та розробка методів інтелектуального аналізу даних, практична реалізація та тестування розробленої системи.

**Практичне значення одержаних результатів**. Практичне значення веб-сервісу визначається великим попитом на подібні системи, визначається великим попитом на подібні системи, оскільки завдання моніторингу та виявлення аномалій має безпосередній вплив на швидкість та якість роботи будь якої локальної мережі та допомагає вчасно виявляти та усувати аномальну активність всередині мережі.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ЩО РОЗГЛЯДАЄТЬСЯ

## 1.1 Дослідження існуючих систем моніторингу та аналізу мережевого трафіку

У процесі обробки та зберігання інформації неминуче виникає необхідність обміну даними між учасниками цього процесу. З кінця 70-х років почався бурхливий розвиток комп'ютерних мереж та супутнього мережевого обладнання. Локальні та глобальні мережі продовжують розвиватися, виникають нові протоколи передачі, розширюються апаратні можливості мережевого обладнання, зростає кількість підключених абонентів та сумарний обсяг трафіку.

Такий інтенсивний розвиток області спричиняє низку проблем. Одна з них полягає в тому, що при зростанні кількості споживачів інформаційних послуг збільшуються вимоги до мережного та серверного обладнання, яке використовується для підтримки належного рівня якості обслуговування. Друга ґрунтується на необхідності захисту інформації, що циркулює усередині мереж [6]. Для вирішення цих проблем використовують моніторинг та аналіз трафіку, що допомагають ефективно діагностувати та вирішувати проблеми при їх виникненні, не дозволяючи мережному обладнанню простоювати довго [1]. Так як інформація по мережі передається майже безперервно, стає зрозуміло, що припинення роботи обладнання або інші причини відмови в обслуговування призводять до збитків організацій чи компаній, що надають послуги. У зв'язку з цим адміністраторам необхідно стежити за рухом мережевого трафіку та продуктивністю всієї мережі, а також перевіряти її на вразливості у безпеці.

## 1.2 Класифікація засобів моніторингу та аналізу аномалій

Інструменти, що пропонують функціонал моніторингу та аналізу

обчислювальних мереж, можна розділити на кілька груп [12]:

*Системи управління мережею (Network Management Systems)* - централізовані програмні системи, які збирають дані про стан мережних пристроїв та інформацію про трафік у мережі. Функціонал даних програм не обмежений моніторингом та аналізом мережі.

Додатково в напівавтоматичному або автоматичному (залежно від реалізації) режимі, здійснюються дії з управління мережею: налаштування та зміна адресних таблиць комутаторів та іншого обладнання, увімкнення та відключення портів пристроїв. До систем у цій категорії відносяться HPOpenView, SunNetManager, IBMNetView.

*Вбудовані системи діагностики та управління (Embedded systems)* системи даного типу виконані у вигляді програмно-апаратних модулів, які встановлюються в комунікаційне обладнання, або операційну систему в якості програмних модулів. Вони дозволяють керувати та діагностувати лише той пристрій, на якому знаходяться.

Прикладом таких систем є модуль керування концентратором Distributed 5000, який виконує функції автосегментації портів після виявлення несправностей, приписування портів внутрішнім сегментам концентратора та деякі інші. Зазвичай вбудовані модулі управління також виконують роль SNMP-агентів, передаючи дані про стан пристрої у систему управління.[49]

*Засоби керування системою (System Management)* - інструменти з цієї групи виконують функції, аналогічні функціям систем управління, але стосовно інших об'єктів. У першому випадку об'єктом управління є програмне та апаратне забезпечення комп'ютерів мережі, а у другому – комунікаційне обладнання. При цьому частина функцій цих двох видів систем можуть дублюватися (наприклад, засоби управління системою можуть проводити найпростіший аналіз трафіку).[50]

*Аналізатори протоколів (Protocol analyzers)* – це програмні або апаратно-програмні системи, що використовуються тільки для моніторингу та

аналізу трафіку у мережах. Хорошим аналізатором вважається той, що може захоплювати та декодувати пакети великої кількості протоколів, застосовуваних у мережах - приблизно кілька десятків. Ця група систем може встановлювати деякі логічні умови для захоплення окремих пакетів і виконувати повне декодування пакетів, тобто відображати в зручній для користувача формі вкладеність пакетів протоколів різних рівнів з розшифруванням змісту кожного поля пакета.

Коли починають проектувати чи модернізувати мережу, часто виникає потреба у кількісному вимірі характеристик мережі: наприклад затримки, що виникають на різних етапах, частота виникнення вибіркових подій, інтенсивність потоків даних по лініях зв'язку, час реакції на запити. [51]

*Обладнання для діагностики та сертифікації кабельних систем* – з назви стає ясно призначення цієї групи. Умовно можна виділити чотири підтипи подібного обладнання: кабельні сканери, мережеві монітори, мультиметри та прилади для сертифікації кабельних систем. Це обладнання призначене для визначення стану та якості кабельних мереж.

Ця група обладнання включає чотири підтипи, кожен з яких відповідає конкретним завданням:

Кабельні сканери використовуються для виявлення і локалізації дефектів у кабельних системах. Вони здатні визначати місце переривань чи коротких замикань у кабелях, що допомагає в усуненні проблем та забезпеченні надійної передачі сигналів.

Мережеві монітори призначені для стеження за обсягами трафіку, пропускною здатністю та іншими параметрами мережі. Це дозволяє виявляти та аналізувати аномалії в роботі мережі, а також визначати причини зниження її продуктивності.

Мультиметри використовуються для вимірювання різних характеристик сигналів, таких як напруга, опір, індуктивність та інші. У контексті кабельних систем, мультиметри допомагають визначити електричні параметри кабелів,

забезпечуючи їхню відповідність стандартам.

Прилади для сертифікації кабельних систем використовується для оцінки та підтвердження відповідності кабельних систем визначеним стандартам. Вони проводять складні вимірювання, такі як витривалість передачі сигналу, довжина кабелю та інші параметри, що є важливими для правильної експлуатації мереж.[52]

*Експертні системи* поєднують людські знання про виявлення причин аномальної роботи мережі та можливих способах повернення мережі в робочий стан. Найчастіше вони представлені у вигляді окремих підсистем інших засобів моніторингу та аналізу мереж, розглянутих раніше.

До простого варіанту експертної системи відноситься контекстно - залежна help-система, а більш складні є так звані бази знань, які мають елементи штучного інтелекту. Прикладом цієї групи є система, вбудована у систему управління Spectrum компанії Cabletron.[53]

*Багатофункціональні пристрої аналізу та діагностики* – через широкого поширення локальних мереж з'явилася потреба в розробці недорогих портативних приладів з функціоналом кількох пристроїв: кабельних сканерів, програм мережного управління та аналізаторів протоколів. Як приклад можна навести Compas компанії MicrotestInc або 675 LANMeter компанії FlukeCorp.[54]

Також варто відзначити ще два способи моніторингу мережі. Перший це маршрутизаторо-орієнтований. Він є способом моніторингу, вбудованим безпосередньо в маршрутизатор і не вимагає додаткового встановлення іншого забезпечення. Другий спосіб, відповідно – це не орієнтований на маршрутизатори, тобто це підібране самим фахівцем необхідне апаратне та програмне забезпечення для поточних потреб.

### 1.3 Системи виявлення аномалій та запобігання вторгненням

Аномалії у контексті кібербезпеки та моніторингу систем

характеризуються як відхилення в діяльності або поведінці системи від відомих ознак, що характеризують коректну або припустиму поведінку об'єкта спостереження. Тобто, аномальна поведінка виявляється, коли дії об'єкта не відповідають звичним або очікуваним моделям, і ці дії можуть суперечити безпековій політиці компанії або встановленим нормам. Ці відхилення можуть виявлятися у різних параметрах. Коли дії об'єкта відхиляються від встановлених параметрів профілю, система виявлення аномалій генерує сигнал про початок аномалії. Ці сигнали можуть вказувати на потенційні загрози безпеці, такі як несанкціонований доступ, збій системи або шкідливу діяльність.

Системи виявлення аномалій та запобігання вторгненням є важливими засобами для захисту інформаційних систем і мережевих інфраструктур. Ці системи використовуються для виявлення та запобігання неавторизованому доступу або мережевим атакам, які можуть загрожувати безпеці інформації.

Система виявлення вторгнень - це програмний або апаратний засіб, призначений для виявлення фактів неавторизованого доступу (вторгнення або мережевої атаки) до комп'ютерної системи чи мережі.

Системи виявлення аномалій постійно моніторять мережевий трафік та активність системи, збираючи різноманітні дані, такі як мережеві пакети, записи системних журналів, аудити транзакцій та інші. Зібрані дані аналізуються на предмет відхилень від встановлених норм або шаблонів поведінки. Аномалії можуть вказувати на потенційні безпекові порушення, такі як мережеві атаки, віруси, шпигунське або шкідливе програмне забезпечення.

Система запобігання вторгненням - це розширення систем виявлення вторгнень. Система також здійснює моніторинг мережі чи системи, але, на відміну від систем виявлення вторгнень, вона активно втручається для запобігання або блокування виявленої шкідливої активності в реальному часі. На відміну від системи виявлення вторгнень, які лише повідомляють про

потенційні загрози, система запобігання вторгненням активно втручається, здійснюючи дії для запобігання або блокування шкідливої активності.

Впровадження подібних систем захисту інформації є необхідністю для всіх серйозних мережевих інфраструктур, оскільки існують програми, які постійно вишукують вразливості у будь-якому обладнанні, підключеному до глобальної мережі. Наприклад, пошуковий двигун Shodan [17] в автоматичному режимі збирає інформацію про підключені пристрої, які не мають будь-якої частини системи безпеки. Користувачі Shodan знаходять системи керування, які не мають реквізитів доступу, або налаштовані за замовчуванням. Отже, до них можна легко проникнути і зменшити працездатність.

Проти такого впливу і спрямовані системи виявлення та запобіганню вторгнень, тому вони часто використовуються в якості інструменту у політиці безпеки [2].

*Система виявлення вторгнень (англ. Intrusion Detection) System (IDS))* – програмний або апаратний засіб, призначений для виявлення фактів неавторизованого доступу (вторгнення або мережевої атаки) у комп'ютерну систему чи мережу.

*Система запобігання вторгненням (англ. Intrusion Prevention) System (IPS))* – програмний або апаратний засіб, що здійснює моніторинг мережі або системи в реальному часі з метою виявлення, запобігання або блокування аномальноїактивності.

Системи запобігання вторгненням можна вважати розширенням систем виявлення вторгнень, оскільки завдання відстеження атак залишається аналогічним. Але СПВ має відстежувати вторгнення в реальному часі та відразу здійснювати дії щодо запобігання атакам. Для цього вони використовують: скидання з'єднань, блокування потоків трафіку в мережі, видачу сигналів оператору. Крім цього, такі системи можуть дефрагментувати пакети, змінювати порядок TCP пакетів для захисту від пакетів із зміненими

SEQ та ACK номерами тощо [3].

Дані системи використовуються для автоматизації процесу контролю над подіями, що протікають у комп'ютерній системі чи мережі, та аналізу цих подій із метою пошуку ознак проблем безпеки. Так як кількість різних способів та видів організації несанкціонованих вторгнень в мережі за останній час значно збільшилася, то системи виявлення вторгнень стали обов'язковою частиною інфраструктури безпеки для більшості організацій. Цьому сприяють як велика кількість літератури з цього питання, яку потенційні зловмисники уважно вивчають, так і все більш витончені підходи до виявлення спроб проникнення до інформаційних систем.

Сучасні системи виявлення вторгнень мають різну архітектуру, основними з яких є: мережева та локальна. Мережеві системи встановлюють на виділених для цього комп'ютерах так, щоб вони могли аналізувати трафік, що протікає по локальній обчислювальній мережі. Локальні системи розміщуються на тих комп'ютерах, які потребують захисту, та вивчають певні події (програмні виклики або дії користувача).

Крім архітектури системи виявлення вторгнень також можуть розрізняти за методикою виявлення: частина систем шукає аномальну поведінку, інша – зловмисну [7].

### 1.3.1 Методики виявлення аномальної та зловмисної поведінки користувачів

*Системи виявлення аномальної поведінки (від англ. Anomaly detection)* засновані на тому, що системи виявлення вторгнень відомі ознаки, що характеризують коректну чи припустиму поведінку об'єкта спостереження. Під «коректною» або «правильною» поведінкою розуміються дії, виконувані об'єктом і які не суперечать безпековій політиці компанії[3].

*Системи виявлення зловмисної поведінки* (misuse detection) засновані на тому, що наперед відомі ознаки, що характеризують поведінку зловмисника.

Найбільш поширеною реалізацією технології виявлення зловмисної поведінки є системи: Snort, RealSecure IDS, Enterasys Advanced Dragon IDS).

Розглянемо докладніше технології, що використовуються у даних системах (Рисунок 1.1) [11].


Рисунок 1.1 – Існуючі технології система виявлення вторгнень

### 1.3.2 Технології виявлення аномальної діяльності

Датчики-сенсори аномалій ідентифікують незвичайну поведінку, так звані аномалії у функціонуванні окремого об'єкта. Тому головна складність у застосуванні їх на практиці пов'язана з нестабільністю самих об'єктів, що захищаються, а також і взаємодіючих з ними зовнішніх об'єктів. Як об'єкт спостереження може виступати мережа в цілому, окремий комп'ютер, мережна служба (наприклад, файловий сервер FTP), користувач і так далі. Датчики спрацьовують за умови, що події відрізняються від «звичайної» (коректної) діяльності. Тут варто відзначити, що в різних реалізаціях є різні визначення допустимого відхилення для спостерігаємої поведінки від дозволеної і своє визначення для «порога спрацьовування» сенсора спостереження.

Заходи та методи, які зазвичай використовуються у виявленні аномалій, включають наступні [11]:

- порогові значення: спостереження за об'єктом виражаються у вигляді числових інтервалів. Вихід за межі цих інтервалів вважається аномальною поведінкою. Параметри за якими можна спостерігати: кількість файлів, до яких звертається користувач у даний період часу, кількість невдалих спроб входу в систему центрального процесора тощо. Пороги можуть бути статичними та динамічними (тобто змінюватися, підлаштовуючись під конкретну систему);
- параметричні: для виявлення атак будується спеціальний "профіль нормальної системи" на основі шаблонів (тобто деякої політики, якої зазвичай має дотримуватись даний об'єкт);
- непараметричні: профіль будується на основі спостереження за об'єктом у період навчання;
- статистичні заходи: рішення про наявність аномалії робиться за великою кількістю зібраних даних шляхом їхньої статистичної передобробки;
- заходи на основі правил (сигнатур): дуже схожі на непараметричні статистичні заходи. У період навчання складається уявлення про нормальну поведінку об'єкта, що записується у вигляді спеціальних "правил". На виході маємо сигнатури «хорошої» поведінки об'єкта;
- Інші методи включають нейронні мережі та генетичні алгоритми, які дозволяють класифікувати певний набір видимих ознак сенсорів.

У сучасних системах виявлення аномалій переважно використовують перші два методи. Слід зазначити, що існують дві крайності при використанні даної технології:

- виявлення аномальної поведінки, яка не є атакою, та віднесення її до класу атак (помилка другого роду);
- пропуск атаки, яка не підпадає під визначення аномальної поведінки (помилка першого роду). Цей випадок набагато небезпечніший, ніж хибне зарахування аномальної поведінки до класу атак.

Тому при встановленні та експлуатації систем такої категорії звичайні

користувачі та фахівці стикаються з двома досить нетривіальними завданнями:

- визначення граничних значень характеристик поведінки суб'єкта для зниження ймовірності появи одного з двох вищеописаних крайніх випадків;
- побудова профілю об'єкта – складне та витратне за часом завдання, яке вимагає від фахівця безпеки великої попередньої роботи, високої кваліфікації та досвіду.

Як правило, системи виявлення аномальної активності використовують журнали реєстрації та поточну діяльність користувача як джерела даних для аналізу. До переваг систем виявлення атак на основі технології виявлення аномальної поведінки можна віднести те, що вони:

- не потребують оновлення сигнатур та правил виявлення атак;
- здатні виявляти нові типи атак, сигнатури для яких ще не розроблено;
- генерують інформацію, яка може бути використана в системах виявлення зловмисної поведінки

Недоліками цих систем є такі фактори:

- генерують багато помилок другого роду;
- вимагають тривалого та якісного навчання;
- зазвичай надто повільні в роботі та вимагають великої кількості обчислювальних ресурсів.

### 1.3.3 Статистичний аналіз аномальної поведінки

Застосування методів статистичного аналізу є найбільш поширеним видом реалізації технології виявлення аномальної поведінки у мережі. Статистичні датчики збирають різну інформацію про типову поведінку об'єкту та формують її у вигляді профілю. Профіль в даному випадку – це набір параметрів, що характеризують типову поведінку об'єкту. Він формується на основі статистики об'єкта, що спостерігається, з застосуванням методів

математичної статистики (наприклад, методу ковзних вікон та методу зважених сум).

Спочатку проходить період початкового формування профілю, після чого дії об'єкта порівнюються з відповідними параметрами та при виявленні суттєвих відхилень подається сигнал про початок аномалії.

Параметри профілю можна систематизувати за поширеними групами:

- категоріальні параметри (імена файлів, команди користувача, відкриті порти тощо);
- числові параметри (кількість переданих даних за різними протоколами, завантаження центрального процесора, кількість файлів, до яких здійснювався доступ тощо);
- що не вписуються в класифікацію нарівні з попередніми типами параметрів.

Також, щоб більш повно описувати поведінку об'єкта, що змінюється, профілі мають механізми динамічної зміни. Системи, що застосовують статистичні методи мають цілу низку переваг:

- не вимагають постійного оновлення бази сигнатур атак (це значно полегшує завдання супроводу цих систем);
- можуть адаптуватися до зміни поведінки користувача і тому є чутливішими до спроб вторгнення, ніж люди;
- можуть виявляти невідомі атаки, сигнатури для яких ще не написані і, отже, бути своєрідним стримуючим буфером до тих пір, поки не буде розроблено відповідний шаблон для експертних систем;
- дозволяють виявляти складніші атаки, ніж інші методи, наприклад, розподілені у часі чи об'єктах нападу.

Серед недоліків систем виявлення вторгнень можна визначити наступні фактори:

- у статистичних методах ймовірність отримання хибних повідомлень про аномалію є набагато вищою, ніж при використанні інших методів;

- труднощі завдання порогового значення (вибір цих значень – дуже нетривіальне завдання, яке потребує глибоких знань контрольованої системи);
- статистичні методи не дуже коректно обробляють зміни у діяльності користувача (наприклад, коли менеджер виконує обов'язки підлеглого у критичній ситуації). Цей недолік може представляти велику проблему в організаціях, де зміни є частими. В результаті можуть з'явитися як помилкові повідомлення про небезпеку, так і негативні хибні повідомлення (пропущені атаки);
- система може сприймати діяльність, що відповідає атаці, в якість нормальної через свою адаптацію до нової поведінки, якщо зміни режиму роботи були поступовими;
- статистичні методи не здатні виявити атаки з боку суб'єктів, котрим неможливо описати шаблон типового поведінки;
- статистичні методи мають бути попередньо налаштовані (задані порогові значення для кожного параметра, для кожного користувача);
- системи, побудовані виключно на статистичних методах, не справляються з виявленням атак з боку суб'єктів, які з самого початку виконують несанкціоновані дії, оскільки шаблон звичайної поведінки для них включатиме лише аномалії;
- статистичні методи на основі профілю нечутливі до порядку прямування подій.

Тим не менш, існують шляхи вирішення цих проблем, та їх практична реалізація є лише питанням часу. Очевидно, що статистичний метод є чистою реалізацією технології виявлення аномального поведінки. Статистичний метод успадковує у технології виявлення аномалій всі необхідні на практиці переваги [9].

## 1.4 Аналіз недоліків сучасних систем виявлення аномалій

З урахуванням сказаного вище, всі системи виявлення аномалій можна розділити на системи, орієнтовані на пошук:

- сигнатур всіх відомих аномалій;
- аномалій взаємодії контрольованих об'єктів;
- спотворення еталонної профільної інформації.

В даний час майже відсутні системи гібридного типу, а також системи, що використовують інформацію розподіленого в часі та просторі характеру. У ході роботи переважної більшості сучасних систем використовується лише сигнатурний метод розпізнавання аномальних впливів або лише пошук аномалій у поведінці контрольованої мережі.[19] Також у багатьох відомих систем відсутній імітатор атак або будь-який інший засіб для перевірки коректності розгорнутої та експлуатованої СОА, яка забезпечувала б простий та надійний засіб тестування конфігураційних параметрів, використаних у кожній конкретній комп'ютерній мережі.

Цей засіб, з логічних міркувань, повинен дозволяти імітувати діяльність програмного забезпечення вірусного типу (наприклад, CodeRed, NetSky, Bagle, MSBlast), атак на відмову в обслуговуванні (наприклад, SYN-шторм або атаку типу fraggle), атак з метою підвищення привілеїв облікового запису (як приклад, можна привести вразливості у мережевих службах MS SQL Server 2000, MS Internet Information Service 5.0), атаку з метою перенаправлення трафіку та нав'язування неправдивих даних (підміна ARP та нав'язування DNS служби). Бажано, щоб при цьому програмний засіб мав можливість генерувати атаки розподіленого характеру.

Наприклад, архітектура деяких типів імітаторів СОА складається з набору агентів різних типів, спеціалізованих для вирішення підзавдань виявлення вторгнень. Агенти розміщуються на окремих комп'ютерах системи. У цій архітектурі у явному вигляді відсутній «центр управління» сімейством агентів, тому що залежно від ситуації провідним може ставати будь-який з

агентів, який ініціює функції кооперації та управління. У разі потреби вони можуть скопіюватися в мережевому та локальному середовищі, або припинити своє функціонування. Залежно від ситуації (виду та кількості атак на комп'ютерні мережі, наявності обчислювальних ресурсів для виконання функцій захисту), може знадобитись генерація декількох екземплярів агентів кожного класу. Передбачається, що архітектура системи може адаптуватися до реконфігурації мережі, зміни трафіку та нових видів атак, використовуючи накопичений досвід.[56]

Багатоагентні системи є цікавою розробкою, але в українських джерелах немає вказівок на використовувані чи розроблені алгоритми виявлення аномалій схожого типу. А також поточні версії відомих імітаторів не функціонують у реальному режимі часу (оскільки цього не дозволяє робити обраний базовий інструментарій).

Загалом відсутність імітаторів атак для оцінки ефективності СОА не є основною проблемою цього напряму. Реальними недоліками існуючих систем виявлення аномалій є примітивність простого сигнатурного пошуку, низька ефективність при виявленні розподілених за часом та місцем складних атак, недостатня інтеграція інформації на рівні хоста та мережі для виявлення комбінованих атак та несанкціонованих проникнень.[25]

Як експлуатаційні недоліки можна відзначити велику кількість обчислювальних операцій для простого поділу власності події на «свій-чужий» і неможливість обробки всієї отриманої інформації в реальному режимі часу на звичайних персональних комп'ютерах, оскільки швидкість обробки мережного чи іншого трафіку подій найчастіше повільніше реального часу в 1.5-2 рази. Тому в деяких системах аналіз відбувається у відкладеному режимі. Це означає, що реалізація атаки на інформаційні та обчислювальні ресурси, що захищаються не буде помічена вчасно і тим більше не буде відображена за допомогою наявних засобів захисту. У цьому режимі засоби виявлення атак можуть бути використані у кращому випадку як засіб

журналювання всіх етапів атаки для подальшої експертизи.

Більшість сучасних СОА спочатку не розробляються для роботи на різних операційних системах і довільних апаратно-обчислювальних платформах. Тому робота на кількох операційних системах більшості продуктів є неможливою. Ці системи не використовують переваги розробки та оптимізації коду для обраних операційних систем та апаратних платформ, що є їх одним із найістотніших недоліків.[20]

Також у жодній програмній чи апаратно-програмній системі не передбачено режим «гарячої заміни», що дозволяє у разі виведення з ладу основного комплексу оперативно ввести в роботу комплекс резервування та відновити знищений рубіж оборони мережевого периметру.[26]

Незважаючи на це, є позитивний момент у розвитку систем виявлення аномалій - це прагнення розробників інтегрувати свої системи з існуючими засобами захисту (міжмережевими екранами, блокаторами каналів, QoS-диспетчерами).

## 1.5 Висновки за розділом

У першому розділі обґрунтовано актуальність проблеми аналізу мережевого трафіку для виявлення аномальної активності та розглянуті існуючі способи виріше02ння цієї проблеми. Проведено дослідження існуючих систем моніторингу та аналізу мережевого трафіку, наведено класифікацію засобів моніторингу та аналізу аномалій. Наведено опис систем виявлення аномалій та запобігання вторгненням та аналіз недоліків сучасних систем виявлення аномалій.

# РОЗДІЛ 2
# АНАЛІЗ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ ПОСТАВЛЕНОГО ЗАВДАННЯ

Метою даної магістерської роботи є розробка та впровадження системи виявлення аномалій у локальних мережах за допомогою інтелектуального аналізу. Дана система буде спрямована на надання допомоги при адмініструванні обчислювальних систем та мереж, забезпечуючи раннє виявлення потенційних загроз та аномалій в роботі мережі.

У сучасному світі, де інформаційні технології грають фундаментальну роль у функціонуванні більшості секторів, безпека та стабільність мережевих систем є критично важливими. Ця робота зосереджена на дослідженні аномалій у роботі локальних мереж, що є особливо актуальним у контексті зростаючих вимог до мережевої безпеки та надійності.

Аномалії в мережах можуть свідчити про різноманітні проблеми, від збоїв в обладнанні до кібератак. Тому виявлення та аналіз цих аномалій є ключовим для попередження втрат даних, забезпечення надійності мережевих сервісів та захисту від шкідливих дій. Розвиток ефективних методів для моніторингу та аналізу аномалій дозволить оперативно реагувати на потенційні загрози та підтримувати стабільну роботу мережевих систем.

Мережеві аномалії - це відхилення від нормального стану або поведінки мережі, які можуть вказувати на проблеми, такі як технічні збої, неправильні конфігурації або навіть мережеві атаки.

Ці аномалії можуть включати, але не обмежуються, несподіваними збоями в трафіку, незвичайними паттернами поведінки користувачів, або незвичайними шаблонами комунікації між вузлами в мережі.

Виявлення мережевих аномалій є важливим завданням у галузі забезпечення кібербезпеки. Це пов’язано з тим, що аномалії здебільшого є початковою стадією мережевих атак, що може спричиняти як негативні

нематеріальні наслідки, так і фінансові втрати для організацій

Аномалією у контексті моніторингу локальних мереж буде вважатися будь-яке значне відхилення в поведінці або діяльності системи від відомих патернів, що характеризують коректну або прийнятну поведінку. Це включає незвичні шаблони мережевого трафіку, показники продуктивності системи та діяльності користувачів, які не відповідають встановленим нормам і можуть вказувати на порушення безпеки, збої системи чи інші операційні проблеми.

Коректна або прийнятна поведінка в локальній мережі може включати, але не обмежуватися наступним:

Регулярний трафік: Обсяг даних, що передаються між вузлами в мережі, є стабільним і відповідає очікуваному рівню для даного часу доби або дня тижня.

Звичайні паттерни поведінки користувачів: Активність користувачів відповідає їх звичайним шаблонам поведінки. Наприклад, користувач, який зазвичай активний протягом робочого дня, продовжує демонструвати таку поведінку.

Звичайні шаблони комунікації між вузлами: Вузли в мережі взаємодіють один з одним відповідно до очікуваних шаблонів. Наприклад, сервер даних може регулярно обмінюватися даними з вузлами, які відомі як його звичайні клієнти.

Вибір цього напряму досліджень обумовлений також стрімким розвитком технологій у сфері обробки великих даних та машинного навчання, що відкриває нові можливості для виявлення та прогнозування аномалій. Використання цих сучасних технологій у магістерській роботі дозволить не лише вирішити актуальні задачі моніторингу мереж, але й сприятиме розвитку області комп'ютерних наук загалом.

Локальні мережі використовуються в різних сферах, включаючи бізнес, освіту, та домашній сектор. Збільшення кількості підключених пристроїв та різноманіття їхніх функцій створює серйозні виклики у забезпеченні безпеки

мереж. Виявлення аномальної активності є ключовим елементом для попередження інцидентів та забезпечення нормального функціонування.

Адміністратори локальних мереж стикаються з постійним тиском з боку кіберзлочинців та внутрішніх загроз. Виявлення аномалій у реальному часі стає необхідністю для оперативного реагування на потенційні загрози та усунення їх наслідків. Це також сприяє зменшенню трудовитрат на виявлення та аналіз аномальної активності вручну.[22]

З урахуванням регулярних змін в кіберпросторі та створення нових загроз, стандарти безпеки постійно змінюються. Розробка та впровадження систем виявлення аномалій є стратегічно важливими для підтримання та підвищення рівня безпеки локальних мереж у відповідь на постійно зростаючі виклики.

З поглибленням цифровізації та поширенням локальних мереж виникає необхідність вдосконалення систем безпеки для забезпечення стійкості та ефективності інформаційних систем. Ростуть загрози безпеці, пов'язані з аномальною активністю у мережах, що вимагає розробки нових підходів до виявлення та управління цими ризиками.

Заострена актуальність проблеми забезпечення безпеки локальних мереж визначає важливість вдосконалення існуючих систем виявлення аномалій. З ростом обсягу та різноманітності даних, що обробляються в мережах, із зростанням кількості пристроїв та користувачів, а також у зв'язку з поширенням розподілених та хмарних обчислень, виникають нові виклики у сфері кібербезпеки.

Цифрові трансформації організацій вносять не тільки позитивні зрушення, а й підвищують загрози від сучасних кібератак. Виявлення аномалій у локальних мережах є ключовим елементом стратегії кіберзахисту, оскільки дозволяє оперативно реагувати на потенційні загрози та мінімізувати можливі наслідки.

В даному контексті інтелектуальний аналіз даних набуває важливого

значення. Застосування алгоритмів машинного навчання та штучного інтелекту дозволяє ефективно аналізувати величезні обсяги інформації, виявляти неочікувані патерни та реагувати на нові форми кіберзагроз. Інтелектуальний аналіз даних стає ключовим елементом для створення систем виявлення аномалій, які забезпечать надійний захист локальних мереж в умовах постійно змінюючогося кіберландшафту.

Основною ідеєю в даному контексті є створення інноваційних підходів до виявлення аномальної активності, які не тільки забезпечать високий рівень точності виявлення, але й зможуть адаптуватися до нових форм загроз та змінюваних умов мережевого середовища.

Розглядаючи специфіку локальних мереж, важливо враховувати їхню унікальність у контексті безпеки та стабільності. Локальні мережі часто піддаються внутрішнім та зовнішнім загрозам, що вимагає розробки спеціалізованих методів виявлення аномалій. Ця робота не тільки вивчає існуючі підходи до моніторингу та аналізу аномалій у локальних мережах, але й пропонує власні методики для покращення точності та ефективності процесів виявлення та відповіді на аномальні події. Зокрема, застосування алгоритмів машинного навчання, таких як ARIMA та LSTM, демонструє високий потенціал у прогнозуванні та швидкому реагуванні на аномалії, що відкриває нові перспективи для підвищення безпеки локальних мереж.

Очікувані результати даної роботи передбачають створення системи виявлення аномалій, яка забезпечує високу точність у виявленні аномалій з мінімізацією хибнопозитивних сигналів, що дозволить знизити непотрібні спрацьовування та підвищити довіру до системи. Швидкість реагування на загрози має бути оптимізована для забезпечення оперативного втручання у випадку виявлення аномалій, що сприятиме підвищенню загальної безпеки мережі. Система буде адаптована до нових типів атак, що забезпечить її ефективність навіть у разі зміни характеру загроз, а її інтеграція з існуючими мережевими інфраструктурами дозволить ширше застосування без потреби в

значних модифікаціях існуючих систем.

## 2.1. Побудова комп'ютерно-математичної моделі

У цій роботі ми розробляємо модель для виявлення аномалій у локальних мережах. Модель інтегрує статистичні методи та алгоритми машинного навчання, включаючи ARIMA та LSTM, щоб підвищити точність та зменшити помилкові спрацьовування. Особлива увага приділяється валідації та тестуванню моделі у різних мережевих умовах, щоб забезпечити її ефективність та адаптивність до реальних сценаріїв використання.

Очікується, що розроблена система значно підвищить точність виявлення аномалій у мережі зменшить кількість хибнопозитивних випадків та сприятиме швидшому реагуванню на потенційні загрози. Буде розроблена система яка вирізняється високою точністю ідентифікації дійсних аномалій, мінімізацією хибнопозитивних сигналів, забезпеченням швидкої відповіді на загрози, гнучкістю щодо адаптації до нових і змінених типів атак, та ефективною інтеграцією з наявними мережевими рішеннями та безпековими системами.

## 2.2. Збір Даних за Допомогою Wireshark

Ключовим інструментом збору даних є Wireshark. Це високоефективний інструмент аналізу мереж, що забезпечує збір та аналіз мережевого трафіку.

Wireshark, як передовий аналізатор трафіку, надає змогу перехоплювати та докладно відображати мережеві пакети, що передаються через мережу. Це дозволяє отримати важливі дані про поведінку мережі, включаючи обсяги відправлених та отриманих даних, протоколи, порти та інші ключові метрики. Така інформація є критичною для розуміння звичайної та незвичайної активності в мережі та для подальшого виявлення аномалій.

У рамках цієї роботи, Wireshark використовувався для збору реальних мережевих даних, які були піддані подальшому аналізу з використанням комп'ютерно-математичних моделей, таких як ARIMA та LSTM. Цей процес не тільки допомагає ідентифікувати потенційні мережеві аномалії, але й надає важливі відомості про загальний стан та продуктивність мережевої інфраструктури.[27]

Використання Wireshark у цій роботі є прикладом інтеграції передових технологій мережевого моніторингу з сучасними методами аналізу даних для створення глибокого та всебічного погляду на безпеку та ефективність мережевих систем.

Налаштування Wireshark для збору мережевих даних включає декілька важливих технічних кроків. Ініціюючи процес збору даних, спочатку було вибрано відповідний мережевий інтерфейс - локальний інтерфейс, через який протікає цікавий нам мережевий трафік.

Важливим етапом налаштування Wireshark було встановлення фільтрів. Це дозволяло зосередитись на конкретних типах трафіку, наприклад, HTTP або FTP, або на трафіку з певних IP-адрес. Таке налаштування було критично важливим для фокусування на релевантних даних та виключення інформації, що не мала значення для дослідження.

Під час збору даних, Wireshark надавав можливість записувати перехоплений трафік для його подальшого аналізу. Було збережено дані сесій як для негайного аналізу, так і для використання у майбутньому. Крім того, розширені функції Wireshark, такі як декодування протоколів і відстеження потоків, дозволяли глибоко аналізувати структуру та вміст пакетів, забезпечуючи детальний огляд мережевої активності.

Під час перехоплення даних активно моніториться трафік, що проходить через інтерфейс, записуючи всі пакети даних протягом визначеного часу. Це дозволило зібрати значний обсяг даних, що включав інформацію про розмір пакетів, джерела, призначення, протоколи та інші характеристики. Отримані

дані надавали детальне уявлення про активність у мережі, що було важливо для ідентифікації потенційних джерел аномалій.

По завершенню сесії перехоплення, зібрані дані були експортовані для подальшого детального аналізу. Перед тим як подати ці дані в аналітичні моделі, було виконано їх очищення та перетворення, включаючи видалення неповних або пошкоджених пакетів і нормалізацію розмірів пакетів.[28]

Для збору даних був використаний інструмент PyShark, який є Python обгорткою для Wireshark. Ось покроковий опис процесу:

1. Встановлення та Налаштування PyShark

Першим кроком було встановлення PyShark на комп'ютер. PyShark дозволяє використовувати функціонал Wireshark безпосередньо з Python, що надає більшу гнучкість для аналізу даних.

2. Вибір Мережевого Інтерфейсу для Моніторингу

Важливим кроком було визначення, через який мережевий інтерфейс слід проводити перехоплення. В даному випадку, був обраний основний мережевий адаптер комп'ютера (eth0).

3. Запуск Сесії Перехоплення Трафіку

Слідуючий код був використаний для ініціювання перехоплення трафіку:

```
import pyshark

def capture_traffic(interface='eth0', duration=60):
    capture = pyshark.LiveCapture(interface=interface)
    capture.sniff(timeout=duration)
    return capture
```

Цей фрагмент коду встановлює сесію перехоплення на вказаному інтерфейсі (eth0) і тривалість (60 секунд).

4. Обробка Зібраних Даних

Після збору даних було використано наступну функцію для обробки отриманих даних:

```
import pandas as pd

def process_data(capture):
    rows = []
    for packet in capture:
        packet_info = {
            'length': len(packet),
            'timestamp': packet.sniff_time,
            'source_ip': packet.ip.src if 'IP' in packet else None,
            'destination_ip': packet.ip.dst if 'IP' in packet else None,
            'protocol': packet.transport_layer if packet.transport_layer else None
        }
        rows.append(packet_info)
    return pd.DataFrame(rows)
```

Цей код ефективно витягує ключову інформацію з кожного перехопленого пакета, включаючи його довжину, часову мітку, IP-адреси джерела і призначення, а також протокол. Зібрані дані були збережені у вигляді DataFrame для подальшого аналізу.

5. Збереження та Аналіз Даних

Зібрані дані були збережені для подальшого аналізу. Використання DataFrame дозволило зручно маніпулювати та аналізувати дані, використовуючи різні можливості Python та бібліотеки Pandas.

Wireshark забезпечує детальний вигляд кожного мережевого пакета, включаючи інформацію про джерело, призначення, розмір, протокол і часову мітку. Така глибина інформації критично важлива для точного аналізу та виявлення аномалій.[23]

Однією з ключових переваг Wireshark є його функціонал фільтрації,

який дозволяє відфільтровувати нерелевантний трафік та зосередитися на специфічних типах даних. Це особливо корисно при аналізі великих обсягів даних. Можливість збору даних у реальному часі та збереження цих даних для подальшого аналізу надає Wireshark гнучкості як для негайного відстеження, так і для довготривалого моніторингу мережі.[32]

## 2.3 Вибір алгоритмів визначення аномалій

Для аналізу аномалій у локальних мережах використовується декілька ключових типів даних, кожен з яких надає унікальну перспективу на мережеву активність та потенційні безпекові ризики:

Мережеві пакети: Ці дані є фундаментальними для моніторингу мережевої активності. Вони містять інформацію про кожен пакет, що передається через мережу, включаючи його джерело, призначення, розмір, часові мітки та інші параметри. Аналіз мережевих пакетів дозволяє ідентифікувати незвичайні шаблони трафіку, такі як раптове збільшення об'єму даних або нетипові маршрути передачі даних, що можуть вказувати на атаки типу DoS (відмова в обслуговуванні), сканування портів та інші мережеві загрози.

Записи системних журналів: Ці журнали включають в себе детальні записи про різноманітні події в системі, такі як помилки, попередження, системні повідомлення та інші важливі події. Аналіз цих журналів може виявити неправильні конфігурації, збої в системі та спроби несанкціонованого доступу. Наприклад, незвичайно висока кількість помилок авторизації може свідчити про спроби злому паролів.

Аудити транзакцій: Дані аудиту відображають інформацію про всі транзакції або дії, які відбуваються у мережі. Вони можуть включати записи про зміни в конфігураціях системи, доступ до файлів та папок, виконання

програм та команд. Аналіз цих даних допомагає виявляти несанкціоновані або підозрілі дії, такі як незаконне внесення змін у системні налаштування або спроби доступу до конфіденційної інформації.[24]

Комбінування та аналіз цих різноманітних даних дозволяє системі виявлення аномалій комплексно оцінювати стан мережі та виявляти потенційні загрози.

На основі аналізу вищезазначених даних, виявлення аномалій у локальних мережах вимагає застосування високоефективних алгоритмів прогнозування, які можуть точно ідентифікувати незвичайні шаблони та поведінку. В цьому контексті, для розв'язання поставленої задачі було обрано два основних алгоритми: модель ARIMA та нейромережеву модель LSTM, які на основі інтелектуального аналізу навчальної послідовності обчислюють елементи послідовності у наступних відрізках часу. Якщо значення, які спрогнозувала програма, розходяться з поданими на вхід значеннями на задане відхилення, то тут відзначається аномалія типу викиду. Розглянемо більш детально ці алгоритми.

### 2.3.1 Модель ARIMA

Для виявлення аномалій у тимчасових рядах обрано використання моделі ARIMA [4] – авторегресійне інтегроване ковзне середнє, яке узагальнює авторегресійне ковзне середнє для випадку нестаціонарних часових рядів.

Модель ARIMA складається з трьох основних компонентів: AR (авторегресійна частина), I (інтегрована частина) та MA (частина ковзного середнього). Модель позначається як ARIMA(p, d, q), де:

p - порядок авторегресійної частини.

d - ступінь диференціювання.

q - порядок частини ковзного середнього.

AR (p) - Авторегресійна частина: Описується рівнянням

$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + \epsilon_t$ ,

де $Y_t$ - поточне значення, ф - коефіцієнти авторегресії, а $\epsilon_t$ - випадкова помилка.

I (d) - Інтегрована частина: Включає диференціювання серії *d разів для* досягнення стаціонарності. Наприклад, одноразове диференціювання:

$$\Delta Y_t - Y_t - Y_{t-1}$$

MA (q) - Ковзне середнє: Моделює поточне значення як лінійну комбінацію попередніх значень білого шуму:

$$Y_t - \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q}$$

де $\mu$ - середнє значення, $\theta_1$ - коефіцієнти ковзного середнього, $\epsilon_t$ - випадкова помилка.

В основі ARIMA лежать такі моделі, як авторегресійна модель і модель ковзного середнього.[45] В авторегресійній моделі значення часового ряду представляється лінійною комбінацією значень ряду в попередні моменти часу за формулою (3):

$$AR(p): X_t = c + \sum_{i=1}^{p} a_i X_{t-i} + \varepsilon_t \quad (3)$$

У цій формулі $X_t$, $X_{t-i}$ − значення часового ряду у моменти часу t, t - і відповідно, $a_1 \ldots a_p$ – параметри моделі, $c$ – постійна, $e_t$ – білий шум, тобто послідовність незалежних і однаково розподілених випадкових величин (як правило, нормальних), випадкова помилка виміру. Число $p$ називається порядком моделі.

Модель ковзного середнього показана у формулі (4):

$$MA(q): X_t = \mu + \sum_{i=1}^{q} b_i \varepsilon_{t-i} + \varepsilon_t \quad (4)$$

де $\mu$ – середнє значення ряду,

$b_1 \ldots b_q$ - параметри моделі,

$\varepsilon_t$ – білий шум,

$q$ – порядок моделі.

У даній моделі елемент тимчасового ряду в даний момент часу обчислюється як сума середнього, білого шуму в даний час і лінійна комбінація білого шуму у попередні моменти часу [11]. Комбінуючи *AR(p)* і *MA(q)*, ми можемо отримати авторегресійне ковзне середнє *ARMA(p, q)*, яке виражається формулою (5):

$$ARMA(p,q): X_t = c + \sum_{i=1}^{p} b_i \varepsilon_{t-i} + \varepsilon_t \quad (5)$$

Побудова моделі *ARMA(p, q)* передбачає стаціонарність тимчасового ряду. Для вирішення практичних завдань зазвичай використовується поняття слабко стаціонарного часового ряду. Слабко стаціонарним називається тимчасовий ряд у разі, якщо його теоретичні математичне очікування та дисперсія не залежать від часу, а коваріація між його значеннями в моменти *t* і *t*+ s залежить тільки від *s*, а не від часу [12]. На практиці показники роботи інформаційно-обчислювальних систем не завжди є стаціонарними.

Однак часовий ряд можна зробити стаціонарним, взявши різниці деякого порядку від вихідного часового ряду. Саме диференціювання часового ряду відрізняє модель *ARMA(p, q)* від *ARIMA(p, d, q)*, яка представляється формулою (6):

$$ARIMA(p,d,q): \Delta^d X_t = c + \sum_{i=1}^{p} a_i(\Delta^d X_{t-i}) + \sum_{i=1}^{q} b_i \varepsilon_{t-i} + \varepsilon_t \quad (6)$$

Якщо різниці $\Delta^d X_t$ тимчасового ряду $X_t$ стаціонарні, а різниці меншого порядку нестаціонарні, то $X_t$ називається інтегрованим рядом порядку $d$.

Для підбору параметрів моделі *ARIMA* $p$, $d$ та $q$ використовується інформаційний критерій, що можна представити формулою (7).

$$AIC = 2k - 2ln(\hat{L}) \quad (7)$$

Де $L$ – функція правдоподібності, $k$ – загальне параметрів моделі.

Найбільш підходять параметри $p$, $d$, $q$, у яких критерій *AIC* досягає найменшого значення, тобто невелика кількість параметрів $p$, $q$ та велика функція правдоподібності.

Для визначення моментів часу з аномаліями для кожного значення розраховується помилка прогнозу:

```
test_df['error'] = test_df['value'] - test_df['predicted']
```

Для помилки обчислюється ковзне середнє та ковзне середньоквадратичне відхилення на 12 попередніх значеннях:

```
df['err_meanval'] = df['error'].rolling(window=12).mean()
df['err_deviation'] = df['error'].rolling(window=12).std()
```

На основі ковзного середнього та середньоквадратичного відхилення будується довірчий інтервал у два середньоквадратичні відхилення:

```
df['-lim'] = df['err_meanval'] - (2 * df['err_deviation'])
df['+lim'] = df['err_meanval'] + (2 * df['err_deviation'])
```

Якщо значення помилки виходить за коректний інтервал, то в момент часу, коли це сталося, відзначається аномалія:

```
df['anomaly_points'] = np.where(((df['error'] > df['+lim']) | 
(df['error'] < df['-lim'])), df['value'], np.nan)
```

### 2.3.2 Модель алгоритму LSTM

LSTM - це розширення стандартної рекурентної нейронної мережі (RNN), яке включає в себе унікальну структуру блоків, здатних запам'ятовувати інформацію на довший термін. Основні елементи LSTM:

Ворота забування (Forget Gate): Використовують сигмоїдну функцію для визначення, яку інформацію потрібно видалити з клітинного стану. Рівняння:

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

Де $\sigma$ - сигмоїдна функція, $W_f$ , $b_f$ - ваги та зміщення, $h_{t-1}$ - попередній прихований стан, $x_t$ - вхід.

Ворота входу (Input Gate): Складається з двох частин - сигмоїди та гіперболічного тангенса. Сигмоїда визначає, які значення потрібно оновити, а гіперболічний тангенс створює нові кандидати на додавання в клітинний стан.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_f)$$

$$\tilde{C}_t = tanh(W_i \cdot [h_{t-1}, x_t] + b_C)$$

Оновлення клітинного стану: Комбінація вищезазначених воріт для оновлення стану.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Де $C_t$ поточний клітинний стан.

Ворота виходу (Output Gate): Визначають, яка частина клітинного стану буде наступним прихованим станом.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

Ці механізми дозволяють LSTM ефективно працювати з довготривалими залежностями, уникати проблеми зникнення та вибуху градієнтів, характерних для традиційних Recurrent Neural Network.

Для виявлення аномалій у часових рядах за допомогою інтелектуального аналізу також можливо використання методів прогнозування. Прогнозування за допомогою нейронних мереж передбачає роботу з вхідними даними, що представляють послідовність, тому важливо щоб нейронна мережа, обробляючи черговий показник у певний момент часу, вміла ґрунтуватися на значеннях у попередні моменти часу. Це дозволяють зробити рекурентні нейронні мережі (Recurrent Neural Network, RNN)[43] (Рисунок 3.1).

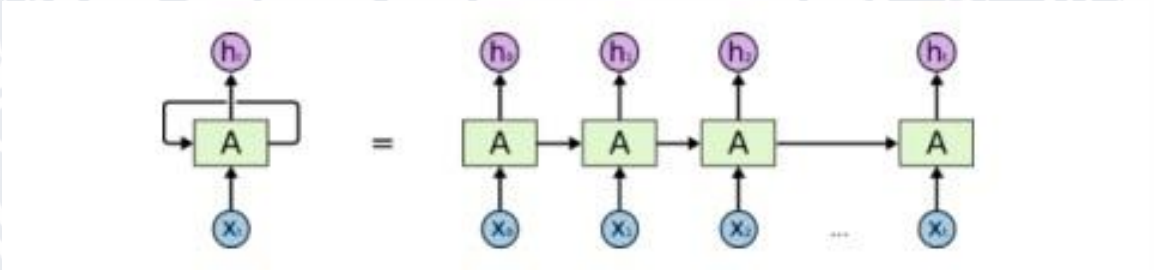


Рисунок 3.1 - Рекурентна нейронна мережа та її розгорнуте уявлення[44]

У такій мережі осередок отримує вхідну послідовність і обробляє її в циклі (Рисунок 3.2): на вхід подається елемент, та перетворене значення відправляється як на вихідний шар, так і на вхід з наступними елементами послідовності. На цьому починається нова ітерація циклу з урахуванням результату попереднього значення.

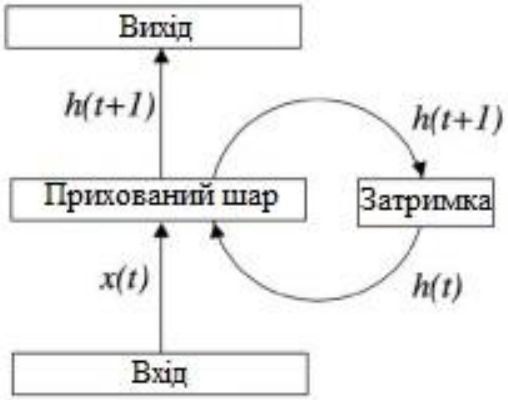


Рисунок 3.2 - Схема роботи прихованого шару

Однак у рекурентних мереж проблема в тому, що в міру отримання нових значень вплив попередніх значень слабшає. Цю проблему покликані вирішити мережі з довготривалою пам'яттю (Long short-term memory; LSTM) [16].

Довга короткострокова пам'ять – різновид рекурентної нейронної мережі, здатна до навчання довгострокових залежностей. Її особливістю є наявність у комірки стану, що регулюється так званими фільтрами (англ. Gate) (Рисунок 3.3).

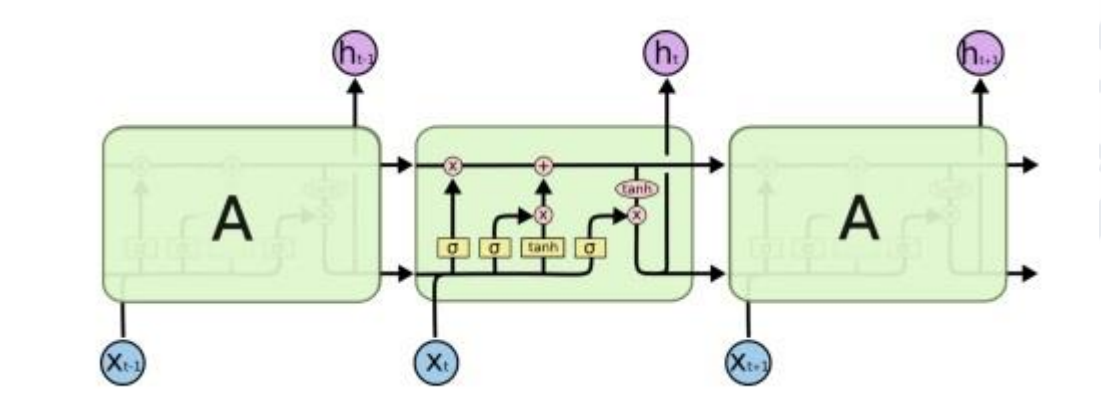


Рисунок 3.3 - Схема осередку LSTM

Фільтри являють собою сигмоїдальні шари. Вони обчислюють виважену

суму елементів конкатенації попереднього вихідного значення $h_{t-1}$ та поточного вхідного значення $x_t$ зі зсувом, після передаючи суму сигмоїдної функції активації.[46] Перший шар – шар фільтра забування – визначає ваги $f_t$ для кожного числа із вектору стану $C_{t-1}$ і виражається формулою:

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

Враховуючи попереднє вихідне значення $h_{t-1}$ та поточне вхідне значення $x_t$, він повертає вектор значень від 0 до 1 $f_t$, де 0 – «забути»

елемент стану, 1 - "зберегти".

Наступним кроком визначається шар вхідного фільтра, що визначає ваги $i_t$ для кожного числа з $C_t$ – значень-кандидатів, які можна додати до стану осередку. Вони обчислюються за формулами:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \sigma(W_c \cdot [h_{t-1}, x_t] + \tilde{b}_c)$$

Далі відбувається оновлення стану $C_{t-1}$ на $C_t$ за формулою:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Наприкінці відбувається генерація вихідних даних $h_t$ з поточного стану $C_t$ із застосуванням фільтра за формулою:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \cdot \tanh(C_t)$$

Для реалізації нейромережевої моделі LSTM було використано бібліотеку Keras, для якої визначення мережі у програмі виглядає наступним чином:

```
def fit_lstm(train, batch_size, epochs, neurons):
 X, y = train[:, :-1], train[:, -1:]
 X = X.reshape(X.shape[0], X.shape[1], 1)
    y = y.reshape(y.shape[0], y.shape[1], 1)
model = models.Sequential()
    model.add(layers.recurrent.LSTM(
          neurons,
```

```
batch_input_shape=(batch_size, X.shape[1], X.shape[2]),
stateful=True)
   )
   model.add(layers.Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
for i in range(epochs):
model.fit(
X, y, epochs=1,
batch_size=batch_size,
verbose="auto", shuffle=False
)
model.reset_states()
return model
```

У порівнянні з моделлю *ARIMA* мережа *LSTM* дозволяє передбачати значення нелінійними перетвореннями наявних значень часового ряду. Також за рахунок обчислення вектора стану у перетвореннях можуть брати участь не тільки найближчі попередні значення, а й значення за більш ранні моменти часу.

### 2.4 Інтеграція моделей та їх застосування

Комп'ютерно-математична модель, що застосовується для виявлення аномалій у локальних мережах базується на двох основних підходах: статистичному прогнозуванні за допомогою моделі ARIMA та використанні нейронних мереж, зокрема LSTM (Long Short-Term Memory). Обидві моделі інтегровані для забезпечення більш точного та надійного виявлення аномалій у локальних мережах. Використання ARIMA дозволяє ефективно аналізувати та прогнозувати лінійні тенденції у часових рядах, тоді як LSTM здійснює більш глибокий аналіз залежностей та шаблонів, які можуть бути неочевидними для ARIMA. Wireshark дозволив нам зібрати великий обсяг даних про мережевий трафік.

У нашому коді ми використовуємо Wireshark - потужний інструмент для аналізу мереж, а моделі машинного навчання були застосовані для ідентифікації потенційних аномалій. Обидві моделі (ARIMA та LSTM) використані для прогнозування майбутніх значень мережевих даних на основі навчальних даних. Потім ми порівнюємо ці прогнози зі спостереженими даними, щоб виявити аномалії. Якщо різниця між прогнозами та спостереженнями перевищує певний поріг (threshold), то це вказує на можливу аномалію.

```
import numpy as np
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator

np.random.seed(0)
simulated_packet_sizes = np.random.normal(1500, 250, 100)  # Середній розмір пакету 1500 байтів

train_data_series = pd.Series(simulated_packet_sizes)

model_arima = ARIMA(train_data_series, order=(5,1,0))
model_arima_fit = model_arima.fit(disp=0)

model_lstm = Sequential()
model_lstm.add(LSTM(50, activation='relu', input_shape=(3, 1)))
model_lstm.add(Dense(1))
model_lstm.compile(optimizer='adam', loss='mse')

n_input = 3
generator = TimeseriesGenerator(simulated_packet_sizes, simulated_packet_sizes, length=n_input, batch_size=1)
model_lstm.fit(generator, epochs=200, verbose=0)
```

```
test_data = np.random.normal(1500, 250, 10)

predictions_arima = model_arima_fit.forecast(steps=len(test_data))

test_data_reshaped = np.array(test_data).reshape((1, len(test_data), 1))
predictions_lstm = model_lstm.predict(test_data_reshaped, verbose=0).flatten()

threshold = 100
anomalies_arima = np.abs(test_data - predictions_arima) > threshold
anomalies_lstm = np.abs(test_data - predictions_lstm) > threshold

print("ARIMA anomalies:", anomalies_arima)
print("LSTM anomalies:", anomalies_lstm)
```

Цей підхід демонструє значний потенціал у використанні Wireshark разом із методами машинного навчання для виявлення аномалій у мережевих системах. Обидві моделі використовуються для прогнозування наступних значень в ваших даних. ARIMA робить це за допомогою статистичного аналізу, тоді як LSTM використовує нейронну мережу для вивчення шаблонів в даних. Після прогнозування, ми визначаємо аномалії як ті точки, де різниця між спрогнозованими та реальними значеннями перевищує певний поріг. Це допомагає нам виявити несподівані відхилення в даних. Поєднання підходів дозволяє моделі адекватно реагувати на складні зміни в поведінці мережі, виявляючи аномалії, які можуть бути ознаками безпекових порушень чи збоїв у системі.

Розроблена комп'ютерно-математична модель демонструє високу ефективність у виявленні аномалій у мережевих системах. Інтеграція ARIMA та LSTM забезпечує глибокий аналіз часових рядів, дозволяючи не тільки прогнозувати майбутній стан мережі, але й виявляти потенційні загрози на ранніх стадіях.

## 2.5 Результати та висновки

У цьому розділі було досліджено важливість систем виявлення аномалій у локальних мережах та зосереджено увагу на методах інтелектуального аналізу даних. Розглянуто специфіку локальних мереж, ключові типи аномалій та методи їх виявлення, включаючи аналіз мережевих пакетів та системних журналів.

Основні результати роботи включають розробку комп'ютерно-математичної моделі для точного виявлення аномалій, використовуючи алгоритми ARIMA та LSTM. Ви також використали Wireshark для збору даних, інтегрувавши статистичні та нейронні мережеві методи, що дозволило ефективно аналізувати та прогнозувати аномалії.

# РОЗДІЛ 3
# ВИЗНАЧЕННЯ АЛГОРИТМУ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

## 3.1 Програмна реалізація моделей виявлення аномальної активності у локальній мережі

Для підготовки вхідних даних був налаштований віртуальний сервер, імітуючий локальну мережу з системою моніторингу Zabbix, де збираються дані про роботу мережі, серед яких завантаження процесорів, обсяг оперативної пам'яті, швидкість мережевих інтерфейсів, навантаження жорстких дисків та ін.

API, що надається системою Zabbix, надає можливість надсилати запити для налаштування серверної частини системи та для отримання даних про конфігурацію Zabbix.[47] Також API дозволяє отримувати історію з того чи іншого показника роботи системи. Надсилання запитів реалізується з за допомогою виклику процедур у протоколі JSON-RPC. Запити та відповіді між клієнти та API закодовані з використанням формату JSON. Для використання цього API у Python будемо використувати бібліотеку pyzabbix.

Також було використано Wireshark для збору даних мережевого трафіку та їх подальшому аналізу з використанням алгоритмів машинного навчання. Мета полягає у зборі детальної інформації про мережеві пакети та їхньому подальшому аналізі за допомогою передових методів обробки даних та машинного навчання, зокрема, з використанням ARIMA та LSTM моделей.

### 3.1.1 Програмна реалізація моделі ARIMA

За допомогою API системи моніторингу Zabbix програма збирає показники локальної мережі за заданим ключем елемента даних у задані

проміжки часу і зберігає отримані значення файл типу csv. Розглянемо для прикладу показник активного часу роботи жорсткого диска у відсотках (Рисунок 3.4).

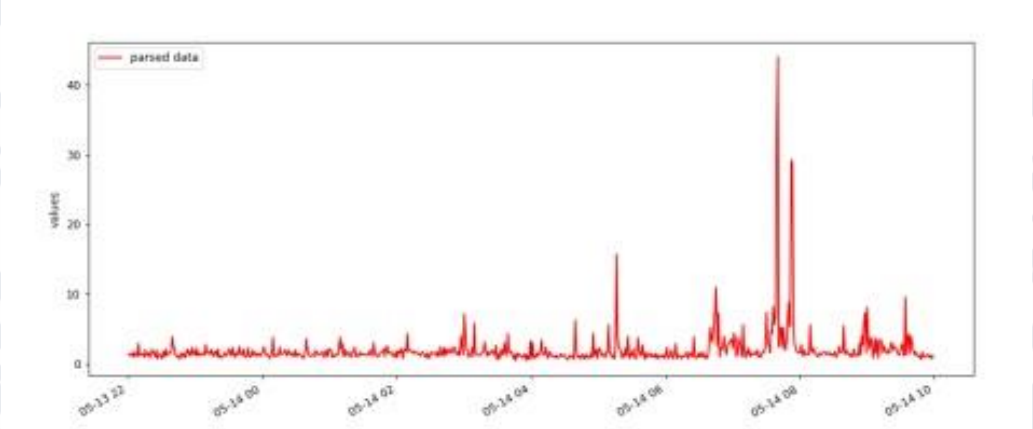


Рисунок 3.4 - Активний час роботи жорсткого диска у відсотках

Після цього з csv-файлу створюється таблиця Pandas.Dataframe, рядки якої поділяються на два відрізки часу: на навчальну та прогнозовану частину. На навчальній частині виконується підбір коефіцієнтів для *ARIMA*, які будуть використовуватися для прогнозування.

Для вибору параметрів *p, d, q* використовується функція *auto_arima*. З її допомогою методом перебору шукаються такі параметри, які дадуть найменше значення інформаційного критерію.

Після навчання моделі у циклі йде прогнозування пакетами по 10 значень, що задано у змінній *forecast_length*. Після чого передбачені значення додаються до моделі, щоб наступні передбачення обчислювати на їх основі, тобто здійснюється оновлення моделі.

Наприклад, передбачення моделі *ARIMA(4, 1, 4)* показано на рисунку (Рисунок 3.5). На даному графіку середньоквадратична помилка склала 3,084.

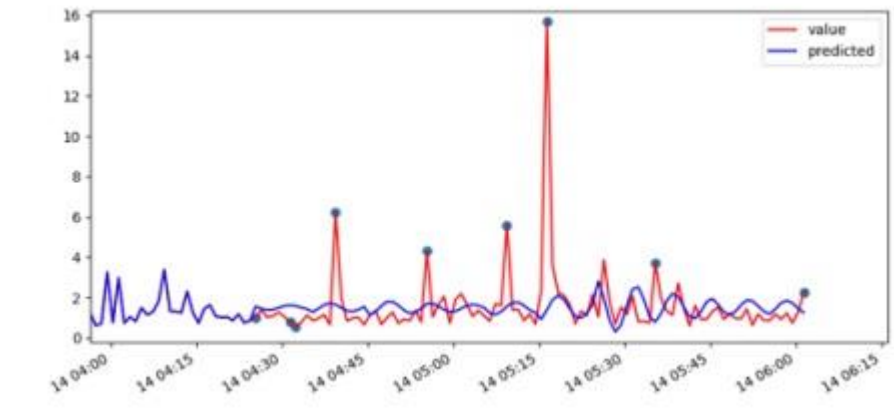


Рисунок 3.5 – Прогноз моделі

Застосування Wireshark для аналізу мережевого трафіку і використання моделі ARIMA виглядає таким чином: Спочатку Wireshark налаштовується на захоплення пакетів мережевого трафіку, які зберігаються у файл .pcap. Надалі, за допомогою tshark, ці дані експортуються у формат CSV. Далі, за допомогою Pandas, створюється DataFrame із цих даних, який поділяється на тренувальний та тестувальний набір для аналізу ARIMA. Використовуючи функцію auto_arima, визначаються оптимальні параметри моделі. Після тренування моделі здійснюється прогнозування на основі зібраних даних, оновлюючи модель для подальших прогнозів.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
import warnings

warnings.filterwarnings("ignore")

np.random.seed(0)
data = pd.DataFrame(np.random.randn(100), columns=['Value'])
```

```
train = data.iloc[:80]
test = data.iloc[80:]

model = ARIMA(train, order=(4,1,4))
model_fit = model.fit()

forecast = model_fit.forecast(steps=20)

plt.figure(figsize=(10, 6))
plt.plot(train.index, train['Value'], label='Train')
plt.plot(test.index, test['Value'], label='Test')
plt.plot(test.index, forecast, label='Forecast')
plt.title('ARIMA Model Forecast')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.show()
```

Дані були поділені на тренувальний (Train) та тестувальний (Test) набори. Для тренувального набору було використано модель ARIMA з параметрами (4, 1, 4), після чого було зроблено прогноз на 20 кроків вперед.

### 3.1.2 Програмна реалізація нейромережевої моделі

Аналогічно моделі ARIMA, програмна реалізація мережі LSTM працює з даними з файлу CSV, які містять зібрані дані з системи Zabbix.

Перед запуском навчання нейронної мережі виконується обробка тимчасового ряду.

На початковому етапі значень створюється масив Pandas.Series. Далі виконується диференціювання отриманого ряду, що у більшій кількості випадків дозволяє досягти стаціонарності (Рисунок 3.6).

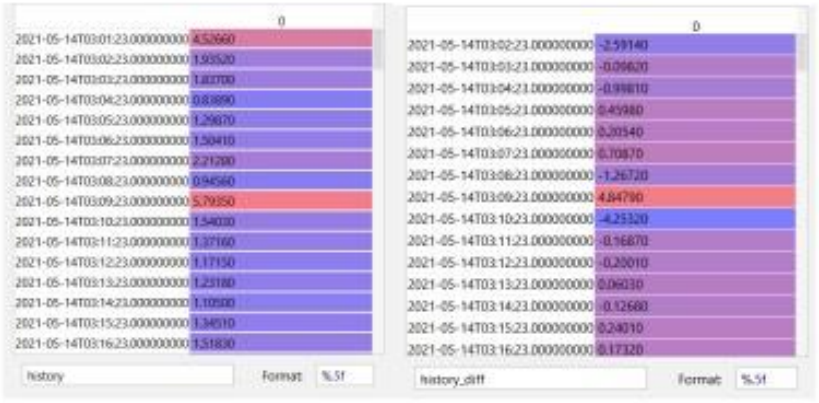

Рисунок 3.6 - Диференціювання часового ряду

Щоб дані були придатні для навчання, на основі масиву створюється матриця, у якої кожен рядок містить фрагмент тимчасового ряду довжиною 20, що визначається змінною *time_steps*. Кожен наступний рядок у цій матриці містить зсув на 1 значення щодо попереднього (Рисунок 3.7).

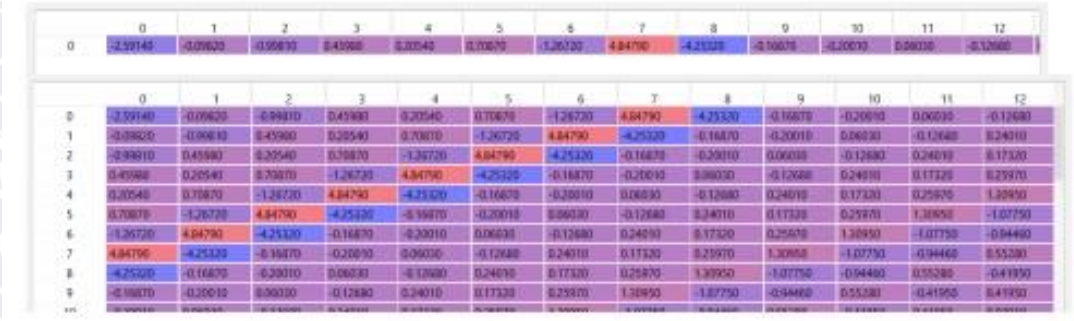

Рисунок 3.7 - Складання матриці з часового ряду

Наступним кроком проводиться нормалізація значень за допомогою функції *MinMaxScaler* пакета *scikit-learn* так, щоб вони перебували у проміжку [-1, 1].

При наборі даних у вигляді матриці зсувів нейронна мережа навчається передбачати останній стовпець матриці на основі попередніх стовпців.

Після здійснюється поділ даних на навчальну та тестову вибірки, де тестова вибірка складається з значень, число яких задано у змінній

*forecast_length*, наприклад 40. Це останні рядки матриці, у яких потрібно виконати передбачення останнього стовпця.

Як параметри були використані значення *time_steps* = 20, *batch_size* = 10, *epochs* = 10, *neurons* = 40. Параметри позначають таке:

- time_steps – довжина тимчасового відрізка, за яким передбачається таке значення;
- batch_size – розмір пакета вхідної послідовності, після обробки якого відбувається оновлення ваг;
- epochs – кількість епох нейронної мережі;
- neurons – число елементів у векторі стану, що у розгорнутому вигляді LSTM означало б число нейронів.

Мережа складається із шару LSTM, який зберігає стан для кожного наступного рядка із матриці. Наступний шар Dense – нейрон, який поєднує 40 значень (вихід шару LSTM) в одне, що вважається прогнозом.

Як функція втрат використовується середньоквадратична помилка, і для її мінімізації обрано оптимізатор Adam.

Так як модель зберігає стан від кожного рядка, то після кожного проходу по всій матриці (закінчення епохи) потрібно явно скидати стан. Тому навчання моделі виконується у циклі по одній епосі.

Оскільки вхідні дані утворюють послідовність, при навчанні ми забороняємо перемішування зразків – рядків матриці, поставивши параметр shuffle = False.

Після навчання моделі вона використовується для отримання останніх стовпців матриці на тестовій вибірці. Передбачення виконується по частинам, що мають число рядків, що дорівнює batch_size. Далі мережа навчається з новими значеннями. Після отримання всіх значень із тестової вибірки виконується зворотне перетворення з нормалізованих значень до диференційованого вигляду, а також від диференційованого до недиференційованого.

Наступним кроком починається пошук аномалій - тих значень, які найбільше відхиляються від передбачених. Їх визначення виконується аналогічно моделі ARIMA за допомогою ковзного середнього помилки прогнозування.

Результат роботи моделі представлений на рисунку (Рисунок 3.8). Середньоквадратична помилка на прогнозованому проміжку дорівнює 5,310.

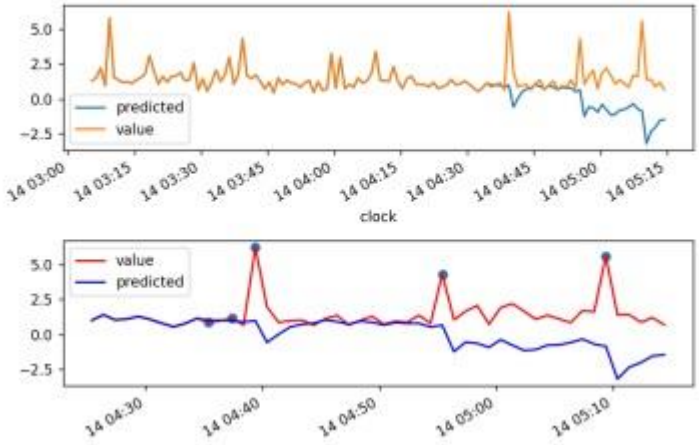


Рисунок 3.8 - Виявлення аномалій за допомогою LSTM

Модель LSTM здатна давати прогнози з більш великим відхиленням при великому обсязі вхідних даних, хоча при їхньому малому обсязі (90 хвилин історії на 40 прогнозів) обидві моделі правильно фіксують аномальні значення на однакових ділянках. Виходячи з цього модель ARIMA більш краща завдяки стабільності роботи та швидкодії (10 секунд навчання проти більше 60 у LSTM) при однаковому рівні якості визначення аномалій у мережі.

Використання Wireshark для захоплення мережевого трафіку та аналізу за допомогою моделі LSTM відрізняється від підходу з ARIMA. Wireshark налаштовується для збору мережевих даних, які потім експортуються у формат CSV. Ці дані використовуються для навчання нейронної мережі LSTM.

На початковому етапі, дані з CSV-файлу перетворюються у Pandas DataFrame. Для підготовки даних до LSTM, виконується їх нормалізація за

допомогою MinMaxScaler, таким чином масштабуючи їх до діапазону [-1, 1]. Далі створюється матриця, де кожен рядок містить відрізок даних довжиною, визначеною змінною time_steps, та зсув на одне значення відносно попереднього.

LSTM мережа конфігурується із параметрами: time_steps = 20, batch_size = 10, epochs = 10, neurons = 40. Мережа складається з LSTM шару, який передає стан для кожного наступного рядка, і Dense шару, який зводить вихідні значення LSTM у одне передбачене значення.

Процес навчання включає кілька епох без перемішування зразків. Після навчання моделі, вона використовується для прогнозування значень на тестовій вибірці, з подальшим оновленням моделі новими даними. Потім виконується пошук аномалій шляхом порівняння передбачених і фактичних значень.

### 3.2 Розробка графічного інтерфейсу для взаємодії

Для взаємодії з реалізованим функціоналом є необхідність реалізації графічного інтерфейсу користувача за допомогою інструментів Python. Для реалізації інтерфейсу програми була використана бібліотека Tkinter. Початковий екран інтерфейсу наведено на рисунку (Рисунок 3.9).

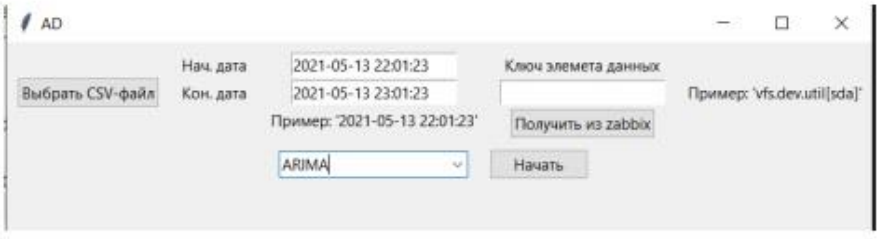


Рисунок 3.9 - Вигляд початкового екрану інтерфейсу

Користувач має можливість ввести інтервал дат для якого треба визначити наявність аномалій, дані за цей проміжок часу буде завантажено у

модель, після чого йому пропонується 2 способи вибору даних:

1) Вибрати з наявних csv-файлів. Для цього за кнопкою "Вибрати CSV" відкривається діалогове вікно вибору файлів.

2) Отримати дані по ключу із Zabbix за допомогою введення ключа елемента даних та натискання кнопки «Отримати з Zabbix». Ці дані також зберігаються локально в якості csv-файла.

У системі Zabbix кожен показник роботи локальної мережі, так званий «Елемент даних» має своє унікальне ім'я – ключ (наприклад, system.cpu.util[,idle] для часу простою процесора). Список ключів доступний у розділі «Останні дані» веб-інтерфейсу Zabbix (Рисунок 3.10).

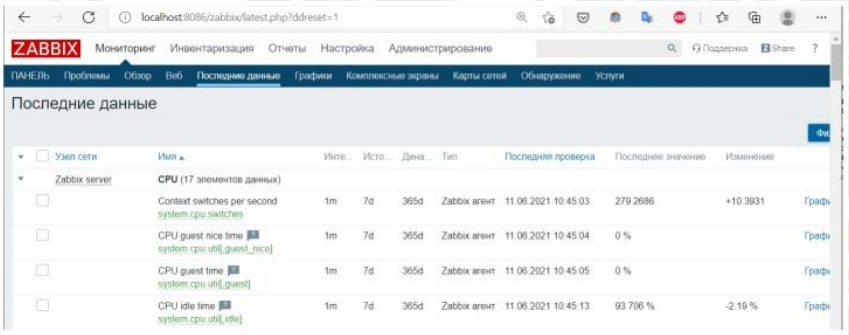


Рисунок 3.10 - Ключі метрик роботи локальної мережі

Після того, як було визначено тимчасовий ряд, користувач має можливість за допомогою спливаючого списку обрати модель прогнозування для виявлення аномалій з числа реалізованих раніше: ARIMA чи LSTM (Рисунок 3.11).


Рисунок 3.11 - Доступні моделі прогнозування

Після натискання кнопки «Почати» починається робота моделі на отриманих даних, і у програмі з'являється графік з результатом, де відзначено аномалії та проміжки часу (Рисунок 3.12).

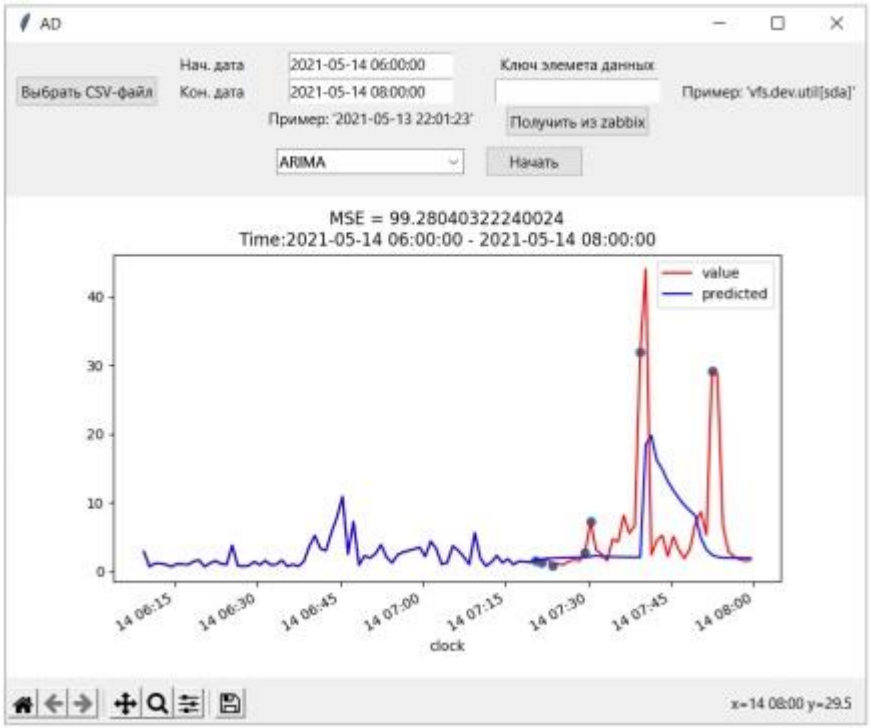


Рисунок 3.12 – Результат роботи реалізованої програми

### 3.3 Висновок до розділу

У даному розділі було проведено визначення алгоритму та практичну реалізацію програмного продукту. Проведено вибір алгоритмів визначення аномалій, які буде реалізовано, за резуальтатами обрано алгоритми ARIMA та LSTM. Виконано програмну реалізацію обраних моделей виявлення аномальної активності у локальній мережі за допомогою інструментів, що було обрано у 2 розділі. Розроблено графічний інтерфейс для взаємодії користувача із системою.

# ВИСНОВОК

Розробка системи виявлення аномальної активності у локальній мережі спрямована на надання допомоги при адмініструванні обчислювальних систем та мереж. Головним чином, виявлення аномальної поведінки дозволяє спростити аналіз навантаження на локальну мережу, зменшити терміни аудиту працездатності мережі та дослідження інцидентів, а також прискорити прийняття рішень щодо оптимізації її конфігурації.

У ході виконання випускної кваліфікаційної роботи були виконані такі завдання:

- Проведено дослідження існуючих систем моніторингу та аналізу локальних мереж;
- Визначено класифікація засобів моніторингу та аналізу аномалій;
- Розглянуто технології виявлення аномальної діяльності;
- Проведено аналіз та вибір засобів реалізації поставленого завдання;
- За допомогою Wireshark було здійснено збір даних про мережеві пакети, що дозволило точніше визначити аномалії та вдосконалити систему моніторингу.
- Налаштований моніторинг імітованої локальної мережі за допомогою системи Zabbix для підготовки вихідних даних;
- Для виявлення аномалій за допомогою інтелектуального аналізу побудовано моделі прогнозування тимчасових рядів авторегресійної інтегрованої ковзної середньої (ARIMA) та нейронної мережі з довгою короткостроковою пам'яттю (LSTM);
- Розроблено програму, що реалізує виявлення аномалій в даних розробленої системи моніторингу.

Таким чином, у цій роботі були виконані поставлені завдання і досягнуто мети: розроблено систему виявлення аномалій у даних моніторингу локальної мережі з використанням методів інтелектуального аналізу.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Система моніторингу мережі [Електронний ресурс] // IT база знань. - URL: https://wiki.merionet.com/servernye-resheniya/sistema-monitoringaseti/).

2. Успенський М. Б. Огляд підходів до виявлення збоїв у системах зберігання даних / М. Б. Успенський // Науково-технічні відомості Інформатики. Телекомунікації. Управління. 2019. Т. 12. № 4. с. 145-158.

3. Розгортання та налаштування Icinga 2 на Debian 8.6 [Електронний ресурс]// Блог IT-KB. – URL: https://blog.it-kb.com/2016/11/21/deploy-andconfigure-icinga-2-on-debian-8-6-part-1-installation-of-icinga-2-core-plugins-anddb-ido/ (дата звернення 18.10.2023).

4. Preetam J. Anomaly detection for monitoring: Статистичний приклад до time series anomaly detection / J. Preetam, B. Schwartz. - O'Reilly, 2015 - 75 с.

5. Кібер-оракул: пошук аномалій у даних моніторингу за допомогою нейромережі [Електронний ресурс] / / Блог компанії ITSumma / Хабр. - URL: https://habr.com/ua/company/itsumma/blog/341598/ (дата звернення 18.10.2023).

6. Виявлення аномалій у даних мережевого моніторингу методами статистики [Електронний ресурс] // Хабр - URL: https://habr.com/ua/post/344762/ (дата звернення 18.10.2023).

7. Гайфуліна Д. А., Котенко І. В. Аналіз моделей глибокого навчання задач виявлення мережевих аномалій інтернету речей // Інформаційно-керуючі системи, 2021 № 1, с. 28–37.

8. Szmit M., Szmit A. Use of Holt-Winters method in the analysis of network traffic: Case study / M. Szmit, A. Szmit // Communications in Computer and Information Science. - 2011. Vol. 160. - с. 224–231.

9. Соболєв К. В. Автоматичний пошук аномалій у тимчасових рядах [Електронний ресурс]. – URL: http://cs.mipt.com/wp/wp-content/uploads/2018/06/ Соболєв-К.В..pdf (дата звернення 18.10.2023).

10. Яковлєва А. В. Компоненти часового ряду – Економетрика [Електронний ресурс]. – URL: https://be5.biz/ekonomika/e008/70.html (дата звернення 18.10.2023).

11. Common Approaches to Univariate Time Series [Електронний ресурс]. – URL: https://www.itl.nist.gov/div898/handbook/pmc/section4/ pmc444.htm (дата звернення 18.10.2023).

12. Лутц М. Вивчаємо Python, 4-те видання. 2011. - 1280 с.

13. Златопольський Д.М. Основи програмування мовою Python. 2017. - 284 с.

14. Лутц М. Програмування на Python, тому I, 4-те видання. 2011. - 992 с.

15. Свейгарт, Ел. Автоматизація рутини задач за допомогою Python: практичний посібник для початківців. Пров. з англ. - М.: Вільямс, 2016. - 592 с.

16. Рейтц К., Шлюссер Т. Автостопом з Python. 2017. - 336 с.: іл. – (Серія "Бестселери O'Reilly").

17. Любанович Білл Простий Python. Сучасний стиль програмування. 2016. - 480 с.: - (Серія "Бестсеппери O'Reilly").

18. Федоров, Д. Ю. Програмування мовою високого рівня Python: навчальний посібник для прикладного бакалаврату [Електронний ресурс]. - URL: https://urait.com/bcode/437489

19. S. Axelsson. Intrusion Detection Systems: A Comprehensive Review [Електронний ресурс]. – URL: https://www.mdpi.com/1424-8220/21/3/748

20. T. Erl. Service-Oriented Architecture: Concepts, Technology, and Design . - 2005. – 792 с.

21. Dhruba K. Bhattacharyya, Jugal K. Kalita. Network Anomaly Detection: A Machine Learning Perspective. – 2014. – 366 с.

22. Бабій М.С. ЛОКАЛЬНІ МЕРЕЖІ ЕОМ [Електронний ресурс] URL: https://www.essuir.sumdu.edu.ua/bitstream-download/123456789/2443/1/m2150.pdf)

23. Шпаргалка Wireshark [Електронний ресурс] URL: https://hackyourmom.com/kibervijna/shpargalka-wireshark/

24. V. Chandola, A. Banerjee, V. Kumar. Anomaly detection: A survey. 2009. [Електронний ресурс] URL: https://www.researchgate.net/publication/220565847_Anomaly_Detection_A_Survey

25. M. Last, A. Kandel, H. Bunke. Data Mining in Time Series Databases. 2004. – 205 с.

26. R. Sommer, V. Paxson. Outside the Closed World: On Using Machine Learning For Network Intrusion Detection. 2010. – 305-316 с.

27. Пєтух А. М., Гончарук В. В. Способи аналізу мережевого трафіку комп'ютерної мережі. [Електронний ресурс]. – URL: https://inmad.vntu.edu.ua/portal/static/61B77AEA-39A2-4D2F-BB35-09541C7D9961.pdf

28. L. Chappell. Wireshark Network Analysis: The Official Wireshark Certified Network Analyst Study Guide. 2010. - 800 с.

29. Monowar H. Bhuyan , Dhruba K. Bhattacharyya , Jugal K. Kalita. Network Traffic Anomaly Detection and Prevention. 2017. – 263 с.

30. S. Raschka, V. Mirjalili. Python Machine Learning. 2017. – 622 с.

31. N. Venkataramanan, A. Shriram. Data Privacy. Principles and Practice. 2016. – 232 с.

32. Yoram Orzach , Nagendra Kumar Nainar , Yogesh Ramdoss. Network Analysis using Wireshark 2. 2018. - 626с.

33. J. Erickson. Hacking: The Art of Exploitation. 2004. – 264 с.thon Documentation. [Електронний ресурс] - URL: https://docs.python.org/

34. C. Stedman. data mining. [Електронний ресурс] - URL: https://www.techtarget.com/searchbusinessanalytics/definition/data-mining

35. Wes Bos. Sublime Text Power User. 2014. – 202 с.

36. Alessandro Del Sole. Visual Studio Code Distilled. 2021 – 260 с.

37. Monowar H. Bhuyan , Dhruba K. Bhattacharyya , Jugal K. Kalita. Network Traffic Anomaly Detection and Prevention. 2017. – 263 с.

38. L. Richardson, M. Amundsen. RESTful Web APIs. 2013. – 373 с.

39. Insomnia [Електронний ресурс] - URL: https://insomnia.rest/

40. Postman Quick Reference Guide [Електронний ресурс] - URL: https://postman-quick-reference-guide.readthedocs.io/en/latest/

41. C. Kankanamge. Web Services Testing with SoapUI. 2012. – 332 с.

42. Rest Assured [Електронний ресурс] - URL: https://rest-assured.io/

43. J. Brownlee. Deep Learning for Time Series Forecasting. 2018. – 575 с.

44. Recurrent Neural Networks [Електронний ресурс] - URL: http://talimi.se/dl/recurrent-neural-networks/

45. George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel and Greta M. Ljung. Time Series Analysis: Forecasting and Control. 2015. [Електронний ресурс] - URL: https://www.researchgate.net/publication/299459188_Time_Series_Analysis_Forecasting_and_Control5th_Edition_by_George_E_P_Box_Gwilym_M_Jenkins_Gregory_C_Reinsel_and_Greta_M_Ljung_2015_Published_by_John_Wiley_and_Sons_Inc_Hoboken_New_Jersey_pp_712_ISBN_

46. A Comprehensive Introduction to Different Types of Convolutions in Deep Learning [Електронний ресурс] - URL: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

47. R. Olups. Zabbix Network Monitoring. 2016. – 754 с.

48. M. Subramanian. Network Management: Concepts and Practice. 1999. – 644 с.

49. Arnold S. Berger. Embedded Systems Design: An Introduction to Processes, Tools, and Techniques. 2002. – 237 с.

50. Jeffrey O. Grady. System Management: Planning, Enterprise Identity, and Deployment. 2010. – 628 с.

51. C. Sanders. Practical Packet Analysis. 2011. – 240 с.

52. B. Woodwar. Cabling: The Complete Guide to Copper and Fiber-Optic Networking. 2014. – 532 с.

53. B. B. Gupta, D. P. Agrawal, Shingo Yamaguchi. Handbook of Research on Modern Cryptographic Solutions for Computer and Cyber Security. 2016. – 589 с.

54. C. Chapman. Network Performance and Security: Testing and Analyzing Using Open Source and Low-Cost Tools. 2016. – 380 с.

55. Y. Wang. Statistical Methods for Network Security: Modern Statistically-Based Intrusion Detection and Protection. 2009. – 476 с.

# ДОДАТКИ

## Додаток А

## Реалізація отримання даних за допомогою Wireshark

```
import pyshark
import pandas as pd
import datetime

def get_network_traffic_data(interface='eth0', duration=60,
csv_filename='network_traffic.csv'):

    capture = pyshark.LiveCapture(interface=interface)

    capture.sniff(timeout=duration)

    data = []
    for packet in capture:
        timestamp = packet.sniff_time
        try:
            src_addr = packet.ip.src
            dst_addr = packet.ip.dst
            length = packet.length
        except AttributeError:
            continue

        data.append([timestamp, src_addr, dst_addr, length])

    df = pd.DataFrame(data, columns=['timestamp', 'src_addr', 'dst_addr',
'length'])

    df.to_csv(csv_filename, index=False)
    return csv_filename

csv_file = get_network_traffic_data(interface='eth0', duration=60,
csv_filename='network_traffic.csv')
```

## Реалізація отримання даних із Zabbix

```
from pyzabbix import
ZabbixAPI import pandas as
pd import json import
datetime import time

def get_zabbix_csv(key_,
```

```
            datetime_from=(datetime.datetime.now() -
datetime.timedelta(hours=6)).strftime('%Y-%m-%d %H:%M:%S'),
datetime_till=datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')):
   ZBX_URL = 'http://127.0.0.1:8086/zabbix'
ZBX_USER = 'diplomawork'
   ZBX_PASSWORD = 'zabbix'
    zapi = ZabbixAPI(url=ZBX_URL, user=ZBX_USER,
password=ZBX_PASSWORD)    print("Connected to Zabbix API Version
%s" % zapi.api_version())
   # отримати id елемента даних
   item = zapi.item.get(filter={'host': 'Zabbix server', 'key_':
key_})    itemid = item[0]['itemid']    print(itemid)

   # Створити часовий проміжок
   time_from_dt = datetime.datetime.strptime(datetime_from, '%Y-%m-%d
%H:%M:%S')    time_till_dt = datetime.datetime.strptime(datetime_till, '%Y-
%m-%d %H:%M:%S')    time_from = int(time.mktime(time_from_dt.timetuple()))
time_till = int(time.mktime(time_till_dt.timetuple()))

   hist_req_params = {"history": 0, "output": "extend",
                "itemids": str(itemid),
                "time_from": str(time_from), "time_till": str(time_till)}
hist = zapi.do_request('history.get', hist_req_params)['result']

   time_now = datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
filename = r'pyzab\zabbixdata_%s_.csv' % time_now

   df = pd.read_json(json.dumps(hist))
   df.to_csv(filename, encoding='utf-8', index=False)
return filename
  def read_zabbix_csv(csv_path, time_from,
time_till):    # Читати з csv та привести до
необхідного вигляду
   df = pd.read_csv(csv_path)
    if df.columns[0] ==
"timestamp":
     df = df.rename(columns={df.columns[0]: "clock"})
     df["clock"] = pd.to_datetime(df["clock"], format='%Y-%m-%d %H:%M:%S')
elif df.columns[1] == "clock":
     df["clock"] = pd.to_datetime(df["clock"], unit='s')
   df.index = df["clock"]
tseries_parsed = df["value"]

   time_mask = (tseries_parsed.index >= time_from) & (tseries_parsed.index <=
time_till)

   return tseries_parsed.loc[time_mask]
```

## Реалізація розмітки аномалій

```
import numpy as np
import pandas as pd
  def detect_anomalies(df, window,
confid_interval=2):    df.fillna(0, inplace=True)
   df['error'] = df['value'] - df['predicted']
   df['error, %'] = (df['value'] - df['predicted']) / df['predicted'] * 100
df['err_meanval'] = df['error'].rolling(window=window).mean()
df['err_deviation'] = df['error'].rolling(window=window).std()
    df['-lim'] = df['err_meanval'] - (confid_interval * df['err_deviation'])
df['+lim'] = df['err_meanval'] + (confid_interval * df['err_deviation'])
df['anomaly_points'] = np.where(((df['error'] > df['+lim']) | (df['error'] <
df['-lim'])), df['value'], np.nan)

   return df
  def anom_plot(df, window, ax):    df.plot(ax=ax, kind='line',
use_index=True, y=['value', 'predicted'], color=['red', 'blue'])
   ax.scatter(df['anomaly_points'].index, df['anomaly_points'])
```

## Реалізація моделі виявлення аномалій ARIMA

```
from pmdarima import
auto_arima import numpy as
np import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import mean_squared_error
from math import sqrt
from warnings import filterwarnings
import time
 from zabbix_load import get_zabbix_csv,
read_zabbix_csv from mark_anomalies import
detect_anomalies, anom_plot


FORECAST_LENGTH = 10


def main_arima(csv_path, time_from='2023-10-17 22:01:23', time_till='2023-10-
17 23:10:23'):
```

```
   tseries_parsed = read_zabbix_csv(csv_path, time_from,
time_till)
   # train and prediction data
variables    train =
tseries_parsed[0:-40]    test =
tseries_parsed[-40:]

   print("loaded data count: %d \n" % (tseries_parsed.size))
print("train data count : %d \n" % (train.size))
print("test data count : %d \n" % (test.size))

   filterwarnings('ignore')

   X = train.tolist()
   model_arima = auto_arima(train, start_p=4, max_p=7, start_q=4, max_q=7,
trace=True)

   predictions = []
   for i in range(0, test.size, FORECAST_LENGTH):\
      predicted_values =
model_arima.predict(n_periods=FORECAST_LENGTH)      for val in
predicted_values:         predictions.append(val)
X.append(val)
     model_arima.update(test[i:i+FORECAST_LENGTH])

   predictions = np.array(predictions)[:test.size]

   test_df = test.to_frame()
   test_df.insert(test_df.shape[1], 'predicted', predictions)
    predictions_series = pd.Series(predictions,
index=test.index)    history_with_predicted = pd.concat([train,
predictions_series])    history_df = tseries_parsed.to_frame()
   history_df.insert(history_df.shape[1], 'predicted',
history_with_predicted)

   mse = mean_squared_error(test, predictions)
rmse = sqrt(mse)
    print("Test result: {}
".format(mse))    print("RMSE : %3.f
\n" % rmse)
    fig = plt.figure()
ax = fig.add_subplot()
    anom_df = detect_anomalies(history_df,
10, 2)    anom_plot(anom_df, 10, ax)

   ax.set_title("MSE = {}\nTime:{} - {}".format(mse, time_from, time_till))
return fig, ax
  if __name__ ==
"__main__":
start_time = time.time()
```

```
   fig, ax = main_arima('D:/pyzab/zabbixdata_2023-10-18_23-58-
30_.json.csv',
      '2023-10-17 02:00:00', '2023-10-17 08:00:00')

   stop_time = time.time()
   print("Time duration: {} sec.".format(stop_time - start_time))
plt.show()
```

## Реалізація моделі виявлення аномалій LSTM

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as
plt import time from math import
sqrt
 from sklearn.metrics import
mean_squared_error from
sklearn.preprocessing import MinMaxScaler
from keras import models, layers
 from zabbix_load import get_zabbix_csv,
read_zabbix_csv from mark_anomalies import
detect_anomalies, anom_plot


FORECAST_LENGTH = 40
TIME_STEPS = 20
BATCH_SIZE = 10
EPOCHS = 10
NEURONS = 40
  def timeseries_to_stacked(series, time_steps):
values = series.values    sequences = []    for i
in range(0, len(values) + 1 - time_steps):
sequences.append(values[i: (i + time_steps)])
return np.stack(sequences)
  def difference(tseries):
tseries_diff = tseries.diff(periods=1)
return tseries_diff[1:]
  def scale(train,
test=None):
   # пепетворити train (необхідна матриця)
    train = train.reshape(train.shape[0],
train.shape[1])
   # навчити нормалізатор
   scaler = MinMaxScaler(feature_range=(-1, 1))
   scaler = scaler.fit(train)
train_scaled = scaler.transform(train)
```

```
    if not (test is
None):      #
transform test
     test = test.reshape(test.shape[0], test.shape[1])
test_scaled = scaler.transform(test)

   return scaler, train_scaled,
test_scaled   def
invert_difference(history,
predicted_diff):    predicted_undiff =
pd.Series(predicted_diff)
   predicted_undiff[0] = history[-len(predicted_diff) - 1] +
predicted_diff[0]    for i in range(1, len(predicted_undiff)):
predicted_undiff[i] = predicted_undiff[i-1] + predicted_diff[i]
predicted_undiff.index = history.index[-len(predicted_undiff):]    return
predicted_undiff
  def invert_scale(scaler, X):
X = scaler.inverse_transform(X)
return X
  def fit_lstm(train, batch_size, epochs,
neurons):
   X, y = train[:, :-1], train[:, -1:]
X = X.reshape(X.shape[0], X.shape[1], 1)
y = y.reshape(y.shape[0], y.shape[1], 1)
    model = models.Sequential()
model.add(layers.recurrent.LSTM(
     neurons,
     batch_input_shape=(batch_size, X.shape[1], X.shape[2]),
stateful=True)
   )
   model.add(layers.Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
# plot_model(model, to_file='model_plot.png', show_shapes=True,
show_layer_names=True)    print(model.summary())    for i in
range(epochs):      model.fit(         X, y, epochs=1,
batch_size=batch_size,         verbose="auto", shuffle=False
     )
model.reset_states()

   return model
  def main_lstm(csv_path, time_from='2023-10-17 00:01:23', time_till='2023-
10-17
05:10:23'):
   global_start_time = time.time()

   history = read_zabbix_csv(csv_path, time_from, time_till)
history = history[len(history) % BATCH_SIZE:]

   history_diff = difference(history)
   stacked_history = timeseries_to_stacked(history_diff,
time_steps=TIME_STEPS)    train, test = stacked_history[:-FORECAST_LENGTH,
:], stacked_history[FORECAST_LENGTH:, :]
   scaler, train_scaled, test_scaled = scale(train, test)
```

```
    print("Model fit
started")    fit_start_time
= time.time()    model =
fit_lstm(train_scaled,
             batch_size=BATCH_SIZE, epochs=EPOCHS, neurons=NEURONS)
print("Model fit complete. Duration: {} sec.".format(time.time() -
fit_start_time))

   X_test = test_scaled[:BATCH_SIZE, :-1]
   X_test = X_test.reshape(X_test.shape[0], X_test.shape[1],
1)
   X_train = train_scaled[:, :-1]
   X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
   X_train = np.concatenate((X_train, X_test), axis=0)
    print("Prediction
started")    pred_start_time
= time.time()
   y = model.predict(X_test, batch_size=BATCH_SIZE)
   print("Prediction complete. Duration: {}
sec.".format(time.time() - pred_start_time))    predicted = y

   y_train = train_scaled[:, -1:]
   y_train = y_train.reshape(y_train.shape[0], 1)
y_train = np.concatenate((y_train, y), axis=0)

   # виконати прогноз та оновити модель

    for i in range(1, int(FORECAST_LENGTH /
BATCH_SIZE)):
      for _ in
range(EPOCHS):
model.fit(
          X_train, y_train, epochs=1,
batch_size=BATCH_SIZE,
verbose="auto", shuffle=False
        )
        model.reset_states()

     start = i * BATCH_SIZE

     X_test = test_scaled[start: start + BATCH_SIZE, :-1]
     X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

     y = model.predict(X_test, batch_size=BATCH_SIZE)
predicted = np.concatenate((predicted, y), axis=0)

     X_train = np.concatenate((X_train, X_test), axis=0)
y_train = np.concatenate((y_train, y), axis=0)

   test_scaled_with_predicted = np.concatenate((test_scaled[:, :-1],
predicted), axis=1)
```

```
   # invert_scale
   test_with_predicted = invert_scale(scaler, test_scaled_with_predicted)

   predicted_unscaled = test_with_predicted[:, -1]
   predicted_undiff = invert_difference(history, predicted_unscaled)

   print("Global time duration: {} sec.".format(time.time() -
global_start_time))
    def anom_detect(history_series,
predicted_series):      history_with_predicted =
pd.concat([history_series[:len(predicted_series)],
predicted_series])      history_df =
history_series.to_frame()
history_df.insert(history_df.shape[1],
'predicted', history_with_predicted)
     fig, axes = plt.subplots(nrows=2, ncols=1)
fig.tight_layout()
     history_df.plot(ax=axes[0], kind='line', y=['predicted',
'value'])
     ROLL_WINDOW = 10
     anom_df = history_df[-FORECAST_LENGTH - ROLL_WINDOW:].copy()
anom_df = detect_anomalies(anom_df, ROLL_WINDOW, 2)
anom_plot(anom_df, ROLL_WINDOW, axes[1])
     # plt.show()
return fig, axes

   fig, axes = anom_detect(history, predicted_undiff)

   mse = mean_squared_error(
     history[-
len(predicted_undiff):].values,
predicted_undiff.values)    rmse =
sqrt(mse)
    print("Test result: {}
".format(mse))    print("RMSE : %3.f
\n" % rmse)

   axes[0].set_title("MSE = {}\nTime:{} - {}".format(mse, time_from,
time_till))    return fig, axes
  if __name__ == "__main__":    fig, axes =
main_lstm('D:/pyzab/zabbixdata_2023-10-18_23-58-30_.json.csv',
              '2023-10-17 03:40:00', '2023-10-17 05:40:00')    plt.show()
```

**Реалізація інтерфейсу взаємодії з програмою**

```
import tkinter as tk from
tkinter import filedialog
from tkinter import ttk

import matplotlib

matplotlib.use("TkAgg")
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2Tk

from lstm import main_lstm
from arima import main_arima, get_zabbix_csv
  def btn_getzabbix_click():    filename.set(get_zabbix_csv(zbx_key.get(),
time_from.get(), time_till.get()))    state_message.set(filename.get())
 def btn_openfile_click():
filename.set(tk.filedialog.askopenfilename(
     initialdir="pyzab", filetypes=[("CSV file", "*.csv"), ("All files",
"*.*")]
))
   state_message.set(filename.get())
  def btn_startdetect_click():
state_message.set(cbx_methods.current())
fig = matplotlib.figure.Figure()
    if cbx_methods.current() == 0:      fig, ax =
main_arima(filename.get(), time_from.get(), time_till.get())
ax.plot()
    if cbx_methods.current() == 1:      fig, axes =
main_lstm(filename.get(), time_from.get(), time_till.get())      for ax in
axes:         ax.plot()
    for widget in
frame_plot.winfo_children():
     widget.destroy()

   canvas = FigureCanvasTkAgg(fig, frame_plot)
   canvas.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)

   toolbar = NavigationToolbar2Tk(canvas, frame_plot)
toolbar.update()
   canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH,
expand=True)
window = tk.Tk()
window.title('AnomDetect')

# ------ Настройка -------
 frame_setting =
tk.Frame(window)
frame_setting.pack(pady=10)

filename = tk.StringVar()

btn_openfile = ttk.Button(frame_setting, text='Выбрать CSV-файл',
command=btn_openfile_click)
```

```
btn_openfile.grid(row=1, column=0, padx=10)

time_from = tk.StringVar(frame_setting, value="2023-10-17 22:01:23")
ent_time_from = tk.Entry(frame_setting, textvariable=time_from)
ent_time_from.grid(row=0, column=2, padx=10)
 time_till = tk.StringVar(frame_setting, value="2023-10-17
23:01:23") ent_time_till = tk.Entry(frame_setting,
textvariable=time_till) ent_time_till.grid(row=1, column=2,
padx=10)

zbx_key = tk.StringVar()
ent_zbx_key = tk.Entry(frame_setting, textvariable=zbx_key)
ent_zbx_key.grid(row=1, column=4, padx=10)

btn_getzabbix = ttk.Button(frame_setting, text='Получить из zabbix',
command=btn_getzabbix_click)
btn_getzabbix.grid(row=2, column=4, padx=10)

cbx_methods = ttk.Combobox(frame_setting, values=["ARIMA", "LSTM"])
cbx_methods.grid(row=3, column=2, columnspan=2)

btn_startdetect = ttk.Button(frame_setting, text='Начать',
command=btn_startdetect_click)
btn_startdetect.grid(row=3, column=4, pady=10, sticky=tk.W)

ttk.Label(frame_setting, text='Нач. дата').grid(row=0, column=1, padx=10,
sticky=tk.E)
ttk.Label(frame_setting, text='Кон. дата').grid(row=1, column=1, padx=10,
sticky=tk.E)
ttk.Label(frame_setting, text="Пример: '2021-05-13 22:01:23'").grid(row=2,
column=2, padx=10, sticky=tk.N)
ttk.Label(frame_setting, text="Ключ элемента данных").grid(row=0, column=4,
padx=10)
ttk.Label(frame_setting, text="Пример: 'vfs.dev.util[sda]'").grid(row=1,
column=5, padx=10, sticky=tk.W)

# ------ Графік -------
frame_plot = tk.Frame(window, width=600, height=400)
frame_plot.pack(fill=tk.BOTH, expand=True)

# ------ Строка стану -------
frame_state = tk.Frame(window)
frame_state.pack(side=tk.BOTTOM, fill=tk.X)

state_message = tk.StringVar()
lbl_messages = tk.Label(frame_state, textvariable=state_message)
lbl_messages.pack(side=tk.LEFT)

window.mainloop()
```