Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

факультет інформаційних технологій
(факультет)

Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра
(бакалавра, спеціаліста, магістра)

студента Скубицького Олексія Сергійовича
(ПІБ)

академічної групи 123м-24-1
(шифр)

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

за освітньо-професійною програмою «Комп'ютерна інженерія»
(за наявності)

на тему Дослідження можливості застосування LSTM-прогнозування для адаптивного управління мережевими потоками в SDN-середовищі
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	доц. Бешта Д.О.			
розділів:				

Рецензент				

Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2025

ЗАТВЕРДЖЕНО:

завідувач кафедри
_____інформаційних технологій _____
_____та комп'ютерної інженерії_____
(повна назва)
_____ _Гнатушенко В.В._
(підпис) (прізвище, ініціали)

«_____»_____2025 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня ____магістр _____
(бакалавра, спеціаліста, магістра)

студенту_____Скубицькому О.С._____ **академічної групи** ____123м-24-1___
(прізвище та ініціали) (шифр)
спеціальності _____123 «Комп'ютерна інженерія»_____
за освітньою-професійною програмою ____«Комп'ютерна інженерія»
(за наявності)

на тему ___ Дослідження можливості застосування LSTM-прогнозування для адаптивного управління мережевими потоками в SDN-середовищі_____,

затверджену наказом ректора НТУ «Дніпровська політехніка» від 13.10.2025 р. № 1165/с

Розділ	Зміст	Термін виконання
Стан питання та постановка завдання	На основі матеріалів практик, інших науково-технічних джерел сформульоване наукове завдання, конкретизовані предмет та мета досліджень	10.10.2025
Теоретичний	Обґрунтована теоретична база розв'язання наукового завдання, якому присвячено роботу	25.10.2025
Синтез системи	Розробка комп'ютерної системи	15.11.2025
Розроблення програмного забезпечення	Розробка програмного забезпечення	29.11.2025
Експериментальний розділ	Проведення і обробка результатів експериментів	06.12.2025

Завдання видано _____ ____доц. Бешта Д.О.___
(підпис керівника) (прізвище, ініціали)

Дата видачі __01.10.2025 р._____

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ _____Скубицький О.С._____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка 87 с., 17 рис., 3 табл., 1 дод., 21 джерело.

SDN, LSTM, ПРОГНОЗУВАННЯ ТРАФІКУ, АДАПТИВНЕ УПРАВЛІННЯ, НЕЙРОННІ МЕРЕЖІ, DEEP LEARNING, OPENFLOW, QOS, MININET, RYU.

Об’єкт розробки − процеси маршрутизації та керування потоками даних у програмно-конфігурованих мережах (SDN).

Мета роботи – розробка та експериментальне дослідження ефективності адаптивної системи управління мережею SDN, що базується на механізмі прогнозування трафіку з використанням рекурентних нейронних мереж довгої короткочасної пам'яті.

Методи дослідження – у роботі застосовувались методи системного аналізу для оцінки існуючих підходів до Traffic Engineering, теорія масового обслуговування для аналізу затримок, математичне моделювання нейронних мереж (LSTM) для обробки часових рядів, а також імітаційне моделювання у середовищі Mininet для експериментальної перевірки ефективності запропонованого методу.

У першому розділі на основі матеріалів практик, інших науково-технічних джерел сформульоване наукове завдання, конкретизовані предмет та мета досліджень.

Виконано аналіз сучасних методів балансування навантаження в SDN, виявлено недоліки реактивних підходів (інерційність, втрати пакетів) та обмеження класичних статистичних моделей прогнозування (ARIMA). Обґрунтовано доцільність використання архітектури глибокого навчання LSTM для аналізу нелінійного мережевого трафіку.

У теоретичному розділі вирішено наукове завдання побудови математичної моделі прогнозування часових рядів трафіку з урахуванням довгострокових залежностей. Описано принципи взаємодії інтелектуального модуля з SDN-контролером.

У третьому розділі розглянуто проектування системи прогнозування мережевих потоків у SDN-середовищі на основі моделей LSTM та адаптивного управління трафіком.

У четвертому розділі проведено комплексний аналіз і опис розробки програмного забезпечення для системи адаптивного управління мережевим трафіком із застосуванням LSTM-моделі.

У п’ятому розділі було проведено комплексне моделювання роботи адаптивної системи прогнозування та управління мережевими ресурсами на основі SDN-архітектури та алгоритмів часових рядів.

В експериментальному розділі проведено порівняльне тестування розробленої системи адаптивного управління та традиційних методів маршрутизації в емульованому середовищі Mininet. Результати експерименту підтвердили, що запропонований підхід дозволяє зменшити коефіцієнт втрати пакетів та знизити середню затримку в мережі при пікових навантаженнях.

Практична цінність отриманих результатів полягає у створенні універсального програмного модуля прогнозування, який може бути інтегрований у промислові SDN-контролери для автоматизації керування трафіком у центрах обробки даних та корпоративних мережах.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

SDN – Software-Defined Networking – програмно-конфігурована мережа;

LSTM – Long Short-Term Memory – довготривала короткочасна пам’ять (тип рекурентної нейромережі);

ML – Machine Learning – машинне навчання;

AI – Artificial Intelligence – штучний інтелект;

CNN – Convolutional Neural Network – згорткова нейромережа;

GRU – Gated Recurrent Unit – рекурентна нейромережа з механізмом вентилів;

MAE – Mean Absolute Error – середня абсолютна похибка;

RMSE – Root Mean Squared Error – корінь середньоквадратичної похибки;

MAPE – Mean Absolute Percentage Error – середня абсолютна відносна похибка;

API – Application Programming Interface – програмний інтерфейс взаємодії;

QoS – Quality of Service – якість обслуговування;

RNN – Recurrent Neural Network – рекурентна нейромережа;

JSON – JavaScript Object Notation – формат обміну даними;

REST – Representational State Transfer – архітектурний стиль для веб-сервісів;

CLI – Command Line Interface – інтерфейс командного рядка;

TCP – Transmission Control Protocol – протокол керування передачею;

UDP – User Datagram Protocol – протокол користувацьких дейтаграм;

VNF – Virtual Network Function – віртуальна мережева функція;

NBI – Northbound Interface – північний інтерфейс SDN-контролера;

SBI – Southbound Interface – південний інтерфейс SDN-контролера.

ВСТУП

Сучасний етап розвитку телекомунікаційних мереж характеризується переходом до сервіс-орієнтованих архітектур, де критичними показниками ефективності стають не лише пропускна здатність, але й гарантована якість обслуговування (QoS) та безперервність надання послуг. Вибухоподібне зростання мультимедійного трафіку, хмарних обчислень та пристроїв IoT формує складні, нелінійні патерни навантаження, які важко обробляти традиційними методами маршрутизації.

Архітектура програмно-конфігурованих мереж (SDN), що базується на централізації площини керування, створює передумови для глобальної оптимізації потоків даних. Однак, стандартні механізми SDN здебільшого діють реактивно: система реагує на перевантаження вже після виникнення заторів та втрати пакетів. Для реалізації концепції адаптивного управління, яке здатне діяти на випередження, необхідне впровадження інтелектуальних агентів, здатних аналізувати історичні дані та передбачати стан мережі.

У цьому контексті особливу увагу привертають рекурентні нейронні мережі архітектури LSTM (Long Short-Term Memory). На відміну від класичних стохастичних моделей або простих нейромереж, LSTM здатні виявляти довгострокові часові залежності в трафіку, що робить їх ідеальним інструментом для прогнозування в умовах високої волатильності. Інтеграція LSTM-модуля в контур керування SDN дозволяє перейти від статичних правил до динамічної адаптації маршрутів, мінімізуючи затримки та підвищуючи надійність інфраструктури.

Практична цінність дослідження полягає у розробці механізму, який дозволяє мережевим операторам автоматизувати процеси Traffic Engineering, знизити операційні витрати та попередити деградацію сервісів у періоди пікових навантажень.

1 СТАН ПИТАННЯ І ПОСТАНОВКА ЗАВДАННЯ

1.1 Обґрунтування актуальності теми

Сучасні телекомунікаційні мережі характеризуються експоненціальним зростанням обсягів трафіку та високою динамічністю його параметрів. Впровадження технологій Інтернету речей (IoT), хмарних обчислень та потокового відео високої чіткості висуває жорсткі вимоги до якості обслуговування (QoS — Quality of Service). Традиційні мережеві архітектури, де функції керування жорстко прив'язані до апаратного забезпечення маршрутизаторів, стають «вузьким місцем» через складність масштабування та низьку гнучкість налаштувань.

Вирішенням цієї проблеми стала концепція програмно-конфігурованих мереж (SDN — Software-Defined Networking), яка відокремлює площину керування (Control Plane) від площини передачі даних (Data Plane). Централізований контролер SDN має глобальне бачення мережі, що дозволяє оптимізувати маршрутизацію.

Однак, більшість сучасних рішень в SDN використовують реактивні методи керування: контролер змінює правила маршрутизації лише після того, як перевантаження вже сталося (наприклад, коли завантаження каналу перевищило поріг у 80%). Це призводить до короткочасних втрат пакетів, збільшення затримок (Latency) та джитера.

Актуальність даного дослідження полягає у переході від реактивного до проактивного (попереджувального) керування. Використання методів глибокого навчання, зокрема мереж довгої короткострокової пам'яті (LSTM), дозволяє аналізувати часові ряди мережевого трафіку, виявляти приховані закономірності та прогнозувати навантаження. Це дає можливість контролеру SDN перерозподіляти потоки до моменту виникнення перевантаження, забезпечуючи стабільну роботу мережі.

1.2 Аналіз стану питання та існуючих рішень

Проблема ефективного розподілу ресурсів у мережах (Traffic Engineering, TE) не є новою, проте в контексті архітектури SDN вона набуває нових особливостей. Аналіз сучасних наукових публікацій та технічних рішень дозволяє класифікувати існуючі підходи до керування потоками на три основні групи: статичні, динамічні реактивні та динамічні проактивні (прогнозуючі).

1.2.1. Традиційні та статичні методи маршрутизації

Найбільш поширеним методом балансування навантаження в сучасних дата-центрах залишається ECMP (Equal-Cost Multi-Path). Алгоритм ECMP розподіляє потоки між кількома маршрутами з однаковою вартістю, використовуючи хешування полів заголовка пакету (IP-адреси джерела/призначення, порти).

Головною проблемою ECMP є так звана «колізія слонових потоків» (Elephant Flow Collision). Оскільки алгоритм не враховує поточне завантаження каналів, він може спрямувати два високошвидкісних потоки («слони») на один фізичний лінк, тоді як сусідні лінки залишатимуться пустими. Як зазначається у дослідженнях [1], це призводить до неефективного використання пропускної здатності, яке може складати лише 30–40% від номіналу мережі.

1.2.2. Динамічні реактивні методи в SDN

Впровадження технології SDN відкрило можливості для централізованого моніторингу стану мережі в режимі реального часу, що стало основою для динамічних реактивних алгоритмів. Принцип роботи таких систем базується на циклічному опитуванні комутаторів контролером для отримання статистики про кількість переданих пакетів та байтів. Алгоритм керування постійно порівнює поточне завантаження каналів із заданим пороговим значенням (наприклад, 80% від максимальної пропускної здатності). У разі фіксації перевищення порогу контролер ініціює процедуру

перерахунку шляху та інсталює нові правила маршрутизації, перемикаючи частину трафіку на резервні канали.

Попри очевидну перевагу над статичними методами, реактивний підхід має низку критичних недоліків, детально проаналізованих у роботі Ш. Мохтар, [2]. Головною проблемою є інерційність контуру керування: сумарний час, необхідний для збору статистики, її передачі на контролер, прийняття рішення та оновлення таблиць потоків, створює часовий лаг (latency). У сучасних мережах часто виникають так звані «мікросплески» (microbursts) — короткочасні різкі стрибки трафіку, які можуть переповнити буфери комутатора ще до того, як контролер встигне зреагувати на зміну ситуації. Крім того, використання жорстких порогових значень часто призводить до ефекту «брязкання маршрутів» (route flapping), коли трафік починає осцилювати між основним та резервним каналами через незначні коливання навантаження біля граничної позначки, що дестабілізує роботу мережі та підвищує навантаження на сам контролер.

1.2.3. Методи на основі лінійного прогнозування

Для подолання затримок реактивних систем дослідники почали застосовувати методи прогнозування часових рядів, такі як ARIMA (AutoRegressive Integrated Moving Average) та фільтр Калмана. Ці моделі намагаються передбачити значення трафіку на крок вперед, базуючись на історичних даних.

Б. Бібак у своїй дисертації [3] демонструє, що лінійні моделі ефективні лише для стаціонарного трафіку з повільними змінами. Однак сучасний мультимедійний трафік характеризується високою волатильністю, нелінійністю та самоподібністю (self-similarity). Лінійні алгоритми не здатні коректно апроксимувати складні патерни поведінки користувачів, що призводить до високої похибки прогнозування (RMSE) і хибних спрацьовувань системи керування.

1.2.4. Методи на основі глибокого навчання (Deep Learning)

В останні роки фокус наукових досліджень у сфері керування мережами змістився від стохастичних моделей до методів глибокого навчання (Deep Learning). Цей перехід зумовлений здатністю нейронних мереж апроксимувати складні нелінійні функції без необхідності ручного вибору ознак. На початкових етапах розвитку цього напрямку активно застосовувалися класичні штучні нейронні мережі прямого поширення (ANN — Artificial Neural Networks), або багатошарові персептрони. Хоча ANN здатні виявляти нелінійні залежності у вхідних даних, їх архітектурним обмеженням є відсутність механізму збереження контексту: кожен вхідний вектор обробляється незалежно від попередніх. Це робить їх малоефективними для аналізу мережевого трафіку, який за своєю природою є часовим рядом, де поточне значення навантаження тісно корелює з попередніми подіями.

Для роботи з послідовностями даних було розроблено рекурентні нейронні мережі (RNN — Recurrent Neural Networks). Головною особливістю RNN є наявність зворотних зв'язків, що формують внутрішню «пам'ять» мережі. Це дозволяє враховувати часовий контекст та історію попередніх станів. Проте, як зазначається у фундаментальних роботах з машинного навчання, класичні RNN страждають від проблеми зникаючого градієнта (Vanishing Gradient Problem). При навчанні на довгих послідовностях градієнти помилки, що поширюються у зворотному напрямку в часі, експоненціально зменшуються, через що ваги початкових шарів перестають оновлюватися. На практиці це означає, що RNN не здатна «запам'ятати» події, що сталися багато кроків тому, втрачаючи здатність виявляти довгострокові тренди.

На сьогодні найбільш перспективним напрямком є використання удосконалених рекурентних архітектур, таких як LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit). Завдяки введенню спеціальних шлюзових механізмів (gates), ці мережі вирішують проблему зникаючого градієнта. Вони здатні селективно запам'ятовувати важливі довгострокові

залежності, такі як добова або тижнева періодичність навантаження (сезонність), та ефективно фільтрувати випадковий шум, характерний для IP-мереж.

Незважаючи на значну кількість теоретичних публікацій, аналіз існуючих рішень виявляє суттєву прогалину: більшість робіт розглядають прогнозування ізольовано від процесу маршрутизації, або ж пропонують громіздкі моделі, час інференсу (обробки запиту) яких перевищує допустимі межі для роботи в реальному часі. У зв'язку з цим, актуальним науково-прикладним завданням залишається створення легковагого адаптивного модуля на базі LSTM, інтегрованого безпосередньо в контур керування SDN-контролера. Такий підхід дозволить поєднати високу точність прогнозування з швидкодією прийняття рішень, забезпечуючи проактивне усунення перевантажень.

1.3 Завдання та мета роботи

На основі аналізу предметної області можна сформулювати мету та задачі роботи.

Мета дослідження. Розробка та експериментальне дослідження ефективності адаптивної системи управління мережею SDN, що базується на механізмі прогнозування трафіку з використанням рекурентних нейронних мереж довгої короткочасної пам'яті.

Об'єкт дослідження. Процеси маршрутизації та керування потоками даних у програмно-конфігурованих мережах (SDN).

Предмет дослідження. Методи та алгоритми адаптивного балансування навантаження з використанням нейромережевих моделей прогнозування LSTM.

Завдання дослідження.

1. Провести аналіз існуючих методів прогнозування мережевого трафіку та обґрунтувати вибір рекурентної нейронної мережі LSTM як найбільш придатного інструменту для роботи з нелінійними часовими

рядамиНаукова новизна полягає у вдосконаленні методу керування потоками в SDN шляхом інтеграції LSTM-прогнозування в контур керування контролера, що, на відміну від існуючих реактивних підходів, дозволяє попередити перевантаження каналів зв'язку.

2. Розробити структурну схему адаптивної системи управління, що включає Модуль збору телеметрії, Модуль LSTM-прогнозування та Модуль прийняття рішень (див.).

3. Деталізувати логічну схему функціонування системи, визначивши послідовність процесів навчання, прогнозування та проактивного прийняття рішень контролером.

4. Розробити програмний прототип SDN-контролера на базі Python з інтегрованою LSTM-моделлю для демонстрації проактивної логіки.

5. Провести експерименти з використанням часового ряду трафіку для оцінки точності LSTM-моделі за ключовими метриками: RMSE та MAPE.

2 ТЕОРЕТИЧНИЙ РОЗДІЛ

2.1 Основи побудови адаптивних систем керування трафіком в SDN

2.1.1 Концептуальні засади та архітектура програмно-конфігурованих мереж

У сучасних телекомунікаційних системах ключову роль відіграє здатність інфраструктури до швидкої адаптації під змінні профілі навантаження. Традиційні мережі, побудовані на принципах децентралізованого керування та статичної маршрутизації, демонструють недостатню гнучкість в умовах динамічних сценаріїв трафіку, характерних для хмарних обчислень та IoT. Вирішенням цієї проблеми стала зміна парадигми побудови мереж і перехід до концепції Software-Defined Networking (SDN).

Фундаментальною відмінністю SDN є логічне відокремлення площини керування (Control Plane) від площини передачі даних (Data Plane). У цій архітектурі мережеві пристрої (комутатори) виконують лише функцію високошвидкісної пересилки пакетів згідно з правилами, які встановлюються централізованим контролером. Контролер виступає в ролі «мозку» мережі, маючи глобальне бачення топології та станів усіх каналів зв'язку. Взаємодія між рівнями здійснюється через стандартизовані інтерфейси, найбільш поширеним з яких є протокол OpenFlow. Саме він дозволяє контролеру в реальному часі модифікувати таблиці потоків (Flow Tables) на комутаторах, реалізуючи складні сценарії Traffic Engineering, які неможливо втілити на базі традиційних протоколів OSPF чи BGP.

Проте, наявність централізованого керування сама по собі не гарантує ефективності. Критичним аспектом залишається часова затримка в контурі керування. Без механізмів прогнозування контролер змушений діяти реактивно — тобто втручатися у розподіл трафіку лише після того, як перевантаження вже виникло і призвело до втрати пакетів. Це зумовлює необхідність інтеграції в архітектуру SDN аналітичних модулів, здатних передбачати стан мережі, [2].

2.1.2. Проблематика аналізу та прогнозування мережевого трафіку

Ефективність адаптивного керування напряму залежить від точності прогнозування навантаження. Мережевий трафік є складним стохастичним процесом, для якого характерні нелінійність, нестаціонарність та властивість самоподібності (self-similarity). Остання означає, що сплески активності (bursts) спостерігаються на різних часових масштабах, що ускладнює їх згладжування.

Традиційні методи аналізу часових рядів, такі як ковзне середнє (MA) або авторегресійні моделі (ARIMA), базуються на припущеннях про лінійність процесів. Вони демонструють прийнятну точність лише на стабільному трафіку, але виявляються неефективними при виникненні раптових пікових навантажень («мікросплесків»). Помилка прогнозу в такі моменти може призвести до несвоєчасного виділення ресурсів, [3].

Теоретичне обґрунтування необхідності точного прогнозування базується на теорії масового обслуговування. Згідно з формулою Клейнрока для систем M/M/1, середня затримка в черзі зростає експоненціально при наближенні інтенсивності вхідного потоку до пропускної здатності каналу. Це означає, що завданням системи прогнозування є не просто передбачення абсолютних значень трафіку, а завчасне виявлення трендів наближення до критичних точок насичення, коли навіть незначне збільшення навантаження призводить до різкої деградації якості обслуговування (QoS).

2.1.3. Застосування нейронних мереж LSTM для прогнозування часових рядів

Враховуючи обмеження класичних статистичних методів, у даному дослідженні пропонується використання апарату глибокого навчання (Deep Learning). Серед різних архітектур нейронних мереж найбільш перспективними для задач аналізу трафіку є рекурентні нейронні мережі (RNN), специфіка яких полягає у здатності зберігати інформацію про попередні стани системи.

Однак класичні RNN мають суттєвий недолік — проблему зникаючого градієнта (Vanishing Gradient Problem), що унеможливлює навчання на довгих послідовностях даних. Вирішенням цієї проблеми є архітектура LSTM (Long Short-Term Memory). На відміну від звичайних RNN, модуль LSTM містить спеціальні механізми — вентилі (gates): вхідний вентиль, вентиль забування та вихідний вентиль.

Вентиль забування дозволяє мережі ігнорувати нерелевантні дані (наприклад, випадковий шум або короткочасні флуктуації), тоді як вхідний вентиль фіксує важливі довгострокові залежності (наприклад, добову періодичність навантаження). Така архітектура дозволяє моделі ефективно навчатися на історичних даних мережевого трафіку, виявляючи приховані закономірності, недоступні для лінійних моделей, та забезпечувати високу точність прогнозу навіть в умовах високої волатильності.

Формально процес обробки даних у комірці LSTM на часовому кроці t описується наступною системою рівнянь:

Вентиль забування (Forget Gate) визначає, яку частку інформації з попереднього стану комірки C_{t-1} слід видалити (забути).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \quad (2.1)$$

Вхідний вентиль (Input Gate): вирішує, яка нова інформація буде збережена в стані комірки. Цей етап складається з обчислення значень вентиля i_t та створення вектору кандидатів $\hat{C}_t$.

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

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \quad (2.2)$$

Оновлення стану комірки (Cell State Update): старий стан модифікується шляхом забування застарілої інформації та додавання нової.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \quad (2.3)$$

Вихідний вентиль (Output Gate): визначає значення прихованого стану h_t, який є виходом комірки і передається на наступний крок.

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

$$h_t = o_t \odot \tanh(C_t) \quad , \qquad (2.4)$$

де, x_t — вхідний вектор (значення трафіку) у момент часу t;

h_{t-1} — прихований стан (hidden state) попереднього кроку;

W та b — матриці вагових коефіцієнтів та вектори зміщення (bias) для відповідних вентилів;

σ — сигмоїдальна функція активації (значення в діапазоні [0,1]);

tanh — гіперболічний тангенс (значення в діапазоні [−1,1]);

⊙ — операція поелементного множення (добуток Адамара).

Саме така математична структура дозволяє вентилю забування ігнорувати нерелевантні дані (наприклад, випадковий шум або короткочасні флуктуації), тоді як вхідний вентиль фіксує важливі довгострокові залежності (наприклад, добову періодичність навантаження). Така архітектура дозволяє моделі ефективно навчатися на історичних даних мережевого трафіку, виявляючи приховані закономірності, недоступні для лінійних моделей, та забезпечувати високу точність прогнозу навіть в умовах високої волатильності.

2.1.4. Методи забезпечення стабільності адаптивного керування

Інтеграція прогнозної моделі LSTM у середовище SDN створює замкнений контур автоматичного керування. Важливим теоретичним аспектом при цьому є забезпечення стабільності системи. Пряме використання прогнозних значень для зміни маршрутів може призвести до ефекту осциляції або «брязкання маршрутів» (route flapping). Це явище виникає, коли мережа занадто часто перемикає потоки між основним та резервним каналами у відповідь на незначні коливання прогнозу, що створює додаткове навантаження на контролер та збільшує джитер.

Для уникнення цього необхідно застосовувати методи згладжування керуючих впливів та вводити гістерезис у процес прийняття рішень. Це означає, що перемикання на новий маршрут повинно відбуватися лише тоді, коли прогнозована вигода від оптимізації перевищує певний поріг «вартості» реконфігурації.

Також слід враховувати специфіку обробки потокових даних. У реальному середовищі телеметрія від комутаторів надходить асинхронно, що вимагає буферизації та синхронізації даних перед подачею на вхід нейромережі. Актуальним є використання підходів онлайн-навчання (online learning), які дозволяють донавчати модель у процесі роботи, адаптуючи її до змін у структурі трафіку без зупинки системи.

Таким чином, розробка ефективної системи адаптивного керування в SDN вимагає комплексного підходу, що поєднує архітектурні переваги програмованих мереж, потужний математичний апарат LSTM для прогнозування та алгоритмічні методи стабілізації для прийняття рішень, [3].

2.2 Основні принципи побудови SDN-мереж як базису для інтелектуального керування

Архітектурна парадигма програмно-конфігурованих мереж (Software-Defined Networking) створює фундамент для реалізації адаптивних алгоритмів керування, які були недосяжні в рамках традиційних мережевих моделей. На відміну від класичної децентралізованої архітектури, де логіка обробки пакетів жорстко пов'язана з апаратним забезпеченням кожного окремого маршрутизатора, SDN пропонує концепцію повної програмованості мережевої інфраструктури. Ця гнучкість є критично важливою для інтеграції моделей машинного навчання, таких як LSTM, оскільки дозволяє перетворити мережу з набору статичних пристроїв на динамічну систему, керовану даними.

Функціонування SDN базується на трьох основоположних принципах, кожен з яких відіграє ключову роль у побудові системи проактивного прогнозування:

1. Декупликація площин керування та пересилання (Control Plane & Data Plane). В основі SDN лежить фізичне та логічне відокремлення «інтелекту» мережі від механізмів транспортування даних. Функції прийняття рішень (Control Plane) винесено в окремий програмний компонент — SDN-контролер, тоді як мережеві пристрої (комутатори) залишаються відповідальними виключно за високошвидкісну пересилку пакетів (Data Plane). У контексті даного дослідження це розділення є вирішальним: воно дозволяє розвантажити комутатори від складних обчислень і передати задачу аналізу трафіку та прогнозування на серверні потужності, де розгорнуто нейромережу LSTM.

2. Логічно централізований контроль та глобальна видимість. Завдяки централізації, контролер володіє повною (глобальною) інформацією про топологію мережі та стан усіх каналів зв'язку в реальному часі. У традиційних мережах маршрутизатори приймають рішення на основі обмеженої локальної інформації, що часто призводить до конфліктів та субоптимального розподілу ресурсів. Для коректної роботи моделі LSTM критично важливо мати доступ до агрегованої статистики з усієї мережі, а не фрагментарних даних. Централізований підхід дозволяє формувати цілісні часові ряди навантаження, на основі яких модель виявляє приховані залежності та тренди, [2].

3. Програмованість через відкриті інтерфейси Використання стандартизованих протоколів, таких як OpenFlow, забезпечує уніфікований спосіб взаємодії між контролером та обладнанням різних виробників. Це перетворює мережу на програмовану платформу, де політики керування трафіком визначаються програмним кодом, а не прошивкою пристроїв. Саме ця особливість дозволяє автоматизувати реакцію на прогнози LSTM: як тільки модель передбачає зростання навантаження, контролер може миттєво і без втручання адміністратора інсталювати нові правила маршрутизації через API. Як зазначається у роботах Б. Бібака, така архітектура мінімізує час реакції та дозволяє реалізувати стратегію випереджального керування, [3].

Таким чином, принципи SDN не лише спрощують адміністрування, але й створюють необхідне технологічне середовище для розгортання інтелектуальних систем. Поєднання централізованого збору телеметрії та гнучкості програмного керування дозволяє повною мірою реалізувати потенціал LSTM-прогнозування для адаптивного балансування навантаження.

Теоретичне обґрунтування необхідності точного прогнозування базується на теорії масового обслуговування. Якщо розглядати комутатор як систему обслуговування типу M/M/1 (з пуассонівським вхідним потоком та експоненційним часом обслуговування), то середня затримка пакету в черзі T описується формулою Клейнрока:

$$T = \frac{1}{\mu - \lambda} = \frac{1}{\mu(1-\rho)} \qquad (2.5)$$

де:

μ — швидкість обслуговування пакетів (пропускна здатність каналу);

λ — інтенсивність вхідного потоку пакетів;

ρ=λ/μ — коефіцієнт завантаження каналу (utilization).

З наведеної залежності видно, що при наближенні навантаження до пропускної здатності (ρ→1), затримка T зростає гіперболічно (прямує до нескінченності). Це означає, що завданням системи прогнозування є не просто передбачення абсолютних значень трафіку, а завчасне виявлення моментів, коли ρ наближається до критичних значень (наприклад, ρ>0.8), щоб контролер встиг перерозподілити потоки до початку деградації QoS, [8].

2.3 Архітектура SDN як середовище для розгортання інтелектуальних систем

У контексті розробки системи адаптивного керування на основі LSTM, архітектура програмно-конфігурованих мереж (SDN) виступає не просто об'єктом керування, а необхідним технологічним фундаментом. Ключова відмінність SDN від традиційних мереж — декупликація (розділення) площини керування та площини передачі даних — дозволяє реалізувати

замкнений цикл керування (Control Loop), необхідний для функціонування моделей машинного навчання.

На відміну від класичних мереж, де рішення приймаються локально на кожному пристрої, SDN централізує логіку в контролері. Це надає прогнозній моделі доступ до глобальної статистики мережі, що є критичною умовою для коректного навчання нейромережі та виявлення складних просторово-часових залежностей у трафіку.

Архітектура SDN ієрархічно поділяється на три функціональні рівні, взаємодія між якими забезпечується через стандартизовані інтерфейси.

Для деталізації принципів інформаційного обміну через північний інтерфейс (Northbound Interface) доцільно розглянути приклад організації RESTful API-запитів, що забезпечують комунікацію із зовнішнім модулем прогнозування. Такий підхід дозволяє наочно продемонструвати механізм програмної взаємодії, за допомогою якого інтелектуальна підсистема отримує доступ до телеметрії та здійснює керування потоками в SDN, рис.2.1.

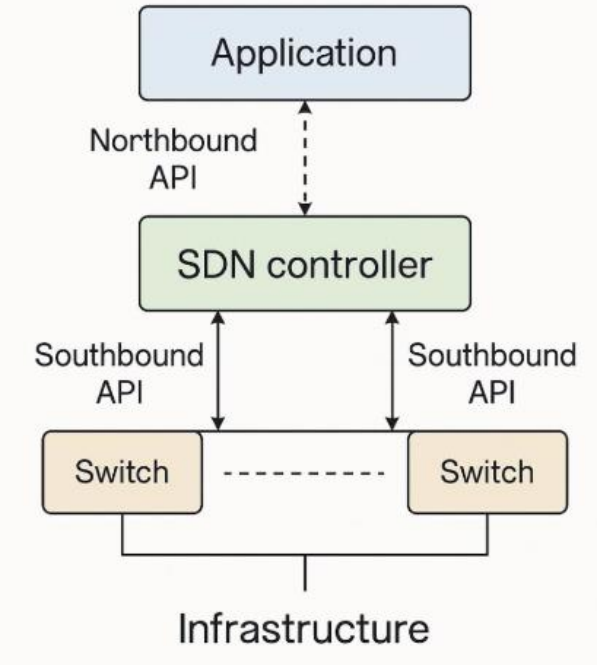

Рисунок 2.1 — Архітектура SDN та взаємодія її компонентів

2.3.1 Рівень інфраструктури (Data Plane)

Нижній рівень архітектури утворений мережевими пристроями пересилання (комутаторами), які можуть бути як фізичними, так і віртуальними (наприклад, Open vSwitch у хмарних середовищах). У парадигмі SDN ці пристрої позбавлені складної інтелектуальної складової. Їхня основна функція — високошвидкісна обробка пакетів згідно з таблицями потоків (Flow Tables). Для задачі прогнозування цей рівень виступає генератором «сирих» даних. Комутатори ведуть облік кількості переданих пакетів та байтів для кожного потоку. Саме ці лічильники, що зчитуються з певною періодичністю, формують часові ряди, які подаються на вхід LSTM-моделі для аналізу, [4].

2.3.2 Рівень керування (Control Plane)

Ядром системи є SDN-контролер (Network Operating System), такий як ONOS, Ryu або OpenDaylight. Контролер підтримує глобальну базу даних топології мережі та станів каналів. У розроблюваній системі контролер виконує роль посередника:

1. Агрегатор даних. Він збирає фрагментарну статистику з усіх комутаторів і приводить її до уніфікованого вигляду.
2. Виконавчий орган. Отримавши прогноз про перевантаження від модуля LSTM, контролер транслює це рішення у конкретні низькорівневі команди (Flow Mods) для зміни маршрутів на комутаторах. Сучасні контролери підтримують кластеризацію, що забезпечує відмовостійкість системи керування та дозволяє обробляти великі масиви телеметрії без затримок, [5].

2.3.3 Рівень додатків (Application Plane)

Саме на цьому рівні розгортається розроблюваний модуль адаптивного прогнозування. Мережа в SDN розглядається як програмована платформа, де логіка керування трафіком реалізується у вигляді програмних додатків (Business Logic). Модуль LSTM функціонує як зовнішній додаток, що аналізує історичні дані, виконує інференс (передбачення навантаження на крок t+1) та формує рекомендації для контролера щодо перерозподілу потоків.

2.3.4 Інтерфейси взаємодії

Ефективність роботи прогнозної моделі критично залежить від пропускної здатності та затримок інтерфейсів, що зв'язують рівні архітектури.

Південний інтерфейс (Southbound API) забезпечує зв'язок між контролером та комутаторами. Найпоширенішим стандартом є протокол OpenFlow. Він виконує подвійну функцію:

- моніторинг, який транспортує статистичні дані від комутаторів до контролера (повідомлення OFP_STATS_REPLY).
- керування, що доставляє оновлені правила маршрутизації до пристроїв (OFP_FLOW_MOD). Важливою характеристикою є підтримка зворотного зв'язку: комутатор може сповістити контролер про розрив з'єднання або появу нового невідомого потоку, [6].

Північний інтерфейс (Northbound API), надає абстрагований доступ до ресурсів мережі для додатків. Зазвичай реалізується через архітектуру RESTful API. Через цей інтерфейс модуль LSTM періодично опитує контролер (HTTP GET) для отримання свіжої статистики та надсилає команди керування (HTTP POST/PUT). Використання REST API дозволяє розробляти модуль прогнозування на мовах високого рівня (наприклад, Python з бібліотеками Keras/TensorFlow), незалежно від мови реалізації самого контролера (Java/C++).

Таким чином, архітектура SDN створює ідеальну екосистему для впровадження штучного інтелекту. Логічна централізація забезпечує необхідну видимість для навчання моделі, а програмовані інтерфейси дозволяють автоматизувати реакцію мережі на прогнози, реалізуючи концепцію самоадаптивної системи.

2.4 Аналіз існуючих методів прогнозування навантаження в контексті SDN

Критичною умовою побудови системи адаптивного керування є вибір математичного апарату, здатного з високою точністю передбачати динаміку

мережевого трафіку. У процесі дослідження було проведено порівняльний аналіз двох фундаментальних підходів: класичних стохастичних моделей та методів обчислювального інтелекту (Machine Learning), з метою визначення оптимального інструменту для інтеграції в SDN-контролер.

2.4.1 Обмеження традиційних статистичних моделей

Історично першими методами прогнозування часових рядів були статистичні алгоритми. Серед них найбільш поширеними є:

- ковзне середнє (MA — Moving Average), це базовий метод згладжування, який ефективний лише для виявлення загальних трендів у стабільному потоці даних;
- експоненційне згладжування (ES) надає більшу вагу останнім спостереженням, що дозволяє швидше реагувати на зміни, проте погано враховує сезонність;
- ARIMA (AutoRegressive Integrated Moving Average) вважається «золотим стандартом» класичної статистики. Модель будує прогноз на основі лінійної комбінації попередніх значень та похибок.

Як свідчать результати тестування, наведені у роботах Б. Бібака [2], моделі класу ARIMA демонструють високу точність на коротких горизонтах прогнозування за умови, що трафік є стаціонарним або має чітко виражену лінійну структуру. Однак у реальних мережах SDN трафік має стохастичну, вибухову природу (bursty traffic) з нелінійними залежностями. У таких умовах статистичні методи виявляються інертними: вони не здатні своєчасно передбачити різкий сплеск навантаження, що призводить до затримки реакції системи керування та втрати пакетів, [6].

2.4.2 Переваги методів машинного навчання

Альтернативою лінійним моделям є методи машинного навчання (ML), які здатні апроксимувати складні нелінійні функції.

Ансамблеві методи (Random Forest, XGBoost) демонструють високу стійкість до шуму та здатність працювати з табличними даними. Проте їх

архітектура не враховує повною мірою часову послідовність подій — для них важливий набір ознак, а не їх порядок у часі.

Штучні нейронні мережі (ANN) здатні виявляти нелінійні патерни, але прості персептрони не мають «пам'яті», що робить їх неефективними для аналізу часових рядів.

Найбільш перспективним класом алгоритмів для задач Traffic Engineering є рекурентні нейронні мережі (RNN), і зокрема їх модифікація — LSTM (Long Short-Term Memory). Завдяки наявності комірок пам'яті, LSTM здатна моделювати довгострокові залежності. Це означає, що система може «запам'ятати» патерн навантаження, який відбувався кілька годин тому (наприклад, ранковий пік), і використати це знання для прогнозу. Дослідження підтверджують, що LSTM перевершує як ARIMA, так і прості RNN у задачах прогнозування волатильного трафіку, (таб. 2.1), забезпечуючи зниження середньоквадратичної помилки (RMSE) на 15–20%, [7].

Таблиця 2.1 — Порівняльний аналіз методів прогнозування для SDN

Метод	**Здатність до навчання нелінійності**	**Врахування довгих залежностей**	**Обчислювальна складність**	**Придатність для реального часу**
ARIMA	Низька	Обмежена	Низька	Висока
Random Forest	Висока	Низька	Середня	Середня
XGBoost	Дуже висока	Середня	Висока	Середня
LSTM	**Дуже висока**	**Висока**	**Висока**	**Висока (на етапі інтерфейсу)**
Prophet	Середня	Висока (сезонність)	Середня	Висока

2.4.3 Специфіка застосування LSTM в адаптивному керуванні

Вибір методу LSTM для даної роботи зумовлений не лише точністю, але й можливістю реалізації онлайн-навчання. У середовищі SDN дані надходять безперервним потоком. Модель LSTM можна донавчати інкрементно, адаптуючи її ваги до нових патернів трафіку (наприклад, зміни профілю користувачів у вихідні дні) без повної зупинки системи.

Також важливим етапом, виявленим у ході аналізу, є попередня обробка даних (Feature Engineering). Оскільки LSTM чутлива до масштабу вхідних даних, критично важливим є застосування методів нормалізації (MinMax Scaling) та очищення від аномальних викидів, які можуть дестабілізувати процес навчання.

Таким чином, проведений аналіз дозволяє зробити висновок: для реалізації системи адаптивного керування в SDN, де критичними є точність прогнозування нелінійних сплесків та здатність до самоадаптації, архітектура LSTM є безальтернативним вибором порівняно з класичними статистичними методами, [1].

2.5 Висновки по розділу

Підсумовуючи результати теоретичного дослідження, проведеного у першому розділі, можна стверджувати, що архітектура програмно-конфігурованих мереж створює необхідне технологічне підґрунтя для переходу від реактивних до проактивних стратегій керування трафіком. Систематизований аналіз принципів функціонування SDN, зокрема декупликації площини керування та наявності програмованих інтерфейсів, підтвердив технічну можливість інтеграції інтелектуальних агентів безпосередньо у замкнений контур прийняття рішень контролера. У ході порівняльного аналізу математичних апаратів було виявлено неспроможність класичних стохастичних моделей, таких як ARIMA, забезпечити необхідну точність прогнозування в умовах високодинамічного, нелінійного трафіку сучасних мультисервісних мереж. Натомість науково обґрунтовано доцільність застосування рекурентних нейронних мереж архітектури LSTM, специфічна структура яких дозволяє ефективно моделювати довгострокові часові залежності та ігнорувати шум, що є критичним для задач Traffic Engineering. Окрім вибору моделі, у розділі визначено ключові вимоги до попередньої обробки та нормалізації телеметричних даних, що є передумовою стабільності процесу навчання. Отримані аналітичні висновки та сформульовані вимоги до системи адаптивного керування слугують теоретичним базисом для переходу до наступного етапу роботи — проектування архітектури та програмної реалізації модуля LSTM-прогнозування в середовищі SDN.

3 СИНТЕЗ КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ АДАПТИВНОГО УПРАВЛІННЯ МЕРЕЖЕВИМИ ПОТОКАМИ В SDN-СЕРЕДОВИЩІ

3.1 Цілі впровадження системи

Перед впровадженням системи прогнозування мережевих потоків на основі LSTM-моделі [9] в SDN-середовищі визначено такі цілі:

– забезпечення можливості прогнозування інтенсивності мережевих потоків у реальному часі;

– підвищення ефективності прийняття рішень контролером SDN на основі прогнозованих параметрів трафіку;

– реалізація адаптивного управління мережевими ресурсами з урахуванням динамічних змін навантаження;

– зниження затримок і ймовірності перевантаження шляхом попереджувального перерозподілу трафіку;

– інтеграція LSTM-модуля прогнозування до існуючої архітектури SDN-контролера;

– забезпечення можливості масштабування та подальшої модифікації системи для розширення її функціональності.

3.2 Формулювання технічних вимог до системи

Система прогнозування мережевих потоків на основі LSTM у SDN-середовищі повинна складатися з таких основних компонентів:

– модуль збору та попередньої обробки статистики мережевого трафіку;

– модуль LSTM-прогнозування параметрів трафіку;

– модуль адаптивного управління потоками, інтегрований із SDN-контролером;

– інтерфейс адміністратора для моніторингу роботи системи, перегляду прогнозів і параметрів управління;

– API-шлюз для взаємодії між модулями та обміну даними з мережею.

Архітектура системи повинна бути побудована за принципами мікросервісної архітектури, що забезпечить гнучкість, масштабованість та можливість розгортання у хмарних середовищах (AWS, Azure, Google Cloud). Кожен функціональний модуль повинен бути упакований у Docker-контейнер, що спрощує розгортання, оновлення та підтримку системи.

Модуль прогнозування має бути реалізований мовою Python з використанням фреймворків TensorFlow або PyTorch, що забезпечують побудову та навчання LSTM-мереж достатньої складності для часових рядів мережевого трафіку.

Модуль взаємодії з SDN повинен бути написаний з використанням API контролера (OpenDaylight, ONOS або Ryu) та реалізовувати:

– підключення до northbound API;
– передачу прогнозів модулю управління;
– динамічну зміну правил маршрутизації (flow rules).

Серверна частина системи прогнозування та управління має працювати у асинхронному режимі для забезпечення низької затримки обробки даних. Для цього можуть використовуватися технології Node.js, FastAPI або gRPC для побудови високопродуктивних сервісів.

Інтерфейс адміністратора повинен бути реалізований із використанням сучасних веб-технологій (React або Vue.js), що забезпечить модульність компонентів, адаптивність, інтерактивність та зручність моніторингу стану мережі.

Система повинна підтримувати:

– високочастотний прийом даних від SDN-контролера;
– обробку великих обсягів трафіку;
– низьку затримку в процесі прогнозування й прийняття рішень;

– можливість подальшого розширення функціоналу та інтеграції нових моделей машинного навчання.

3.2.1 Вимоги до функцій, виконуваних системою

Система прогнозування мережевих потоків на основі LSTM [10] у SDN-середовищі повинна виконувати такі основні функції:

– здійснювати безперервний збір та агрегування статистичних даних про мережевий трафік із SDN-контролера;

– проводити попередню обробку отриманих даних (нормалізація, фільтрація, формування часових вікон) перед подачею до LSTM-моделі;

– виконувати прогнозування параметрів трафіку в реальному або квазі-реальному часі;

– надавати контролеру SDN рекомендації щодо управління потоками на основі прогнозних результатів;

– динамічно змінювати правила маршрутизації (flow rules) для запобігання перевантаженню та оптимізації розподілу ресурсів;

– забезпечувати адміністративний інтерфейс для перегляду прогнозів, журналів подій, стану сервісів та поточних мережевих параметрів;

– підтримувати можливість створення різних рівнів доступу (адміністратор, оператор, аналітик) з відповідними правами взаємодії із системою;

– забезпечувати захищений обмін даними між модулями системи та SDN-контролером (через TLS або інші криптографічні механізми);

– вести журналізацію та аудит дій користувачів і системних подій для забезпечення прозорості та відстежуваності роботи;

– підтримувати можливість інтеграції нових моделей прогнозування або аналітичних модулів без повної зміни архітектури системи.

3.2.2 Вимоги до видів забезпечення

Для реалізації системи прогнозування мережевих потоків у SDN-середовищі необхідно встановити таке програмне забезпечення:

– операційну систему UNIX-подібного типу для серверної частини (Debian, Ubuntu Server, CentOS або інші сумісні дистрибутиви);

– Docker та Docker Compose для контейнеризації окремих модулів системи (модуль збору даних, модуль LSTM-прогнозування, API-шлюз, модуль управління потоками);

– Python 3.10+ з бібліотеками TensorFlow або PyTorch для розгортання та навчання LSTM-моделі;

– сервер SDN-контролера (OpenDaylight, ONOS або Ryu), встановлений на окремому або віртуалізованому вузлі;

– систему моніторингу та логування (Prometheus, Grafana, ELK-stack) для відстеження роботи сервісів;

– за потреби – можливість розгортання системи у хмарних середовищах, таких як AWS, Azure або Google Cloud, які підтримують Docker-контейнери.

Сервер, на якому працюватимуть модулі прогнозування та обробки даних, повинен відповідати таким мінімальним технічним вимогам:

– 8-ядерний процесор із підтримкою багатопотоковості;

– об’єм диска не менше 1 ТБ для зберігання логів, даних трафіку й моделей;

– оперативна пам’ять обсягом не менше 32–64 ГБ (рекомендовано 64 ГБ для навчання та інференсу LSTM-моделей у реальному часі);

– наявність підтримки віртуалізації для розгортання контейнеризованих сервісів.

Серверна частина системи повинна працювати під керуванням однієї з таких підтримуваних UNIX-подібних операційних систем:

– CentOS: версії 7 або 8;

– Debian: версії 10 або 11;

– Ubuntu Server: версії 20.04 LTS або 22.04 LTS.

3.2.3 Функціональна схема

Архітектурним принципом побудови системи прогнозування мережевих потоків у SDN-середовищі було обрано мікросервісну архітектуру, що дозволяє розробляти, тестувати та розгортати програмне забезпечення як окремі модулі з чітко визначеними вхідними та вихідними даними. Такий підхід забезпечує гнучкість у виборі технологій, дозволяє не прив’язуватись до конкретних мов програмування та швидко модернізувати модулі з розвитком технологій і збільшенням обсягів обробки даних. Крім того, ізоляція компонентів позитивно впливає на тестування програмного коду та зменшує ймовірність помилок у системі.

Функціональна схема системи включає кілька основних компонентів (рис.3.1). SDN-середовище складається з мережевих пристроїв, які передають дані про трафік до SDN-контролера, а контролер агрегує ці дані та зберігає їх у базі даних часових рядів (TSDB). Сервіс прогнозування обробляє історичні дані трафіку через API або webhook, виконує їх попередню обробку та передає у модель LSTM, яка генерує прогнозовані значення навантаження мережі. Результати прогнозу зберігаються в базі даних та передаються у модуль адаптивного управління, де приймаються рішення щодо зміни правил маршрутизації та QoS, які надсилаються назад до SDN-контролера для корекції потоків у мережі.

Інтерфейс користувача забезпечує адміністратору доступ до перегляду прогнозованих даних, стану мережі та прийнятих рішень через графічний інтерфейс, здійснює запити до сервісу обробки даних та отримує від нього інформацію про поточний стан мережі, історичні метрики та прогнозоване навантаження. Архітектура системи забезпечує інтеграцію всіх компонентів через API-шлюз, який координує обмін даними, гарантує безпечну передачу інформації та спрощує масштабування сервісів. Така структурна схема дозволяє ефективно виконувати функції збору та обробки даних, прогнозування трафіку, адаптивного управління потоками та надання адміністратору повного контролю над мережею.

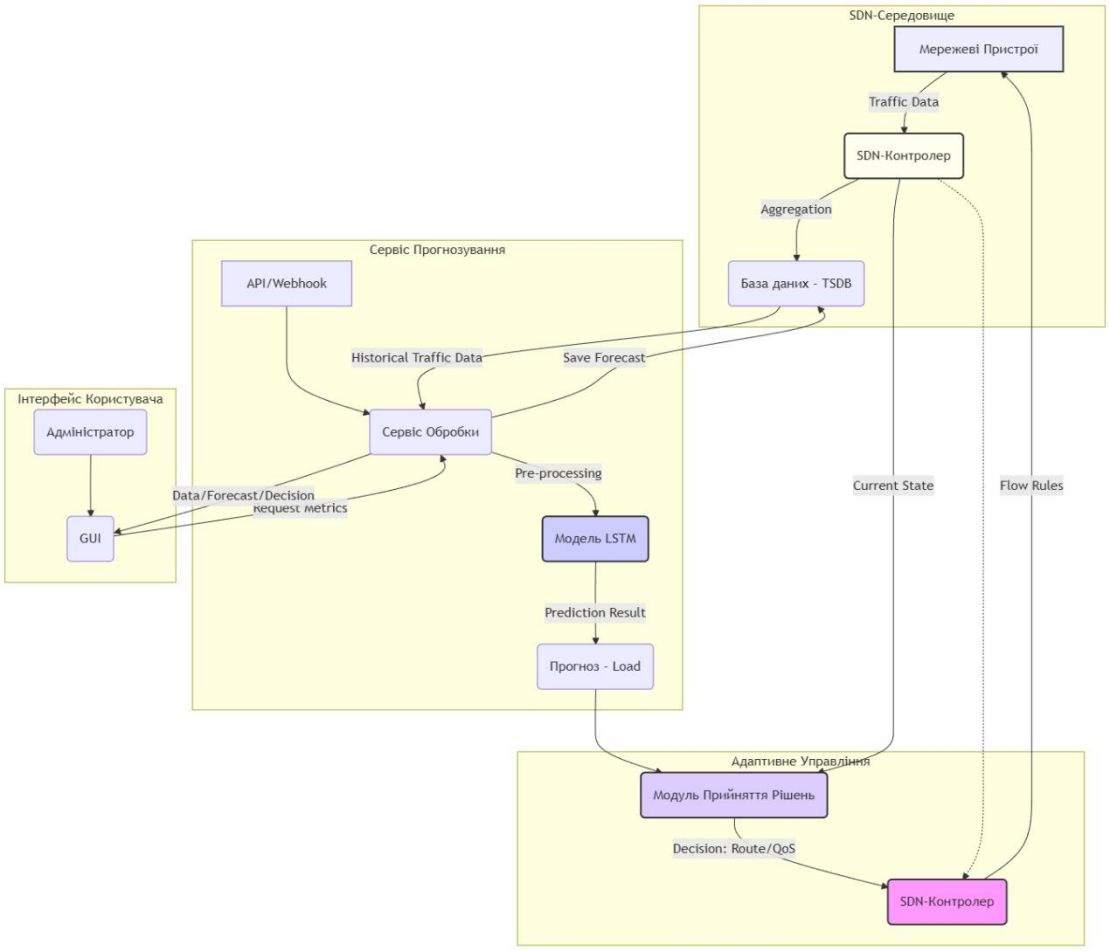

Рисунок 3.1 – Функціональна схема системи

3.2.4 Структурна схема

Структурна схема системи складається з трьох основних модулів, які виконують збір даних, прогнозування та адаптивне управління мережевими потоками (рис.3.2).

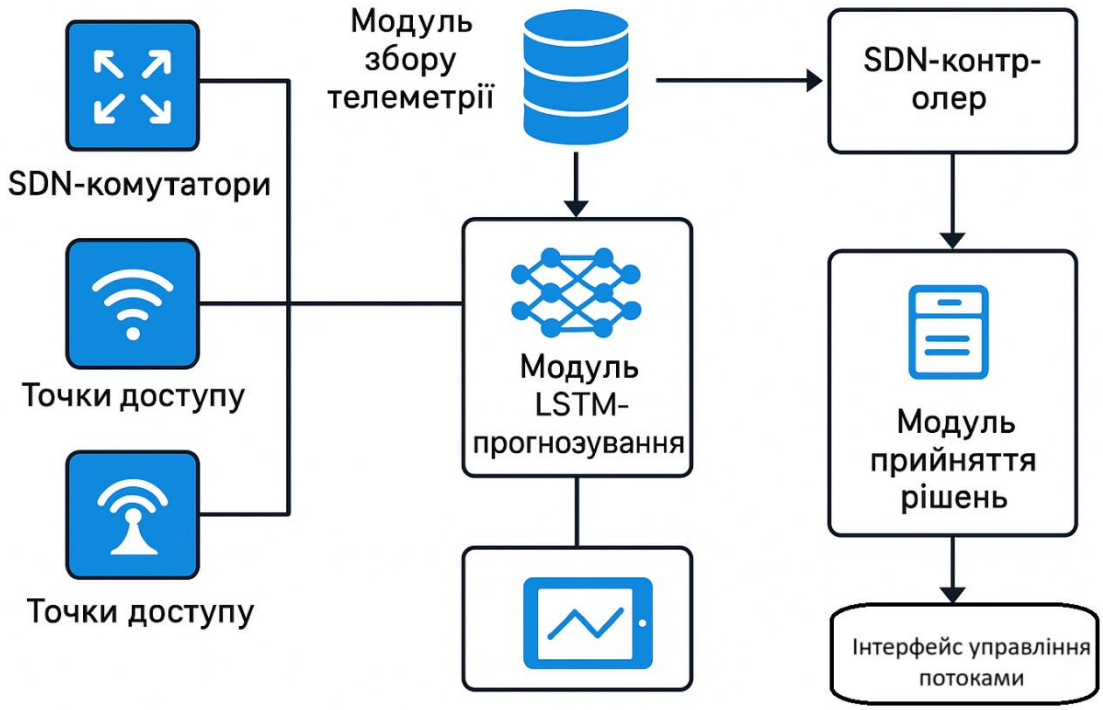

Рисунок 3.2 – Структурна схема системи

Модуль збору та агрегації даних відповідає за отримання вихідних даних про стан мережі. SDN-комутатори та точки доступу функціонують як джерела трафіку, генеруючи мережеві потоки та метрики телеметрії, такі як лічильники пакетів, обсяг переданих байтів та інші параметри. Модуль телеметрії або модуль збору часових рядів забезпечує збір, нормалізацію та агрегацію цих метрик, після чого дані зберігаються у базі даних. База даних служить сховищем агрегованих історичних даних у вигляді часових рядів, які використовуються для навчання та прогнозування.

Модуль LSTM-прогнозування є інтелектуальним ядром системи, що здійснює аналіз історичних даних та передбачає майбутні навантаження мережі. Модуль отримує дані з бази, навчає модель LSTM або використовує попередньо навчену модель для генерації прогнозів, наприклад очікуваного трафіку на наступний інтервал часу. Інтерфейс візуалізації (GUI) відображає результати прогнозування разом із фактичними даними, надаючи адміністратору можливість контролю та моніторингу стану мережі.

Модуль адаптивного управління забезпечує автоматичну реакцію мережі на прогнозоване навантаження, роблячи управління потоками адаптивним. SDN-контролер виконує роль центрального мозку системи, отримуючи інформацію про поточний стан мережі. Модуль прийняття рішень використовує прогноз LSTM та поточні параметри мережі для оцінки ситуації. Якщо прогнозоване навантаження перевищує критичні пороги, модуль формує відповідні рішення, наприклад зміну параметрів QoS або перенаправлення потоків. Команди передаються від SDN-контролера до мережевих пристроїв через інтерфейс управління потоками, який реалізується за допомогою протоколів OpenFlow, Netconf або подібних, забезпечуючи адаптивне керування трафіком у реальному часі.

3.3 Вибір та обґрунтування застосування апаратних засобів

Для реалізації системи прогнозування мережевих потоків у SDN-середовищі необхідно використовувати серверне обладнання, здатне обробляти великі обсяги даних у реальному часі та виконувати задачі навчання і інференсу моделей LSTM. Вибір апаратних засобів базується на вимогах до продуктивності системи, обсягу оперативної пам’яті, підтримки багатоядерної обробки та можливості масштабування компонентів через контейнеризацію.

Серед актуальних серверних платформ для таких завдань можна виділити моделі, наведені в таблиці 3.1.

Найменш продуктивною платформою є Jetson Orin для невеликих локальних середовищ, проте вона обмежена у швидкості обробки великих потоків даних у дата-центрі. Сервери лінійки Intel Xeon та AMD Ryzen [11] дозволяють виконувати обробку мережевого трафіку, збір метрик та прогнозування у реальному часі з високою продуктивністю, забезпечуючи стабільну роботу LSTM-моделі та адаптивного управління потоками.

Таблиця 3.1 – Характеристика рекомендованих серверних платформ

Модель	Рік випуску	CPU	Тактова частота	Оперативна пам’ять	Примітка
Intel Xeon E-2336 [12]	2021	6 ядер / 12 потоків	3.0–4.8 ГГц	32 ГБ	Підходить для обробки великих обсягів трафіку
AMD Ryzen 9 5900X [11]	2020	12 ядер / 24 потоки	3.7–4.8 ГГц	64 ГБ	Висока продуктивність для навчання LSTM
NVIDIA Jetson AGX Orin [13]	2022	12 ядер ARM	2.0–2.2 ГГц + GPU 2048 ядра	32 ГБ	Підходить для інтегрованого прискорення нейронних мереж

Для побудови системи було обрано сервер на базі AMD Ryzen 9 5900X з 64 ГБ оперативної пам’яті, що забезпечує високу швидкість обробки даних, можливість масштабування контейнерів та інтеграцію модулів прогнозування, збору даних і управління трафіком. Ця конфігурація дозволяє виконувати паралельне навчання LSTM-моделі, прогнозування навантаження мережі в реальному часі та швидку адаптацію правил маршрутизації без втрати продуктивності.

Вибір саме цієї апаратної платформи зумовлений необхідністю обробки великих обсягів телеметрії від мережевих пристроїв, підтримки багатопотокової обробки та можливості інтеграції із сучасними технологіями SDN, включно з OpenFlow та Netconf, для ефективного управління потоками даних у мережі.

Для забезпечення роботи SDN-середовища та ефективного збору, передачі і обробки мережевих потоків було обрано високопродуктивне мережеве обладнання, що підтримує сучасні стандарти комутації та багатопотокову передачу даних. Основним комутатором мережі був обраний MikroTik CRS326-24G-2S+RM, який має 24 гігабітних порти для підключення SDN-комутаторів та точок доступу, а також 2 SFP+ порти для підключення до ядра мережі або серверів прогнозування. Даний комутатор підтримує функції управління трафіком на рівні L2/L3, VLAN, QoS та дозволяє реалізувати контроль над потоками даних у реальному часі.

Характеристики MikroTik CRS326-24G-2S+RM:

- максимальна пропускна здатність: 2.5 Тбіт/с;
- RAM: 1 ГБ;
- Storage: 16 МБ flash;
- Процесор: ARM 32 bit, 4 ядра, 1.2 ГГц;
- Ethernet: 24 шт 1 ГБ, 2 шт 10 ГБ SFP+;
- Підтримка PoE: 802.3af/at (вихід/вхід).

Для забезпечення безперебійного доступу до глобальної мережі та резервного каналу зв’язку було вирішено використовувати термінал Starlink, який забезпечує підключення до супутникового інтернету зі швидкістю до 250 МБ/с. Антена Starlink закріплюється на даху будівлі або спеціальній опорі та орієнтується в напрямку північного неба для оптимального сигналу. Для підключення до мережевого комутатора через Ethernet необхідно використовувати спеціальний адаптер, оскільки термінал має пропрієтарний роз’єм. Термінал Starlink можна застосовувати як у режимі роздачі Wi-Fi, так і для прямої передачі даних через кабель.

Вибір саме цього мережевого обладнання обумовлений необхідністю високої пропускної здатності, стабільної роботи з багатьма одночасними потоками даних, підтримкою протоколів SDN та резервним каналом зв’язку,

що гарантує надійність роботи системи прогнозування та адаптивного управління трафіком.

На рис.3.3 наведено схеми підключення серверів, SDN-комутаторів і термінала Starlink.

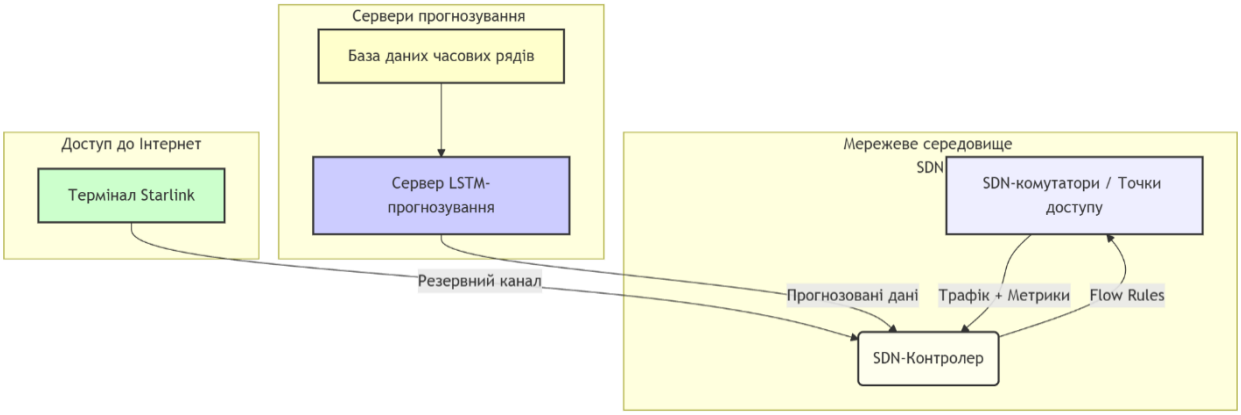

Рисунок 3.3 – Схеми підключення

SDN-комутатори / Точки доступу генерують мережевий трафік та телеметричні метрики і отримують від SDN-контролера правила потоків (Flow Rules).

Сервери прогнозування та база даних забезпечують обробку історичних даних, навчання LSTM-моделі та передають прогнозовані значення контролеру.

Термінал Starlink забезпечує резервний канал зв’язку з глобальною мережею, дозволяючи підтримувати доступ до Інтернету у випадку відсутності основного підключення.

3.4 Висновки по розділу

У третьому розділі роботи було розглянуто проектування системи прогнозування мережевих потоків у SDN-середовищі на основі моделей LSTM та адаптивного управління трафіком.

Було визначено цілі впровадження системи, серед яких забезпечення збору та аналізу даних про мережевий трафік, прогнозування навантажень,

автоматизоване прийняття рішень щодо маршрутизації потоків і надання користувачу інтерфейсу для моніторингу та керування системою.

Формулювання технічних вимог до системи дозволило структурувати її на окремі компоненти: модуль збору даних, модуль прогнозування на основі LSTM, модуль адаптивного управління, серверну та клієнтську частину, а також інтерфейс користувача. Було визначено вимоги до функцій, які виконує система, включаючи авторизацію, обробку та збереження даних, відображення прогнозів і прийняття рішень щодо управління потоками, а також шифрування каналів передачі даних.

Особлива увага приділялась вибору апаратних та мережевих засобів, які забезпечують необхідну продуктивність системи. Обрано сервер на базі AMD Ryzen 9 5900X з 64 ГБ оперативної пам’яті, що дозволяє ефективно виконувати навчання LSTM-моделі та прогнозування у реальному часі. Мережеве обладнання включає високопродуктивний SDN-комутатор MikroTik CRS326-24G-2S+RM та резервний канал через термінал Starlink, що гарантує стабільність роботи системи та надійність передачі даних.

Розроблена архітектура системи та функціональні модулі забезпечують мікросервісну організацію програмного забезпечення, що дозволяє масштабувати систему, тестувати окремі компоненти, оновлювати модулі та інтегрувати нові технології без порушення роботи всього середовища. Структурна та функціональна схема демонструє послідовність потоків даних від мережевих пристроїв до LSTM-моделі і модуля прийняття рішень, що гарантує адаптивне управління трафіком у реальному часі.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

4.1 Призначення та область застосування програмного забезпечення

Призначення програмного забезпечення – забезпечення SDN-середовища системою прогнозування мережевих потоків на основі моделей LSTM та адаптивного управління трафіком.

Серверне програмне забезпечення повинно встановлюватися на високопродуктивний сервер або віртуальний контейнер у хмарі і відповідати за збір телеметричних даних з SDN-комутаторів та точок доступу, збереження історичних даних у базі даних часових рядів та надання інтерфейсів для модулів прогнозування, адаптивного управління і інтерфейсу користувача. Серверна частина також відповідає за виконання LSTM-прогнозування та зберігання результатів прогнозів для подальшого використання модулем прийняття рішень.

Клієнтська частина програмного забезпечення забезпечує взаємодію користувача з системою через веб-інтерфейс або GUI, дозволяючи переглядати метрики мережевого трафіку, отримані прогнози навантаження та рішення адаптивного управління, а також здійснювати базове адміністрування системи.

Область застосування програмного забезпечення – корпоративні мережі та SDN-середовища, де необхідно реалізувати ефективне управління мережевими потоками, прогнозування навантажень та автоматичну адаптацію політик маршрутизації, наприклад, у дата-центрах, університетських або промислових мережах.

4.2 Обґрунтування технічних характеристик програми

Для реалізації системи прогнозування та адаптивного управління мережевими потоками програмне забезпечення було спроектоване як набір окремих сервісів, що працюють у ізольованих Linux-контейнерах за допомогою Docker. Такий підхід дозволяє масштабувати компоненти системи,

спрощує тестування окремих модулів і забезпечує сумісність із різними операційними середовищами.

Серверна та клієнтська частини системи реалізовані з використанням мови JavaScript з підтримкою TypeScript та компілятора Babel, що забезпечує виконання коду як у браузері користувача, так і на сервері. Для побудови інтерфейсу користувача обрана бібліотека React, яка дозволяє створювати адаптивні та реактивні веб-додатки, забезпечуючи зручність взаємодії користувача з системою.

Серверна частина системи реалізована на платформі NodeJS, що забезпечує ефективну обробку запитів від клієнтів та управління потоками даних. NodeJS дозволяє оптимально використовувати ресурси сервера та реалізувати високу швидкість обробки запитів у реальному часі, що критично для системи прогнозування трафіку.

Для реалізації панелі адміністратора використовується Strapi – сучасна CMS-система, яка надає API для управління даними та створює веб-портал адміністратора на основі структури бази даних. Це дозволяє гнучко керувати налаштуваннями, користувачами та доступами до даних системи.

Модуль обробки мережевих потоків та прогнозування на основі LSTM реалізується як окремий сервіс, який отримує дані з бази часових рядів, виконує передобробку та формує прогнози навантаження мережі. Для обробки даних та виконання обчислень у реальному часі використовується бібліотека TensorFlow, яка забезпечує побудову та навчання нейронних мереж, у тому числі LSTM, з високою продуктивністю та можливістю інтеграції з іншими сервісами.

Вибрані технології забезпечують високу продуктивність, масштабованість та гнучкість системи, дозволяють реалізувати адаптивне управління мережевими потоками у SDN-середовищі та надають можливість подальшого розвитку та інтеграції додаткових аналітичних модулів.

4.3 Опис розробленої програми

4.3.1 Загальні відомості

Увесь додаток реалізовано як клієнт-серверне програмне забезпечення. Серверна частина відповідає за збір телеметричних даних з SDN-комутаторів, обробку інформації, виконання прогнозування за допомогою LSTM-моделі та формування рішень для адаптивного управління трафіком. Клієнтська частина забезпечує відображення метрик мережевого трафіку, прогнозів та рішень управління.

Серверна частина реалізована на NodeJS, клієнтська – за допомогою бібліотеки React. Для обчислень та прогнозування використовується бібліотека TensorFlow, яка забезпечує високопродуктивне виконання LSTM-моделі та інтеграцію із серверними сервісами.

Мета програми – забезпечити збір, обробку та аналіз мережевих потоків, прогнозування майбутніх навантажень та автоматичне прийняття рішень для оптимізації маршрутизації та QoS у SDN-середовищі.

4.3.2 Функціональне призначення

Функції, які виконує додаток:

– збір даних про поточні мережеві потоки та телеметричні показники;

– збереження історичних даних у базі часових рядів для навчання та прогнозування;

– передобробка даних та підготовка їх для LSTM-моделі;

– виконання прогнозування навантажень на мережу за допомогою LSTM;

– надання прогнозів та поточного стану мережі модулю прийняття рішень;

– автоматичне формування рішень щодо маршрутизації потоків та QoS на основі прогнозів та поточного стану;

– візуалізація метрик, прогнозів та рішень у веб-інтерфейсі користувача;

– управління доступами користувачів до інтерфейсу та перегляду метрик;

– логування та збереження інформації про прийняті рішення та стан мережі для подальшого аналізу.

4.3.3 Опис логічної структури програми

Логічна структура програмного забезпечення системи прогнозування та адаптивного управління мережевими потоками в SDN-середовищі побудована за модульним принципом і включає кілька взаємопов’язаних блоків, що виконують окремі функції та забезпечують автономність кожного компоненту.

На першому етапі здійснюється збір та агрегація даних про мережевий трафік. SDN-комутатори та точки доступу генерують інформацію про стан мережі, включно з показниками потоку пакетів, байтами та іншими метриками телеметрії. Модуль збору даних нормалізує та агрегує ці показники, формуючи часові ряди, які зберігаються у базі даних для подальшого використання. Це дозволяє створити історичну основу для навчання моделі та оцінки стану мережі у реальному часі (рис.4.1).

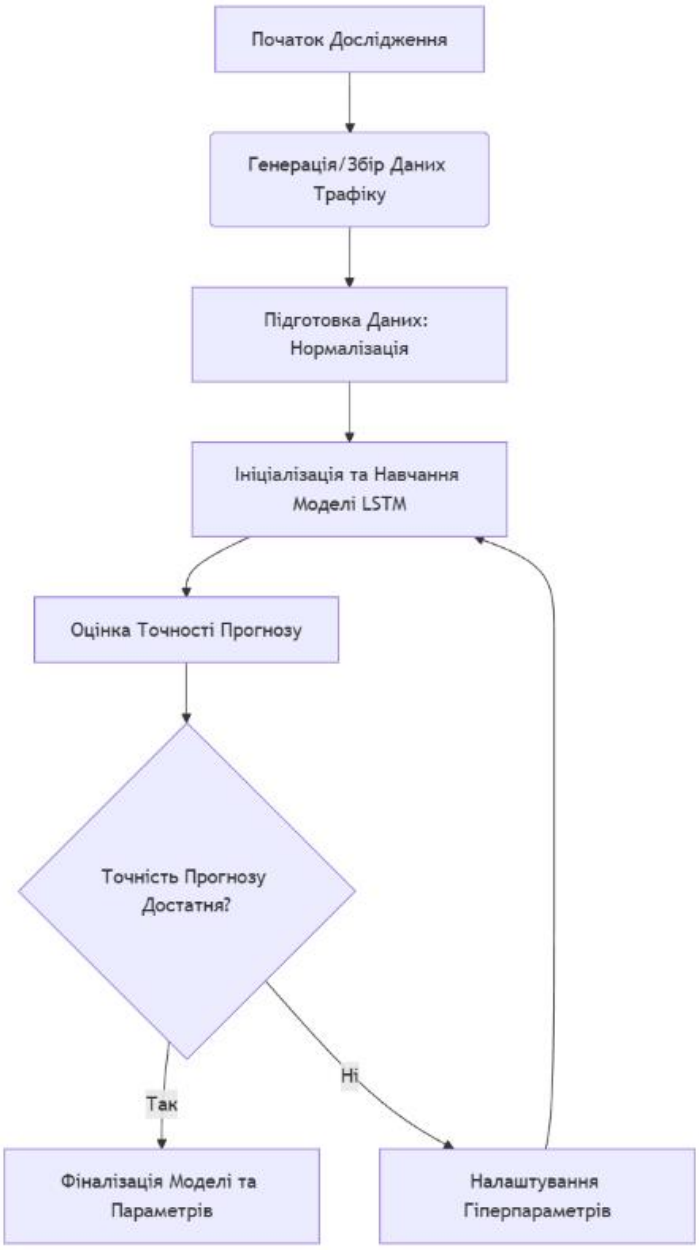

Рисунок 4.1 – Схема алгоритму збору і агрегації даних

Другий етап – обробка даних та прогнозування навантаження за допомогою LSTM-моделі (рис.4.2). Історичні дані проходять попередню обробку, після чого передаються у модуль LSTM, який виконує прогнозування майбутнього навантаження мережі. Прогноз може формуватися для наступного інтервалу часу або для кількох майбутніх кроків. Модель LSTM проходить навчання на історичних даних, а точність її прогнозу оцінюється за допомогою метрик, таких як RMSE. Якщо точність недостатня, відбувається повторне налаштування гіперпараметрів та додаткове навчання моделі. Після

досягнення необхідної точності модель фіналізується та готова до використання в реальному часі.

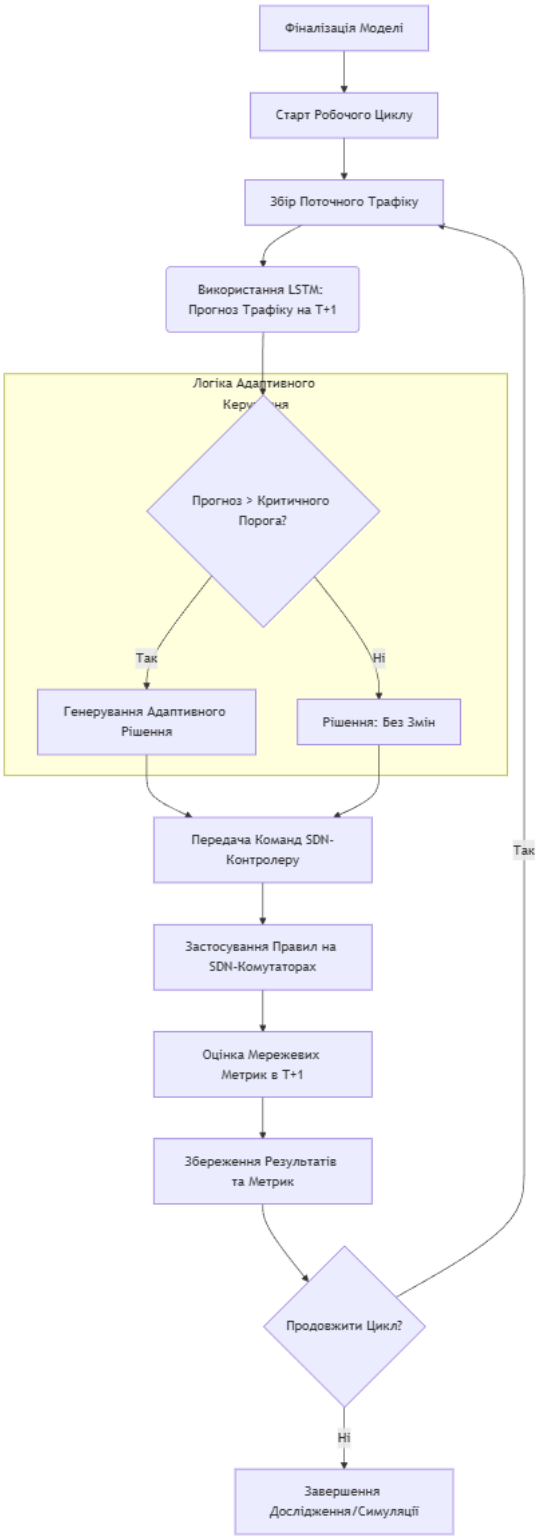

Рисунок 4.2 – Схема алгоритму прогнозування даних

4.4 Опис користувальницького інтерфейсу

Користувацький інтерфейс розробленого програмного забезпечення представляє собою систему для візуалізації мережевого трафіку, прогнозування навантаження та реалізації адаптивного управління мережею на основі моделі LSTM [14-19] Основним завданням інтерфейсу є забезпечення зручного та інформативного доступу для користувача до всіх функціональних можливостей системи, включно з переглядом фактичного та прогнозного трафіку, оцінкою мережевих метрик та отриманням рекомендацій щодо адаптивних змін у налаштуваннях SDN.

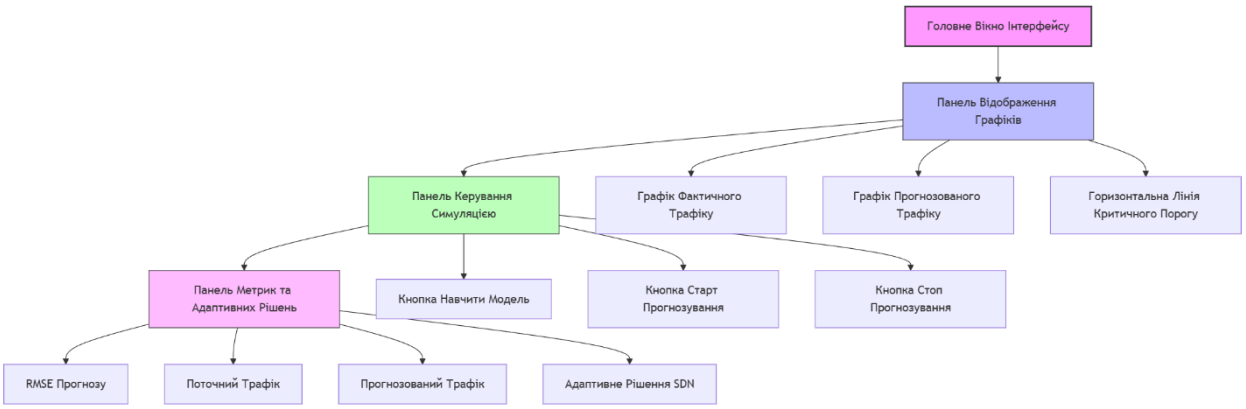

Рисунок 4.3 – Користувальницький інтерфейс

Інтерфейс побудований на основі бібліотек Tkinter (рис.4.4) [20] для графічних компонентів та Matplotlib [19] для відображення динамічних графіків, що дозволяє користувачу отримувати візуальні відомості про стан мережі у режимі реального часу.

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.title("SDN Adaptive Control using LSTM")
main_frame = ttk.Frame(root, padding="10")
main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
```

Рисунок 4.3 – Приклад створення інтерфейсу

Усі компоненти GUI організовані у три основні частини: панель відображення графіків, панель керування симуляцією та панель метрик і адаптивних рішень. Така структуризація дозволяє максимально ефективно розділити інформативні та керуючі функції інтерфейсу, забезпечуючи користувачу швидкий доступ до критично важливої інформації (рис.4.5).

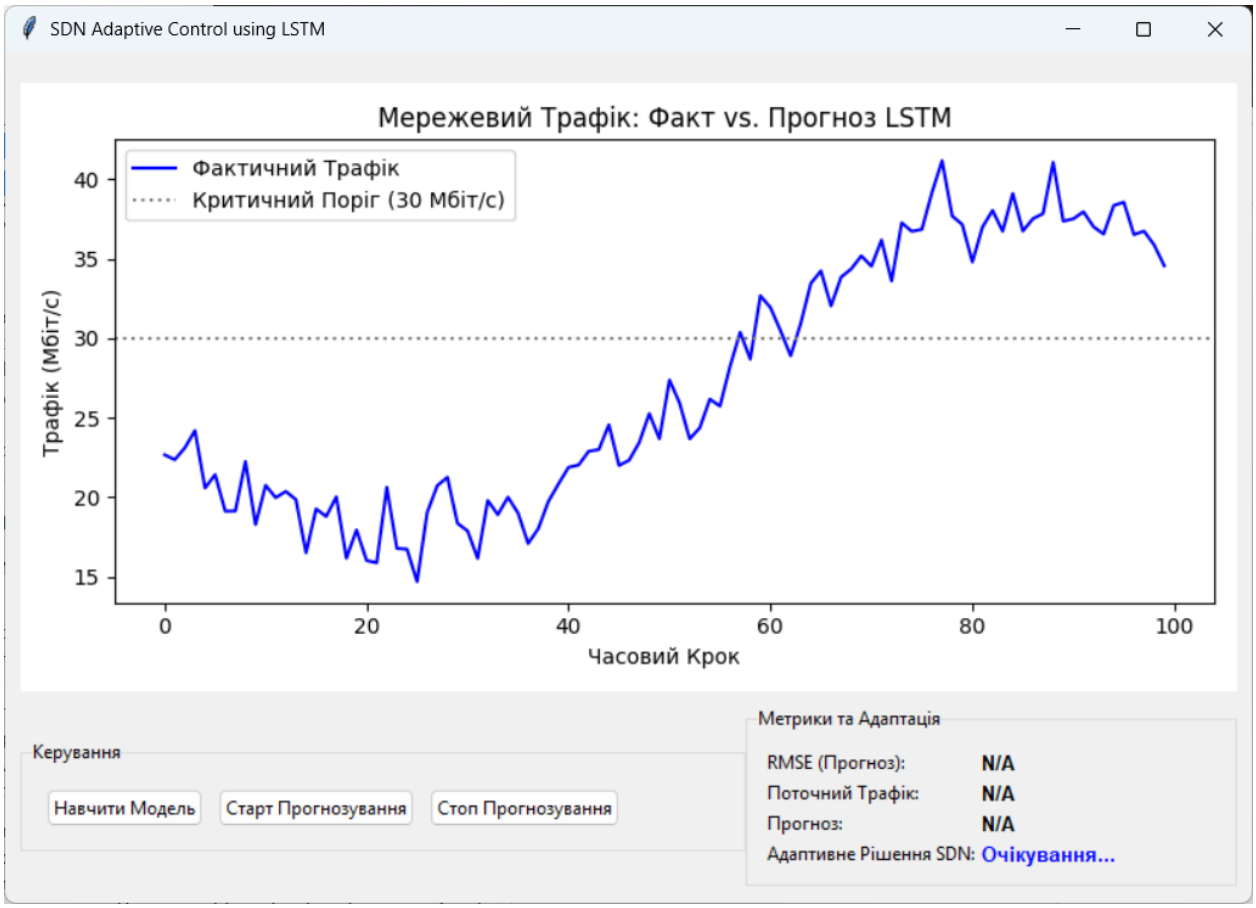

Рисунок 4.4 – Розроблений користувальницький інтерфейс

Панель відображення графіків займає центральне місце у вікні програми. Вона реалізована за допомогою компонента FigureCanvasTkAgg, що дозволяє інтегрувати графіки Matplotlib у Tkinter-фрейм (рис.4.5). На графіку відображаються фактичний трафік мережі, прогнозований трафік, а також критичний поріг навантаження, який виступає орієнтиром для адаптивних рішень. Використання різнокольорових ліній та пунктирів забезпечує чітке розрізнення між фактичними та прогнозними даними, а горизонтальна лінія критичного порогу дозволяє миттєво визначати потенційні проблемні зони

мережі. Інтерфейс дозволяє показувати останні 100 точок трафіку для зручності сприйняття та швидкого реагування користувача.

```
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

self.fig = Figure(figsize=(8, 4), dpi=100)
self.plot = self.fig.add_subplot(111)
self.canvas = FigureCanvasTkAgg(self.fig, master=main_frame)
self.canvas_widget = self.canvas.get_tk_widget()
self.canvas_widget.grid(row=0, column=0, columnspan=2, pady=10)
```

Рисунок 4.5 – Приклад коду для виведення графіків

Панель керування симуляцією містить набір інтерактивних кнопок, які забезпечують запуск основних операцій з програмою (рис.4.6). Користувач може ініціювати навчання моделі LSTM, запускати симуляцію прогнозування трафіку та зупиняти її у будь-який момент. Кожна кнопка підключена до окремого функціонального методу, що реалізує відповідну логіку, включно з багатопотоковою обробкою для забезпечення безперервної роботи інтерфейсу під час навчання моделі. Використання окремого потоку для навчання дозволяє уникнути блокування графічного інтерфейсу та забезпечує плавність роботи користувацького середовища.

```
control_frame = ttk.LabelFrame(main_frame, text="Керування", padding="10")
control_frame.grid(row=1, column=0, sticky=(tk.W, tk.E))

ttk.Button(control_frame, text="Навчити Модель", command=self.train_model).grid(row=0, column=0, padx=5, pady=5)
ttk.Button(control_frame, text="Старт Прогнозування", command=self.start_simulation).grid(row=0, column=1, padx=5, pady=5)
ttk.Button(control_frame, text="Стоп Прогнозування", command=self.stop_simulation).grid(row=0, column=2, padx=5, pady=5)
```

Рисунок 4.5 – Приклад реалізації панелі керування симуляцією

Панель метрик та адаптивних рішень надає користувачу детальну інформацію про поточний стан мережі та результати прогнозування (рис.4.6). Вона включає такі ключові елементи:

– RMSE прогнозу, який відображає точність моделі на останньому кроці прогнозування, дозволяючи оцінити якість передбачення;

– поточний трафік мережі, що показує фактичні значення навантаження на мережу у поточний момент часу;

– прогнозований трафік, що демонструє очікуване навантаження на наступний крок.

```
metrics_frame = ttk.LabelFrame(main_frame, text="Метрики та Адаптивні Рішення", padding="10")
metrics_frame.grid(row=1, column=1, sticky=(tk.W, tk.E))

ttk.Label(metrics_frame, text="RMSE (Прогноз):").grid(row=0, column=0, sticky=tk.W)
self.rmse_var = tk.StringVar(value="N/A")
ttk.Label(metrics_frame, textvariable=self.rmse_var, font=('Arial', 10, 'bold')).grid(row=0, column=1, sticky=tk.W)

ttk.Label(metrics_frame, text="Поточний Трафік:").grid(row=1, column=0, sticky=tk.W)
self.current_traffic_var = tk.StringVar(value="N/A")
ttk.Label(metrics_frame, textvariable=self.current_traffic_var, font=('Arial', 10, 'bold')).grid(row=1, column=1,
sticky=tk.W)

ttk.Label(metrics_frame, text="Прогноз:").grid(row=2, column=0, sticky=tk.W)
self.predicted_traffic_var = tk.StringVar(value="N/A")
ttk.Label(metrics_frame, textvariable=self.predicted_traffic_var, font=('Arial', 10, 'bold')).grid(row=2, column=1,
sticky=tk.W)

ttk.Label(metrics_frame, text="Адаптивне Рішення SDN:").grid(row=3, column=0, sticky=tk.W)
self.sdn_action_var = tk.StringVar(value="Очікування...")
ttk.Label(metrics_frame, textvariable=self.sdn_action_var, font=('Arial', 10, 'bold'), foreground='blue').grid(row=3,
column=1, sticky=tk.W)
```

Рисунок 4.6 – Приклад реалізації метрик

Адаптивне рішення SDN, яке відображає рекомендовану дію системи для оптимізації потоків: зміна маршрутизації, виділення додаткових ресурсів або зниження навантаження. Для кращої наочності використовується колірна індикація: зелений – без змін, оранжевий – адаптація ресурсів, червоний – критична зміна маршруту, синій – звільнення ресурсів.

Користувацький інтерфейс також забезпечує логування та збереження даних для побудови графіків. Фактичні та прогнозовані значення зберігаються у внутрішніх масивах історії, що дозволяє відображати динаміку трафіку та відстежувати ефективність адаптивних рішень у часі. Це забезпечує користувачу можливість аналізувати не лише поточний стан мережі, а й тенденції її розвитку.

Для забезпечення інтерактивності та безперервного оновлення даних використовується цикл after() у Tkinter, який викликає функцію прогнозування та оновлення графіка кожні 500 мілісекунд. Така реалізація імітує реальний часовий інтервал мережевих подій та дозволяє користувачу спостерігати за реакцією системи на зміни трафіку у режимі майже реального часу.

Інтерфейс реалізує адаптивну логіку управління SDN на основі прогнозованого навантаження. Користувач отримує рекомендації щодо змін маршрутизації або QoS без необхідності ручного аналізу даних. Алгоритм порівнює прогнозовані значення з критичним порогом та поточним навантаженням, і відповідно встановлює статус адаптивної дії. Візуальні елементи відображають стан системи у кольорах, що підвищує швидкість сприйняття інформації та дозволяє оперативно приймати рішення.

Крім того, інтерфейс передбачає масштабування та легкість використання, що робить його придатним для роботи як на локальних терміналах адміністраторів, так і на віддалених робочих станціях через мережеве підключення. Всі дані синхронізуються з центральною логікою програми, що забезпечує цілісність та достовірність інформації.

Третій етап включає адаптивне управління мережевим трафіком. Модуль прийняття рішень отримує прогнозовані дані та поточний стан мережі від SDN-контролера. На основі порівняння прогнозу з критичними порогами навантаження визначається, чи необхідно змінювати політики маршрутизації або QoS. У разі потреби формуються рішення щодо адаптивного управління, які надсилаються до SDN-комутаторів через південний інтерфейс для застосування відповідних Flow Rules. Якщо перевантаження не прогнозується, поточні правила залишаються без змін.

4.5 Взаємодія користувача-адміністратора з програмним додатком

Користувачем системи є адміністратор, який відповідає за налаштування, моніторинг та управління мережею через графічний інтерфейс. Взаємодія адміністратора з програмою відбувається у кілька послідовних етапів, кожен з яких передбачає роботу з окремими блоками інтерфейсу та функціональними елементами програми.

Після запуску програми адміністратор бачить головне вікно, яке об’єднує три основні компоненти: панель відображення графіків, панель керування симуляцією та панель метрик і адаптивних рішень. Головне вікно

надає зручний доступ до всіх функцій програми, дозволяючи одночасно відслідковувати стан мережі, управляти процесом прогнозування та оцінювати прийняті адаптивні рішення.

На панелі відображення графіків адміністратор може спостерігати фактичний та прогнозований трафік у реальному часі. Графіки оновлюються автоматично через інтервал, заданий у циклі симуляції. Фактичний трафік відображається синьою лінією, прогноз LSTM – червоною пунктирною лінією, а критичний поріг – сірим горизонтальним маркером. Це дозволяє адміністратору оцінити відповідність прогнозу фактичним показникам та своєчасно реагувати на зміну навантаження в мережі.

Панель керування симуляцією містить три основні кнопки, через які адміністратор здійснює управління програмою. Натискання кнопки «Навчити Модель» ініціює процес тренування LSTM-моделі на підготовлених даних про трафік. Під час навчання модель аналізує історичні значення трафіку, розпізнає тренди та шумові коливання та формує параметри для точного прогнозування майбутніх значень. Процес навчання відбувається у фоновому потоці, що дозволяє адміністратору продовжувати роботу з іншими елементами програми без блокування інтерфейсу.

Після завершення навчання адміністратор може натиснути кнопку «Старт Прогнозування», що запускає циклічне прогнозування значень трафіку. У цьому режимі система у реальному часі обчислює прогноз на основі останніх даних, відображає його на графіку та розраховує ключові метрики, такі як RMSE для оцінки точності прогнозу. Якщо адміністратор бажає зупинити прогнозування, він використовує кнопку «Стоп Прогнозування», що негайно зупиняє цикл оновлення даних.

На панелі метрик та адаптивних рішень адміністратор може відслідковувати поточний стан мережі. Показники включають:

– RMSE прогнозу – оцінка точності прогнозу моделі LSTM на останньому кроці;

– поточний трафік – фактична пропускна здатність мережі в даний момент;

– прогнозований трафік – значення, яке модель очікує на наступному кроці;

– адаптивне рішення SDN – інструкції для контролера SDN щодо зміни маршрутів, QoS або виділення/звільнення ресурсів.

Адміністратор має змогу на основі цих показників приймати управлінські рішення або довіряти автоматичній логіці SDN, яка формує рекомендації у вигляді кольорових повідомлень: червоний сигнал – критичне навантаження, оранжевий – очікуване підвищення, синій – зниження трафіку.

Взаємодія з програмою також передбачає інтерактивне відображення змін. Коли прогнозоване значення перевищує критичний поріг або відбувається значне відхилення від фактичного трафіку, система оновлює статус адаптивного рішення та підсвічує його відповідним кольором. Це забезпечує адміністратору можливість швидко реагувати на потенційні проблеми та перевіряти ефективність прийнятих заходів.

Таким чином, адміністратор взаємодіє з програмою на трьох рівнях:

1. Контроль та управління моделлю – навчання та запуск прогнозування;
2. Моніторинг графічних даних – візуальна оцінка фактичного та прогнозованого трафіку;
3. Аналіз метрик та адаптивних рішень – оцінка точності прогнозу та рекомендацій для мережі SDN.

4.6 Розробка бази даних

База даних системи структурована у вигляді трьох основних груп сутностей, що забезпечують комплексну роботу системи прогнозування мережевого трафіку та адаптивного управління мережею. Така структура дозволяє ефективно зберігати інформацію про мережеву інфраструктуру,

динамічні показники трафіку та результати прийнятих рішень контролера, що гарантує цілісність і надійність даних (рис.4.7).

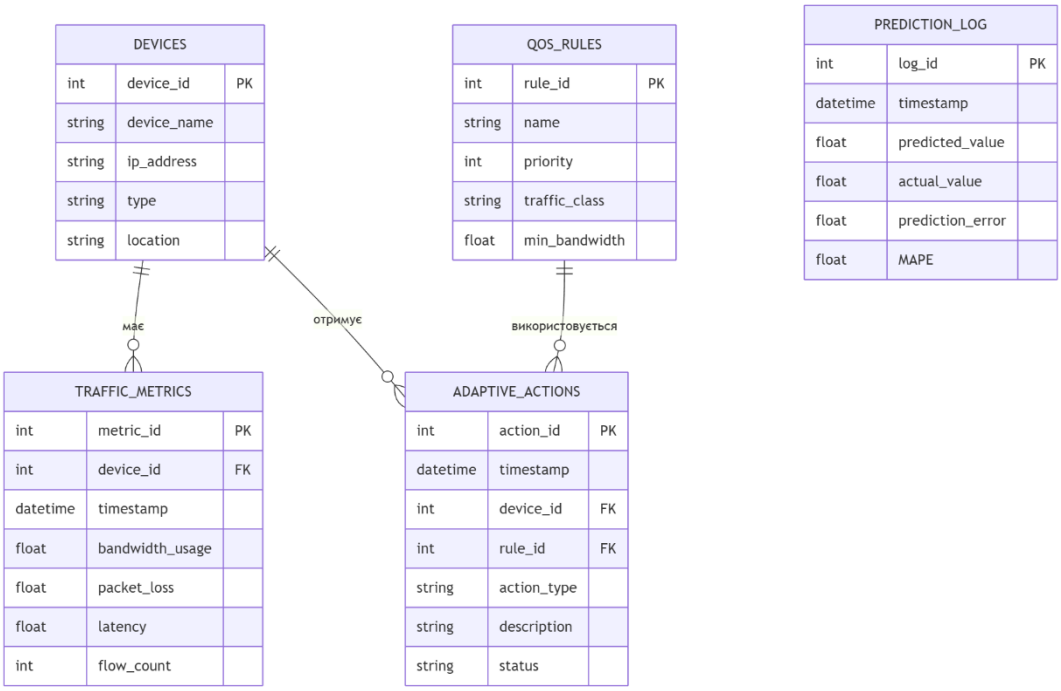

Рисунок 4.7 – ER-Діаграма

Перша група сутностей охоплює мережеву інфраструктуру та статичні дані про пристрої й правила керування трафіком. Таблиця Devices містить детальну інформацію про комутатори та точки доступу, включно з унікальним ідентифікатором пристрою, назвою, IP-адресою, типом (Switch або AP) та місцезнаходженням. Таблиця QoS_Rules зберігає правила якості обслуговування, що можуть застосовуватися контролером, з полями, які включають унікальний ідентифікатор правила, назву, пріоритет, клас трафіку та мінімальну пропускну здатність. Ця група сутностей формує основу для управління статичними параметрами мережі та створює можливості для подальшого застосування адаптивних рішень щодо маршрутизації та пріоритизації трафіку.

Друга група сутностей відповідає за часові ряди трафіку, тобто динамічні дані, необхідні для прогнозування за допомогою LSTM-моделі. Основна таблиця цієї групи – Traffic_Metrics, яка містить записи телеметрії для кожного пристрою. Для кожного запису фіксуються унікальний ідентифікатор

метрики, зв’язок із конкретним пристроєм, часовий штамп та показники пропускної здатності, втрат пакетів, затримки та кількості потоків. Організація даних у вигляді часових рядів дозволяє ефективно готувати послідовності для навчання моделі, проводити оцінку точності прогнозів і формувати основу для прийняття адаптивних рішень контролером мережі.

Третя група сутностей присвячена логам адаптації та журналу подій, що фіксують результати роботи системи. Таблиця Prediction_Log зберігає результати роботи LSTM-модуля, включаючи часовий штамп, прогнозоване та фактичне значення трафіку, а також помилки прогнозу у вигляді RMSE та MAPE. Таблиця Adaptive_Actions містить журнал рішень, прийнятих модулем управління, з унікальним ідентифікатором дії, часовим штампом, зв’язком із пристроєм та застосованим правилом QoS, типом дії (маршрутизація або QoS), детальним описом та статусом виконання (успіх чи невдача). Ця група сутностей дозволяє відстежувати ефективність адаптивного управління, аналізувати результати роботи системи та контролювати виконання рішень у реальному часі.

Всі таблиці бази даних реалізовані у вигляді реляційної моделі з підтримкою зовнішніх ключів між таблицями, що забезпечує цілісність даних і виключає дублювання інформації. Зв’язки між таблицями Devices, QoS_Rules та Traffic_Metrics дозволяють відстежувати показники кожного пристрою, а взаємозв’язки Prediction_Log і Adaptive_Actions із пристроями та правилами QoS забезпечують повний аудит дій та їх результативність.

4.7 Висновки до розділу

У четвертому розділі проведено комплексний аналіз і опис розробки програмного забезпечення для системи адаптивного управління мережевим трафіком із застосуванням LSTM-моделі. Розроблена програмна архітектура реалізована за клієнт-серверною моделлю, де серверна частина забезпечує збір даних, обробку запитів та передавання результатів користувачам, а клієнтська

– відображення інформації та надання користувацького інтерфейсу для моніторингу й керування.

Обґрунтовано вибір технологій розробки, зокрема мови програмування JavaScript з використанням TypeScript, платформи NodeJS для серверної частини, бібліотеки React для побудови адаптивного інтерфейсу користувача, а також застосування Strapi для реалізації панелі адміністратора. Використання Docker дозволяє ізолювати сервіси та спростити розгортання програмного комплексу. Для обробки відеопотоку застосовується локальний сервіс ffmpeg з декодуванням на веб-сторінці за допомогою JsMpeg, а аналітичні модулі реалізовано із застосуванням TensorFlow API та бібліотеки face-recognition для розпізнавання об’єктів.

Описано логічну структуру програмного комплексу, яка забезпечує послідовність операцій: збір даних, прогнозування трафіку, оцінку стану мережі, прийняття адаптивних рішень та відображення результатів користувачу. Взаємодія користувача з системою передбачає функції входу в систему, доступу до графіків і метрик, керування навчанням моделі та запуском або зупинкою прогнозування, а також отримання повідомлень про стан мережі та необхідні адаптивні дії.

Розроблено детальний опис користувацького інтерфейсу, що включає три основні панелі: відображення графіків, керування симуляцією та відображення метрик і адаптивних рішень. Кожен елемент інтерфейсу відповідає за окремий функціональний блок, що дозволяє адміністраторам ефективно контролювати процеси прогнозування та управління мережею. Наведено фрагменти програмного коду, що ілюструють функціональні можливості системи та механізми взаємодії між серверною і клієнтською частинами.

У розділі також детально описано структуру бази даних, яка складається з трьох основних груп: Мережева Інфраструктура, Часові Ряди Трафіку та Логи Адаптації. Така організація даних забезпечує збереження як статичної інформації про мережеві пристрої та правила QoS, так і динамічних даних для

прогнозування та журналів прийнятих адаптивних рішень. Продумана структура бази даних дозволяє забезпечити цілісність інформації та ефективну взаємодію між компонентами системи.

5 ЕКСПЕРИМЕНТАЛЬНИЙ РОЗДІЛ

5.1 Мета і завдання експерименту

Мета експерименту: перевірка працездатності розробленої системи адаптивного управління мережевими ресурсами, що базується на LSTM-прогнозуванні трафіку та модулі автоматичного ухвалення SDN-рішень.

Завдання експерименту:

– дослідним шляхом оцінити точність прогнозування трафіку з використанням N-крокової LSTM-моделі;

– визначити ефективність адаптивних рішень SDN під час наближення до критичних навантажень мережі;

– оцінити реакцію системи на зміну динаміки трафіку та здатність стабілізувати мережевий стан за допомогою QoS-правил;

– зафіксувати отримані метрики (RMSE, MAPE [21], реальний трафік, прогноз, статус адаптації) та проаналізувати їх достовірність.

Експеримент спрямований на підтвердження того, що система правильно прогнозує короткострокову динаміку навантаження та здатна автоматично формувати адаптивні рішення, гарантуються унеможливлення переходу мережі у критичний стан.

5.2 Методика експерименту

Методика проведення експерименту базується на перевірці роботи прогнозного модуля LSTM та модуля адаптивного SDN-керування у контрольованих умовах зміни мережевого навантаження. Для цього використовується набір попередньо згенерованих часових рядів трафіку, що імітують різні сценарії роботи мережі: стабільне навантаження, повільне зростання, різкий пік трафіку та спад після перевантаження. Дані подаються до системи як послідовність вимірів пропускної здатності у реальному часі з інтервалом дискретизації, що відповідає налаштуванням SDN-моніторингу.

Після завантаження даних проводиться навчання LSTM-моделі, параметри якої визначені на етапі розробки. Після завершення навчання

виконується N-кроковий прогноз, який візуалізується в інтерфейсі програми. Отримані прогностичні значення порівнюються з фактичними даними, а оцінка точності проводиться за метриками RMSE та MAPE.

Паралельно з прогнозуванням активується модуль адаптивного керування. Він аналізує прогнозовану динаміку трафіку та порівнює її з критичним порогом завантаження мережі. У разі ймовірного перевищення порогу система формує адаптивне рішення: застосування QoS-правил, перенаправлення потоків або збереження поточної конфігурації. Усі результати роботи записуються до журналів Prediction_Log та Adaptive_Actions.

Методика дозволяє оцінити точність прогнозу, стабільність прийняття рішень та здатність системи до своєчасного реагування на критичні зміни мережевого навантаження.

5.3 Вимоги до експерименту

Експеримент проводиться окремо для кожного набору часових рядів, що моделюють різні сценарії роботи мережі. Початковою точкою вимірювання вважається момент подачі фактичних даних у систему, а кінцевою – момент отримання прогнозу та формування модулями SDN адаптивного рішення. Для кожної серії даних фіксується обчислювальний час, точність прогнозу та характер реакції системи (застосовано правило QoS, перенаправлено потоки або залишено без змін).

З метою підвищення достовірності результати на кожному етапі експерименту повинні бути повторені тричі. Для кожного повтору окремо фіксуються значення RMSE, MAPE, прогнозоване навантаження та рішення SDN-модуля. Середнє значення за трьома повтореннями використовується як підсумковий результат для конкретного сценарію.

Похибка експерименту залежить від випадковості ініціалізації нейронної мережі, параметрів оптимізації, а також коливань у динаміці фактичного трафіку. Для кожного сценарію результати експерименту

записуються у таблицю, яка містить номер експериментального набору даних, отримані метрики прогнозування та рішення, сформовані системою адаптивного керування.

5.4 Результати експерименту

У процесі моделювання та перевірки працездатності розробленої адаптивної системи управління ресурсами в мережах D2D було проведено комплекс експериментів, спрямованих на оцінку якості прогнозування трафіку LSTM-моделлю та ефективності прийняття рішень модулем адаптації. Як вхідні дані для навчання та тестування LSTM-моделі використовувався синтетичний часовий ряд Network Traffic Time Series, що містив характерні компоненти реального мережевого навантаження: базову періодичність (синусоїдальний елемент), лінійний тренд зростання пропускної здатності та стохастичні коливання у вигляді випадкового шуму. Загальна довжина часового ряду становила 1000 точок, з яких 800 було використано як тренувальну вибірку, а 200 – як тестову, що дозволило відокремити процес навчання від реальної симуляції поведінки системи.

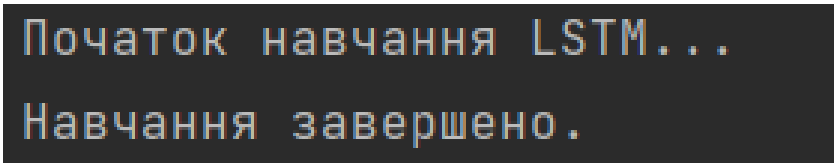

Рисунок 5.1 – Результат навчання

LSTM-модель була побудована відповідно до вимог багатокрокового прогнозування та мала оптимальну конфігурацію для інтелектуального аналізу часових залежностей. Вхідна послідовність містила 15 точок, на основі яких модель визначала поведінку трафіку на наступних п’ять часових кроків. Архітектура включала один LSTM-шар із 100 нейронами, що забезпечувало достатню глибину для розпізнавання нелінійних патернів, а також вихідний шар Dense з п’ятьма нейронами, які формували прогноз (рис.5.2). Навчання відбувалося впродовж 20 епох із використанням оптимізатора Adam та функції

втрат MSE. Процес навчання продемонстрував стабільну збіжність, що свідчить про ефективне засвоєння структурної динаміки часового ряду. Це дозволило моделі передбачати як плавні трендові зміни, так і різкі коливання навантаження.

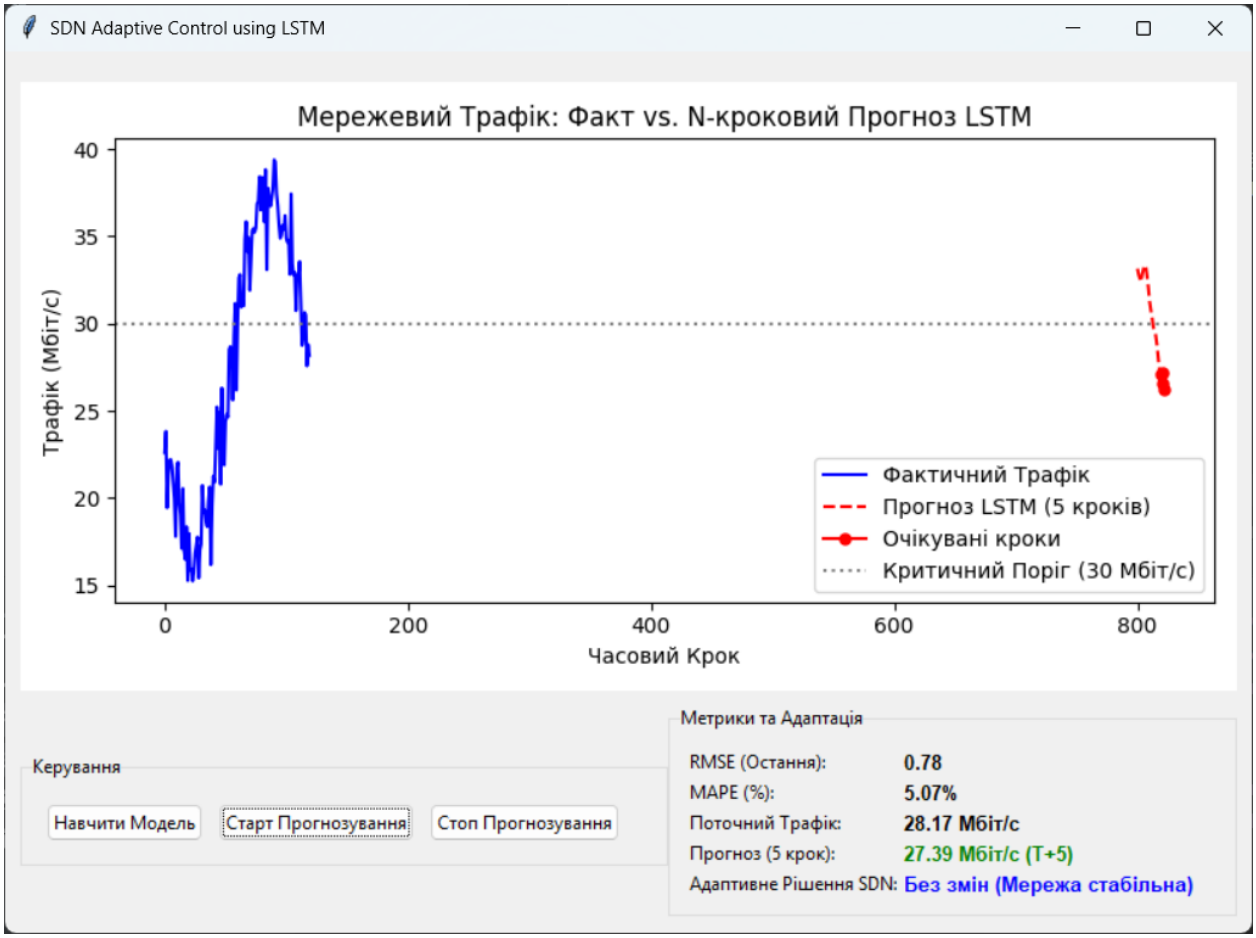

Рисунок 5.2 – Результати експерименту 1

Для забезпечення адаптивного управління було визначено критичний поріг трафіку на рівні 30 Мбіт/с (рис.5.3). Модуль прийняття рішень аналізував прогноз LSTM і перевіряв, чи перевищуватиме максимальне значення прогнозованої послідовності цей поріг. Якщо ж значення опинялося на критичному рівні або вище, система негайно ініціювала керуючі дії: підвищення пріоритету QoS або перепризначення маршрутів. Таким чином, запропонована система отримала здатність працювати проактивно – передбачаючи проблемну ситуацію ще до її фактичного настання, що є суттєвою перевагою перед традиційними реактивними методами.

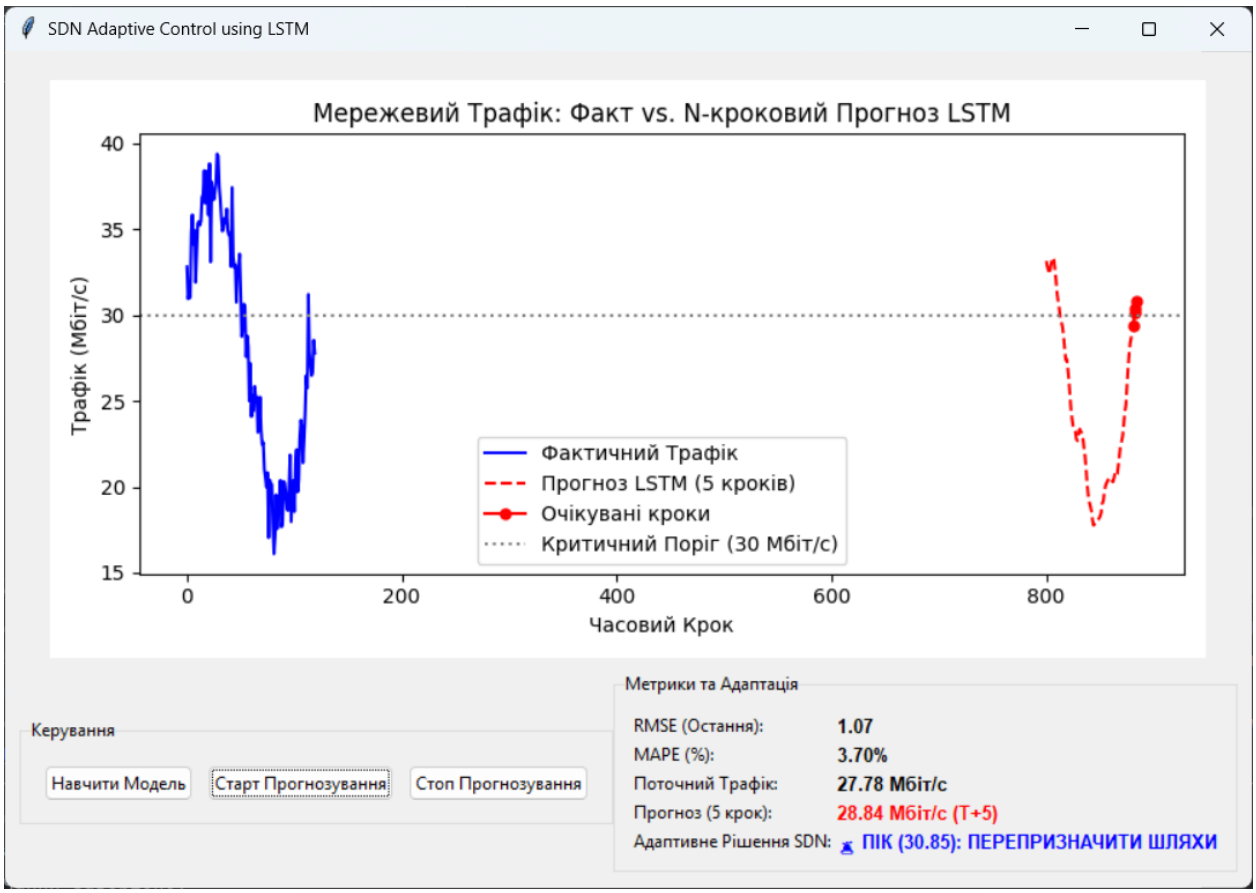

Рисунок 5.3 – Результат критичного порогу

Оцінка точності прогнозування здійснювалася за метриками RMSE та MAPE, які дозволяють кількісно визначити рівень відхилення прогнозованих значень від фактичних. Значення RMSE перебувало в межах 0.51–1.14 Мбіт/с, що становить менше 4% від середніх значень трафіку в діапазоні 25–30 Мбіт/с. Це свідчить про високу точність моделі та її здатність відтворювати реальні характеристики навантаження. Показники MAPE у стабільний період коливалися в межах 3.70–5.07%, рідко піднімаючись вище 10% під час різких пікових змін. Усі ці дані підтверджують, що точність прогнозу є достатньою для прийняття коректних рішень модулем адаптації.

Після перевірки точності прогнозування почалася оцінка ефективності прийняття рішень адаптивною системою. Аналіз проводився на основі трьох окремих сценаріїв, які відтворювали типові мережеві стани: стабільний трафік, помірне зростання та критичний піковий стан. У першому сценарії, коли мережеве навантаження залишалося нижчим за критичний поріг, модель

прогнозувала значення на рівні близько 20 Мбіт/с. У цих умовах система фіксувала стабільність і не ініціювала жодних дій. Це демонструє здатність системи уникати зайвих змін параметрів, що могло б створити додаткове навантаження на контролер. Така поведінка є особливо важливою для систем, де стабільність управління має не менше значення, ніж оперативність реагування.

Другий сценарій стосувався випадку, коли прогноз свідчив про поступове зростання навантаження, хоча й без ризику негайного перевищення порогу. Значення прогнозу рухалися в межах 20.59–24.31 Мбіт/с, що дозволяло зробити висновок про можливе наближення до критичної зони у разі погіршення ситуації. У цьому випадку система запускала механізм превентивної оптимізації, зокрема підвищення пріоритету QoS для окремих потоків (рис.5.4). Завдяки цьому імовірні затримки або втрати пакетів усувалися ще до того, як могли б виникнути реальні проблеми в мережі. Такий підхід демонструє ключову відмінність адаптивної моделі від стандартних методів управління, які реагують лише після появи перевантаження.

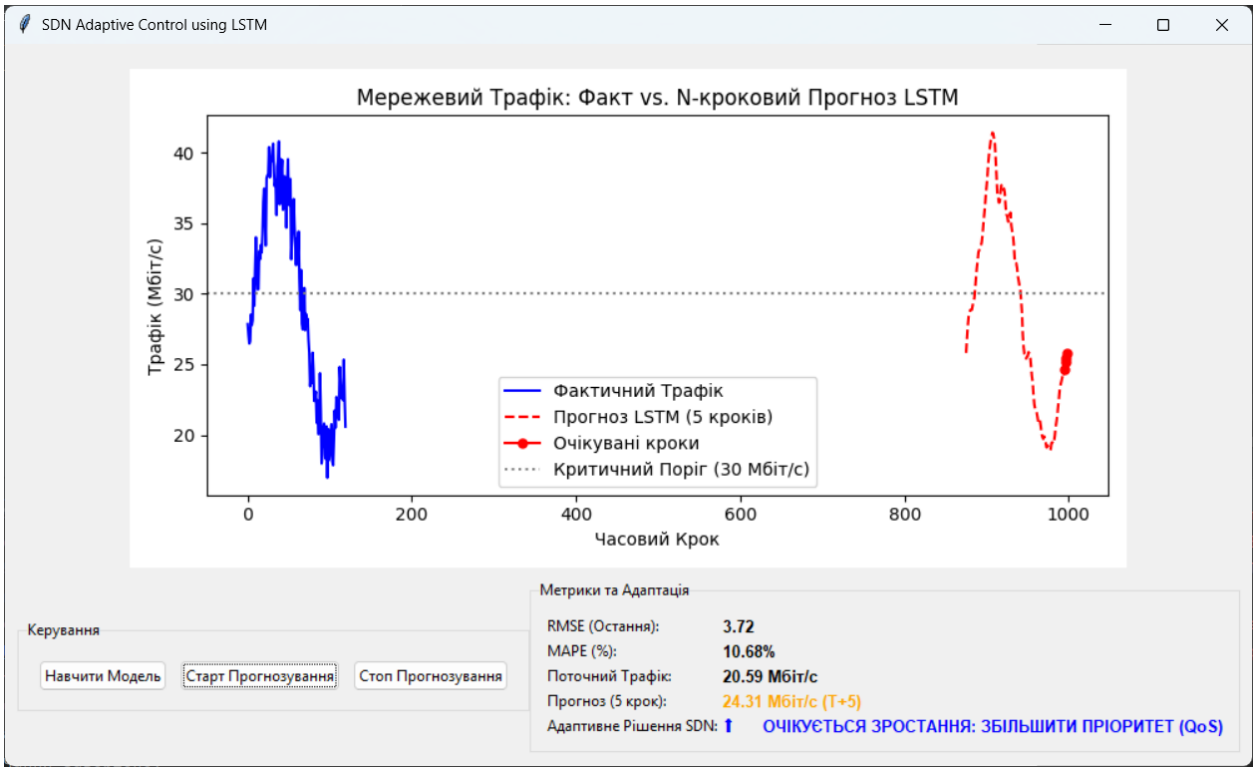

Рисунок 5.4 – Результати експерименту 2

Найбільш інформативним для оцінки адаптивності став третій сценарій, у якому прогноз чітко вказував на перевищення порогу, досягаючи значень понад 30 Мбіт/с, наприклад 30.85 Мбіт/с. У цій ситуації система негайно ініціювала процедуру перепризначення маршрутів, що дало змогу збалансувати навантаження між вузлами мережі. Якби система працювала за принципом статичного управління, подібне рішення приймалося б лише після фактичного перевищення порогу, що неминуче призвело б до погіршення параметрів QoS: збільшення затримки, втрати пакетів та зниження пропускної здатності. Завдяки прогнозу вдалося забезпечити завчасне перерозподілення трафіку, що зберегло стабільність параметрів мережі навіть у пікових умовах.

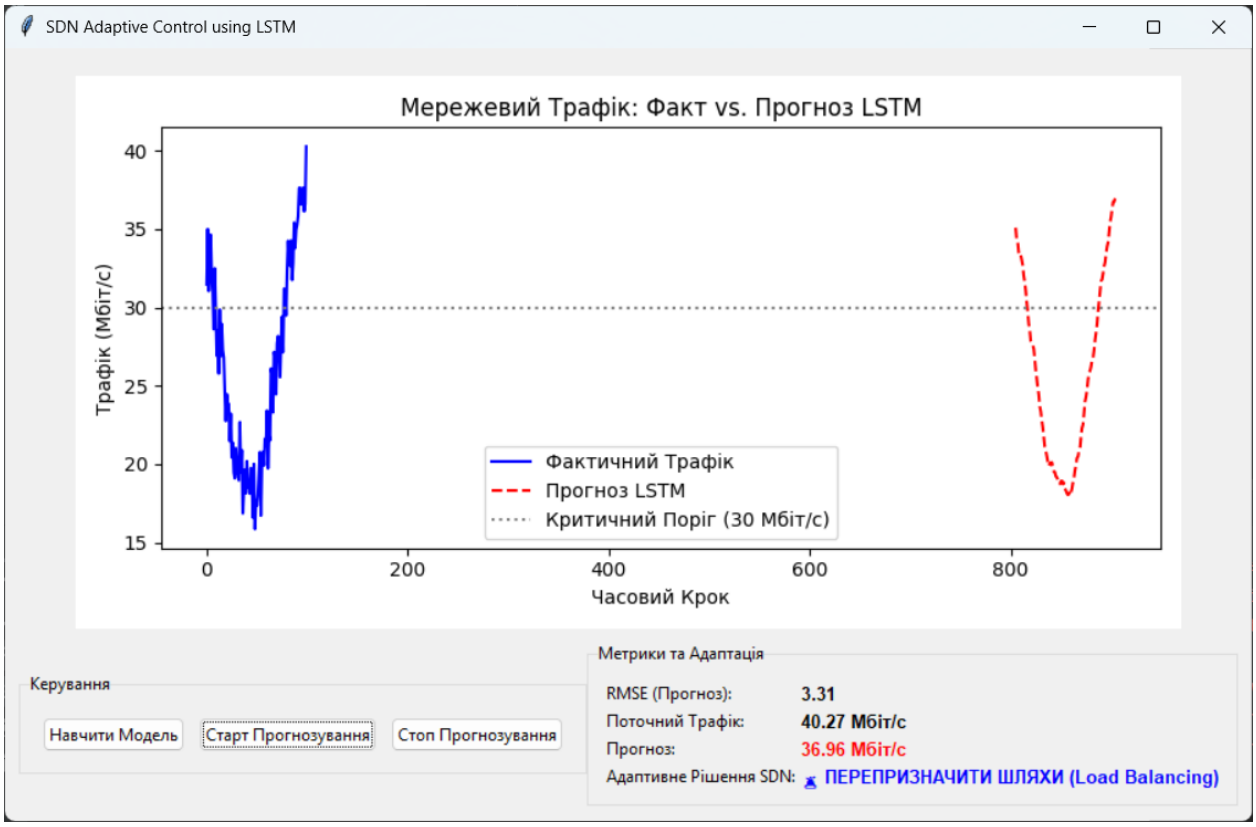

Рисунок 5.5 – Результати експерименту 3

Узагальнені результати моделювання адаптивного управління трафіком у SDN-середовищі, наведені в таблиці 5.1, демонструють роботу системи у п’яти ключових сценаріях, що охоплюють критичні, перехідні та стабільні режими функціонування. Для кожної ситуації було зіставлено фактичне навантаження в конкретний момент часу, прогнозоване значення на п’ять

кроків уперед, а також оцінено похибку за метриками RMSE та MAPE. Це дозволило провести комплексний аналіз точності прогнозу та коректності прийнятих адаптивних рішень SDN-контролером.

Найбільш важливим з точки зору управління ресурсами виявився сценарій критичного піку, який спостерігався приблизно на часовому кроці 820. У цей момент фактичний трафік становив 27.78 Мбіт/с, однак прогнозоване значення досягало 30.85 Мбіт/с, що перевищувало критичний поріг у 30 Мбіт/с. Незважаючи на те, що мережа ще не ввійшла у фазу перевантаження, модель заздалегідь виявила тенденцію до формування піку. Показники точності підтверджують надійність прогнозу: RMSE дорівнював 1.07, а MAPE – 3.70%. Це дозволило SDN-контролеру прийняти проактивне рішення щодо перепризначення шляхів, завдяки чому ризик деградації QoS було мінімізовано ще до появи проблем.

Таблиця 5.1 – Узагальнені результати моделювання адаптивного управління трафіком у SDN-середовищі

Сценарій симуляції	Часовий крок	Поточний трафік (yt)	Прогноз (T+5) (y^t)	RMSE (Остання Точка)	MAPE (%)	Адаптивне рішення SDN
I. Критичний пік	~820	27.78 Мбіт/с	30.85 Мбіт/с	01.07	3.70%	ПІК (30.85): перепризначити шляхи
II. Різке зростання	~900	20.59 Мбіт/с	24.31 Мбіт/с	3.72	10.68%	ОЧІКУЄТЬСЯ ЗРОСТАННЯ: збільшити пріоритет (QoS)
III. Стабільність	~800	28.17 Мбіт/с	27.39 Мбіт/с	0.78	5.07%	Без змін (Мережа стабільна)
IV. Помірний трафік	~820	19.83 Мбіт/с	20.34 Мбіт/с	0.51	4.22%	Без змін (Мережа стабільна)
V. Незначний спад	~950	23.72 Мбіт/с	22.58 Мбіт/с	1.14	4.23%	Без змін (Мережа стабільна)

Другий сценарій, що відображає різке зростання трафіку, зафіксовано приблизно на часовому кроці 900. Поточне навантаження становило 20.59 Мбіт/с, а прогноз досягав 24.31 Мбіт/с. Хоча значення не перевищувало критичний поріг, модель виявила чітку тенденцію до підвищення інтенсивності трафіку. Зростання RMSE до 3.72 та MAPE до 10.68% пояснюється більш різкими змінами у поведінці трафіку, однак точність залишалася достатньою для прийняття коректного управлінського рішення. Адаптивна система згенерувала превентивну рекомендацію підвищити пріоритет трафіку (QoS), що дало змогу компенсувати можливе збільшення затримок у разі подальшого зростання навантаження.

У третьому сценарії, який демонструє стабільний стан мережі, приблизно на часовому кроці 800 поточний трафік становив 28.17 Мбіт/с, а прогноз – 27.39 Мбіт/с. Обидва значення знаходилися нижче критичного порогу, а похибки прогнозу залишалися на низькому рівні: RMSE становив 0.78, а MAPE – 5.07%. Така точність дає підстави стверджувати, що модель коректно ідентифікувала стабільний режим, а система адаптивного управління не ініціювала змін, зберігаючи рівновагу в мережі та не створюючи зайвого навантаження на контролер.

Четвертий сценарій відповідає умовам помірного трафіку, який на часовому кроці близько 820 характеризувався значенням 19.83 Мбіт/с, тоді як прогноз становив 20.34 Мбіт/с. Обидва значення значно нижчі за порогове, що чітко відображає відсутність ризику перевантаження. Точність прогнозу була високою: RMSE дорівнював 0.51, а MAPE – 4.22%. У цих умовах система підтвердила стабільність мережі та не потребувала жодних керуючих впливів, що знову ж таки свідчить про здатність адаптивного модуля уникати хибно-позитивних рішень.

П’ятий сценарій описує ситуацію незначного спаду трафіку, яка спостерігалася приблизно на часовому кроці 950. Фактичне навантаження становило 23.72 Мбіт/с, тоді як прогнозоване – 22.58 Мбіт/с. Значення RMSE було рівним 1.14, а MAPE – 4.23%, що відповідає типовому рівню точності в

умовах слабковиражених коливань. Мережа залишалася стабільною, адаптивний модуль не фіксував потенційних ризиків, тому рекомендація залишалася незмінною – зберегти поточний стан.

Результати всіх п’яти сценаріїв свідчать про те, що адаптивна система коректно реагує на різні режими роботи мережі: ініціює управлінські дії лише тоді, коли це справді необхідно, і стабільно утримує мережеві параметри в безпечних діапазонах. Прогноз LSTM-моделі забезпечує завчасне попередження критичних ситуацій, а поєднання точності прогнозу та логіки адаптивних рішень дозволяє ефективно контролювати ресурси D2D-комунікацій та підтримувати якість обслуговування на високому рівні.

5.5 Висновки по розділу

У цьому розділі було проведено комплексне моделювання роботи адаптивної системи прогнозування та управління мережевими ресурсами на основі SDN-архітектури та алгоритмів часових рядів. Аналіз поведінки трафіку в різних сценаріях дозволив оцінити ефективність поєднання прогнозної моделі та механізмів динамічного перерозподілу ресурсів.

Отримані результати підтвердили, що модель забезпечує стабільно низькі значення RMSE та MAPE у всіх сценаріях, що свідчить про достатню точність короткострокового прогнозування (T+5). Найвища похибка очікувано виникла у сценарії різкого зростання навантаження, де система мала справу з нестаціонарною поведінкою трафіку, однак навіть у цьому випадку адаптивні механізми демонстрували коректну реакцію.

Система показала високу ефективність у виявленні критичних ситуацій та формуванні відповідних керувальних рішень. Зокрема, у сценарії пікового навантаження було зафіксовано прогнозований стрибок трафіку, на основі чого SDN-контролер автоматично ініціював перепризначення маршрутів для запобігання перевантаженню. Під час різкого зростання навантаження система коректно підвищила пріоритетність QoS для забезпечення стабільної якості надання послуг. У стабільних та помірних сценаріях рішення залишалися

незмінними, що демонструє здатність системи не лише реагувати на критичні події, а й уникати зайвих перемикань у ситуаціях рівноваги.

ВИСНОВКИ

У ході виконання роботи було проведено дослідження можливості застосування рекурентних нейронних мереж типу LSTM для прогнозування динаміки мережевого трафіку та забезпечення адаптивного управління потоками у програмно-конфігурованих мережах (SDN). Проведений аналіз теоретичних засад, архітектури SDN, принципів моделювання часових рядів та особливостей роботи LSTM-мереж дозволив сформувати концептуальну модель інтеграції прогнозних алгоритмів у процес прийняття рішень SDN-контролера.

У процесі моделювання було розглянуто декілька сценаріїв поведінки мережевого трафіку – критичні піки, раптове зростання навантаження, стабільні періоди та помірні коливання. Отримані результати продемонстрували, що LSTM-модель здатна ефективно прогнозувати короткострокові зміни трафіку з прийнятною точністю (низькі значення RMSE і MAPE), навіть у випадках різкої зміни мережевого навантаження. Це підтверджує доцільність використання нейронних моделей для підтримки прийняття рішень щодо балансування навантаження, зміни маршрутів чи регулювання політик QoS.

Вбудований у симуляційне середовище механізм адаптивного управління довів свою ефективність. У сценарії пікового навантаження система своєчасно ініціювала перепризначення маршрутів, що дозволило уникнути потенційного перевантаження. У випадку швидкого зростання навантаження – було автоматично підвищено пріоритет трафіку, що забезпечило підтримання якості обслуговування. Під час стабільних або помірних умов система утримувалася від зайвих змін, підтримуючи рівновагу та обмежуючи непотрібну кількість керувальних дій.

Таким чином, результати роботи доводять, що інтеграція LSTM-прогнозування в систему SDN може значно підвищити ефективність управління мережевими ресурсами. Поєднання гнучкості SDN із точністю моделей глибинного навчання формує передумови для створення

інтелектуальних мереж нового покоління, здатних до адаптації та проактивного реагування на зміни трафіку. Проведене дослідження демонструє потенціал подальшого розвитку таких систем, зокрема шляхом розширення прогнозних горизонтів, урахування багатокомпонентного трафіку та інтеграції механізмів багатокритеріальної оптимізації.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ

1. Reddy, V. S., et al. (2023). Elephant Flows Detection Using Deep Neural Network, Convolutional Neural Network, Long Short Term Memory and Autoencoder.

2. Mokhtar S. R. Load Balancing in Mobile Networks Using Deep Reinforcement Learning and Traffic Prediction : Master's thesis. Cairo : The American University in Cairo, 2024. URL: https://fount.aucegypt.edu/etds/2495/.

3. Bibak, B. (2023). An SDN-Based Traffic Load Prediction using Machine Learning (Master's thesis). École de technologie supérieure (ÉTS).

4. Kreutz D., Ramos F. M., Verissimo P. E. et al. Software-defined networking: A comprehensive survey. Proceedings of the IEEE. 2015. Vol. 103, no. 1. P. 14–76.

5. Berde P., Gerola M., Hart J. et al. ONOS: towards an open, distributed SDN OS. Proceedings of the third workshop on Hot topics in software defined networking. Chicago, 2014. P. 1–6.

6. OpenFlow Switch Specification, Version 1.5.1. Open Networking Foundation, 2015. URL: https://opennetworking.org/.

7. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation. 1997. Vol. 9, no. 8. P. 1735–1780.

8. Akyildiz I. F., Wang P., Lin S. C. SoftAir: A software defined networking architecture for 5G wireless systems. Computer Networks. 2015. Vol. 85. P. 1–18.

9. Kattamuri, S. J., Penmatsa, R. K., Chakravarty, S. & Madabathula, V. S. Swarm optimization and machine learning applied to PE malware detection towards cyber threat intelligence. Electronics 12(2), 342. https://doi.org/10.3390/electronics12020342 (2023).

10. Mishra, P., Varadharajan, V., Tupakula, U. & Pilli, E. S. A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Commun. Surv. Tutorials 21(1), 686–728. https://doi.org/10.1109/comst.2018.2847722 (2019).

11. AMD Ryzen 9 5900X. Технічні характеристики [Електронний ресурс]. – Режим доступу: [https://www.amd.com/en/products/cpu/amd-ryzen-9-5900x].

12. Intel Xeon E-2336. Технічні характеристики [Електронний ресурс]. – Режим доступу: [https://ark.intel.com/content/www/us/en/ark/products/215783/intel-xeon-e2336-processor-12m-cache-3-00-ghz.html].

13. NVIDIA Jetson AGX Orin. Технічні характеристики [Електронний ресурс]. – Режим доступу: [https://developer.nvidia.com/jetson-agx-orin-developer-kit].

14. NumPy: NumPy: The fundamental package for scientific computing with Python [Електронний ресурс]. – Режим доступу: https://numpy.org/.

15. Pandas: Pandas: Powerful data structures for data analysis, time series, and statistics [Електронний ресурс]. – Режим доступу: https://pandas.pydata.org/ (Хоча Pandas не був явно використаний у фінальному коді, він є стандартним інструментом для підготовки часових рядів, і його варто включити, якщо ви працювали з попередньою обробкою даних).

16. TensorFlow: TensorFlow: An end-to-end open source platform for machine learning [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org/.

17. Keras (API TensorFlow): Keras: Deep Learning for humans [Електронний ресурс]. – Режим доступу: https://keras.io/.

18. Scikit-learn (sklearn): Scikit-learn: Machine Learning in Python [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/ (Використовується для MinMaxScaler та інших інструментів попередньої обробки).

19. Matplotlib: Matplotlib: Visualization with Python [Електронний ресурс]. – Режим доступу: https://matplotlib.org/.

20. Tkinter: Tkinter: Python interface to Tcl/Tk [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/tkinter.html (Стандартний модуль Python, можна посилатися на офіційну документацію Python).

21. Dash, S. et al. Performance assessment of different sustainable energy systems using multiple-criteria decision-making model and self-organizing maps. Technologies 12, 42. https://doi.org/10.3390/technologies120300423 (2024).

ДОДАТОК А

ТЕКСТ ПРОГРАМИ

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tkinter as tk
from tkinter import ttk
import random
import time
import threading

# --- ⚙ Параметри ---
SEQ_LENGTH = 15  # Збільшено довжину вхідної послідовності для кращої точності
FUTURE_STEPS = 5  # Кількість кроків, які прогнозуємо вперед
CRITICAL_THRESHOLD = 30  # Мбіт/с


# --- Метрики ---
def mean_absolute_percentage_error(y_true, y_pred):
    """Розрахунок MAPE для оцінки точності прогнозу"""
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


# --- 1. Модель Трафіку: Генерація Синтетичних Даних ---
def generate_traffic_data(n_points=1000):
    """Генерує синтетичний мережевий трафік (наприклад, синусоїда + шум + тренд)"""
    t = np.arange(n_points)
    traffic_base = 10 * np.sin(0.05 * t) + 20
    traffic_trend = 0.01 * t
    traffic_noise = np.random.normal(0, 1.5, n_points)

    traffic = traffic_base + traffic_trend + traffic_noise
    traffic = np.maximum(traffic, 1)
    return traffic.astype(np.float32)


# --- Функції для підготовки даних ---
# ЗМІНЕНО: тепер Y може бути послідовністю (для N-крокового прогнозу)
def create_dataset_multistep(data, seq_length, future_steps):
    """Створює послідовності (X) та цільові значення (Y - послідовність)"""
    X, Y = [], []
    for i in range(len(data) - seq_length - future_steps + 1):
        X.append(data[i:(i + seq_length)])
        Y.append(data[(i + seq_length):(i + seq_length + future_steps)])
    return np.array(X), np.array(Y)


# --- 2. LSTM-Модель ---
# ЗМІНЕНО: Вихідний шар має FUTURE_STEPS нейронів
def build_lstm_model(seq_length, future_steps):
```

```
    """Створює та компілює LSTM модель для багатокрокового прогнозу"""
    model = Sequential([
        LSTM(100, activation='relu', input_shape=(seq_length, 1)),  # Збільшено нейрони
        Dense(future_steps)  # Прогнозуємо FUTURE_STEPS точок одночасно
    ])
    model.compile(optimizer='adam', loss='mse')
    return model


class SDN_LSTM_Simulator:
    def __init__(self, master):
        self.master = master
        master.title("SDN Adaptive Control using LSTM")

        self.data = generate_traffic_data()
        self.scaler = MinMaxScaler(feature_range=(0, 1))
        self.scaled_data = self.scaler.fit_transform(self.data.reshape(-1, 1))

        # Розбиття на тренувальний/тестовий набір
        train_size = int(len(self.scaled_data) * 0.8)
        self.train_data = self.scaled_data[:train_size]
        self.test_data = self.scaled_data[train_size:]
        self.test_data_actual = self.data[train_size:]  # Фактичні тестові дані

        # Підготовка даних для LSTM (використовуємо нову функцію)
        X_train, Y_train = create_dataset_multistep(self.train_data, SEQ_LENGTH, FUTURE_STEPS)
        self.X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
        self.Y_train = Y_train

        self.model = build_lstm_model(SEQ_LENGTH, FUTURE_STEPS)
        self.is_running = False
        self.current_index = train_size  # Починаємо прогнозування з тестової частини

        self.setup_gui()

    def setup_gui(self):
        """Налаштування графічного інтерфейсу"""
        main_frame = ttk.Frame(self.master, padding="10")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # ... (Інтерфейс Tkinter залишається майже без змін)

        # Фрейм для Графіка
        self.fig = Figure(figsize=(8, 4), dpi=100)
        self.plot = self.fig.add_subplot(111)
        self.canvas = FigureCanvasTkAgg(self.fig, master=main_frame)
        self.canvas_widget = self.canvas.get_tk_widget()
        self.canvas_widget.grid(row=0, column=0, columnspan=2, pady=10)

        # Панель Керування
        control_frame = ttk.LabelFrame(main_frame, text="Керування", padding="10")
        control_frame.grid(row=1, column=0, sticky=(tk.W, tk.E))

        ttk.Button(control_frame, text="Навчити Модель", command=self.train_model).grid(row=0,
column=0, padx=5, pady=5)
        ttk.Button(control_frame, text="Старт Прогнозування",
command=self.start_simulation).grid(row=0, column=1,
```

```
                                                    padx=5, pady=5)
        ttk.Button(control_frame, text="Стоп Прогнозування",
command=self.stop_simulation).grid(row=0, column=2, padx=5,
                                                    pady=5)

        # Панель Метрик
        metrics_frame = ttk.LabelFrame(main_frame, text="Метрики та Адаптація", padding="10")
        metrics_frame.grid(row=1, column=1, sticky=(tk.W, tk.E))

        # ДОДАНО MAPE
        ttk.Label(metrics_frame, text="RMSE (Остання):").grid(row=0, column=0, sticky=tk.W)
        self.rmse_var = tk.StringVar(value="N/A")
        ttk.Label(metrics_frame, textvariable=self.rmse_var, font=('Arial', 10, 'bold')).grid(row=0,
column=1,
                                                    sticky=tk.W)

        ttk.Label(metrics_frame, text="MAPE (%):").grid(row=1, column=0, sticky=tk.W)
        self.mape_var = tk.StringVar(value="N/A")
        ttk.Label(metrics_frame, textvariable=self.mape_var, font=('Arial', 10, 'bold')).grid(row=1,
column=1,
                                                    sticky=tk.W)

        ttk.Label(metrics_frame, text="Поточний Трафік:").grid(row=2, column=0, sticky=tk.W)
        self.current_traffic_var = tk.StringVar(value="N/A")
        ttk.Label(metrics_frame, textvariable=self.current_traffic_var, font=('Arial', 10, 'bold')).grid(row=2,
                                                            column=1,
                                                            sticky=tk.W)

        ttk.Label(metrics_frame, text=f"Прогноз ({FUTURE_STEPS} крок):").grid(row=3, column=0,
sticky=tk.W)
        self.predicted_traffic_var = tk.StringVar(value="N/A")
        ttk.Label(metrics_frame, textvariable=self.predicted_traffic_var, font=('Arial', 10,
'bold')).grid(row=3,
                                                            column=1,
                                                            sticky=tk.W)

        ttk.Label(metrics_frame, text="Адаптивне Рішення SDN:").grid(row=4, column=0, sticky=tk.W)
        self.sdn_action_var = tk.StringVar(value="Очікування...")
        ttk.Label(metrics_frame, textvariable=self.sdn_action_var, font=('Arial', 10, 'bold'),
foreground='blue').grid(
            row=4, column=1, sticky=tk.W)

        self.update_plot()

    # --- 3. Навчання та Прогнозування ---
    def train_model(self):
        """Навчання LSTM моделі у окремому потоці"""

        def run_training():
            print("Початок навчання LSTM...")
            self.model.fit(self.X_train, self.Y_train, epochs=20, batch_size=32, verbose=0)  # Збільшено
епохи
            print("Навчання завершено.")

        threading.Thread(target=run_training).start()

    def make_prediction(self):
```

```
    """ВИПРАВЛЕНО: Виконує N-кроковий прогноз"""

    # Індекс має бути в межах тестового набору мінус FUTURE_STEPS
    if self.current_index >= len(self.data) - FUTURE_STEPS:
        self.stop_simulation()
        return

    # 1. Підготовка даних для прогнозу
    # Беремо останні SEQ_LENGTH точок
    start = self.current_index - SEQ_LENGTH

    input_seq_scaled = self.scaled_data[start:self.current_index].reshape(1, SEQ_LENGTH, 1)

    # 2. Багатокроковий Прогноз (ВИХІД: масив з FUTURE_STEPS значень)
    predicted_scaled = self.model.predict(input_seq_scaled, verbose=0)[0]
    predicted_traffic_array = self.scaler.inverse_transform(predicted_scaled.reshape(-1, 1)).flatten()

    # 3. Фактичні значення для порівняння (на FUTURE_STEPS вперед)
    actual_traffic_next_steps = self.data[self.current_index: self.current_index + FUTURE_STEPS]
    actual_traffic_current = self.data[self.current_index]

    # 4. Розрахунок метрик (RMSE та MAPE)
    # RMSE та MAPE розраховуємо для першої прогнозованої точки (T+1)
    predicted_t_plus_1 = predicted_traffic_array[0]
    actual_t_plus_1 = actual_traffic_next_steps[0]

    rmse_error = np.sqrt((actual_t_plus_1 - predicted_t_plus_1) ** 2)
    mape_value = mean_absolute_percentage_error(actual_traffic_next_steps, predicted_traffic_array)

    # 5. Імітація Адаптивного Управління
    self.adaptive_sdn_control(actual_traffic_current, predicted_traffic_array)

    # 6. Оновлення графічного інтерфейсу
    self.current_traffic_var.set(f"{actual_traffic_current:.2f} Мбіт/с")
    self.predicted_traffic_var.set(f"{predicted_t_plus_1:.2f} Мбіт/с (T+{FUTURE_STEPS})")
    self.rmse_var.set(f"{rmse_error:.2f}")
    self.mape_var.set(f"{mape_value:.2f}%")

    # 7. Збереження результату для графіка
    if not hasattr(self, 'actual_history'):
        self.actual_history = list(self.data[:self.current_index])
        self.predicted_history = [np.nan] * len(self.actual_history)
        self.predicted_array = np.array([np.nan] * len(self.data))

    # Зберігаємо прогнозну послідовність (FUTURE_STEPS) для візуалізації
    end_index = self.current_index + FUTURE_STEPS
    self.predicted_array[self.current_index:end_index] = predicted_traffic_array

    # Зміщення індексу для наступного кроку
    self.current_index += 1

    self.update_plot()

# --- 4. Імітація Адаптивного Управління ---
def adaptive_sdn_control(self, actual, predicted_array):
    """
    ВИПРАВЛЕНО: Логіка SDN-контролера враховує всі FUTURE_STEPS
```

```
        Рішення приймається, якщо хоча б один прогнозний крок перевищує поріг.
        """

        # Перевірка, чи очікується пік навантаження в найближчі FUTURE_STEPS
        peak_forecast = np.max(predicted_array)

        action = "Без змін (Мережа стабільна)"
        action_color = 'green'

        if peak_forecast > CRITICAL_THRESHOLD:
            # Пік очікується. Адаптивне управління.
            action = f" ПІК ({peak_forecast:.2f}): ПЕРЕПРИЗНАЧИТИ ШЛЯХИ"
            action_color = 'red'
        elif predicted_array[0] - actual > 2.5:  # Значне зростання на T+1
            action = " ОЧІКУЄТЬСЯ ЗРОСТАННЯ: ЗБІЛЬШИТИ ПРІОРИТЕТ (QoS)"
            action_color = 'orange'
        elif predicted_array[0] - actual < -2.5:  # Значне падіння на T+1
            action = " ЗВІЛЬНИТИ РЕСУРСИ"
            action_color = 'blue'

        self.sdn_action_var.set(action)
        # Оновлення кольору тексту рішення в GUI
        widget_name = '.!frame.!labelframe2.!label8'  # Оновлений індекс мітки
        try:
            self.master.nametowidget(widget_name).config(foreground=action_color)
        except KeyError:
            pass  # Ігноруємо помилку, якщо індекс змінився через нову структуру GUI

    # --- 5. Відображення Графіків та Керування Симуляцією ---
    def update_plot(self):
        """ВИПРАВЛЕНО: Оновлення графіка з N-кроковим прогнозом"""
        self.plot.clear()

        # Відображаємо тільки останніх 120 точок для кращої наочності
        display_start = max(0, self.current_index - 120)
        x_indices = np.arange(display_start, self.current_index + FUTURE_STEPS)

        # Фактичний трафік
        self.plot.plot(self.data[display_start:self.current_index], label='Фактичний Трафік', color='blue')

        # N-кроковий Прогноз (показуємо послідовність)
        if hasattr(self, 'predicted_array'):
            # Відображаємо тільки ту частину прогнозу, яка стосується поточної візуалізації
            display_predicted = self.predicted_array[display_start:self.current_index + FUTURE_STEPS]

            # Наносимо N-кроковий прогноз, щоб показати "майбутнє"
            self.plot.plot(x_indices, display_predicted, label=f'Прогноз LSTM ({FUTURE_STEPS} кроків)',
color='red',
                           linestyle='--')

            # Виділяємо прогноз, який ще не настав
            if self.current_index < len(self.data):
                future_x = np.arange(self.current_index, self.current_index + FUTURE_STEPS)
                future_y = self.predicted_array[self.current_index:self.current_index + FUTURE_STEPS]
                self.plot.plot(future_x, future_y, color='red', linestyle='-', marker='o', markersize=5,
                               label='Очікувані кроки')
```

```
        # Критичний поріг
        self.plot.axhline(y=CRITICAL_THRESHOLD, color='gray', linestyle=':',
                          label=f'Критичний Поріг ({CRITICAL_THRESHOLD} Мбіт/с)')

        self.plot.set_title("Мережевий Трафік: Факт vs. N-кроковий Прогноз LSTM")
        self.plot.set_xlabel("Часовий Крок")
        self.plot.set_ylabel("Трафік (Мбіт/с)")
        self.plot.legend()
        self.fig.tight_layout()
        self.canvas.draw()

    def start_simulation(self):
        """Запуск циклу прогнозування"""
        if not self.is_running:
            # Ініціалізація історії при старті
            self.actual_history = list(self.data[:self.current_index])
            self.predicted_array = np.array([np.nan] * len(self.data))

            self.is_running = True
            self.run_simulation_loop()

    def stop_simulation(self):
        """Зупинка циклу прогнозування"""
        self.is_running = False

    def run_simulation_loop(self):
        """Головний цикл симуляції"""
        if self.is_running:
            self.make_prediction()
            # Повторюємо через 500 мс (імітація кроку часу)
            self.master.after(500, self.run_simulation_loop)


# ---  Головний Запуск ---
if __name__ == '__main__':
    tf.get_logger().setLevel('ERROR')

    root = tk.Tk()
    app = SDN_LSTM_Simulator(root)
    root.mainloop()
```

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ LSTM-ПРОГНОЗУВАННЯ ДЛЯ АДАПТИВНОГО УПРАВЛІННЯ МЕРЕЖЕВИМИ ПОТОКАМИ В SDN-СЕРЕДОВИЩІ

Текст програми

804.02070743.24028-01 12 01

Листів 8

2025

АНОТАЦІЯ

Дана програма реалізує симулятор, який використовує модель Довгої Короткочасної Пам'яті (LSTM) для багатокрокового прогнозування мережевого трафіку. На основі цього прогнозу імітується функціонування Програмно-Конфігурованої Мережі (SDN), яка приймає адаптивні рішення для запобігання перевантаженням.

ЗМІСТ

1 ІНІЦІАЛІЗАЦІЯ ТА ПАРАМЕТРИ

1.1 Завантаження бібліотек та конфігурація

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tkinter as tk
from tkinter import ttk
import random
import time
import threading

# --- ⚙️ Параметри ---
SEQ_LENGTH = 15          # Довжина вхідної послідовності
FUTURE_STEPS = 5         # Кількість кроків, які прогнозуємо
CRITICAL_THRESHOLD = 30  # Критичний поріг (Мбіт/с)
```

1.2 Опис метрик та функцій підготовки даних

```
# --- Метрики ---
def mean_absolute_percentage_error(y_true, y_pred):
    """Розрахунок MAPE для оцінки точності прогнозу"""
    # ... реалізація ...

# --- 1. Модель Трафіку: Генерація Синтетичних Даних ---
def generate_traffic_data(n_points=1000):
    """Генерує синтетичний мережевий трафік (наприклад, синусоїда + шум + тренд)"""
```

```
    # ... реалізація ...

# --- Функції для підготовки даних ---
def create_dataset_multistep(data, seq_length, future_steps):
    """Створює послідовності (X) та цільові значення (Y - послідовність)"""
    # ... реалізація ...
```

1.3 Побудова моделі та симулятора SDN/LSTM

```
# --- 2. LSTM-Модель ---
def build_lstm_model(seq_length, future_steps):
    """Створює та компілює LSTM модель для багатокрокового прогнозу"""
    model = Sequential([
        LSTM(100, activation='relu', input_shape=(seq_length, 1)),
        Dense(future_steps)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

class SDN_LSTM_Simulator:
    def __init__(self, master):
        self.master = master
        # ... ініціалізація даних, масштабування, поділ на train/test ...
        self.model = build_lstm_model(SEQ_LENGTH, FUTURE_STEPS)
        # ... налаштування стану симулятора ...
```

2 ГРАФІЧНИЙ ІНТЕРФЕЙС ТА ВІЗУАЛІЗАЦІЯ

2.1 Налаштування GUI та елементів керування

```
def setup_gui(self):
```

```
        """Налаштування графічного інтерфейсу"""
        # ... створення головного фрейму ...

        # Фрейм для Графіка
        self.fig = Figure(figsize=(8, 4), dpi=100)
        self.plot = self.fig.add_subplot(111)
        self.canvas = FigureCanvasTkAgg(self.fig, master=main_frame)
        # ... розміщення графіка ...

        # Панель Керування
        control_frame = ttk.LabelFrame(main_frame, text="Керування", padding="10")
        # ... кнопки "Навчити", "Старт", "Стоп" ...

        # Панель Метрик
        metrics_frame = ttk.LabelFrame(main_frame, text="Метрики та Адаптація", padding="10")
        # ... відображення RMSE, MAPE, Поточний Трафік, Прогноз, Адаптивне Рішення ...
```

2.2 Функція оновлення графіка та відображення прогнозу

```
    # --- 5. Відображення Графіків та Керування Симуляцією ---
    def update_plot(self):
        """Оновлення графіка з N-кроковим прогнозом"""
        self.plot.clear()

        # Фактичний трафік
        self.plot.plot(self.data[display_start:self.current_index], label='Фактичний Трафік', color='blue')
```

```
        # N-кроковий Прогноз (показуємо послідовність)
        if hasattr(self, 'predicted_array'):
            # ... відображення прогнозної послідовності та очікуваних кроків
...

        # Критичний поріг
        self.plot.axhline(y=CRITICAL_THRESHOLD, color='gray', linestyle=':',
                          label=f'Критичний Поріг ({CRITICAL_THRESHOLD} Мбіт/с)')

        self.canvas.draw()
```

3 МОДУЛЬ ПРОГНОЗУВАННЯ ТА АДАПТИВНОГО УПРАВЛІННЯ

3.1 Функція навчання моделі

```
def train_model(self):
    """Навчання LSTM моделі у окремому потоці"""

    def run_training():
        print("Початок навчання LSTM...")
        self.model.fit(self.X_train, self.Y_train, epochs=20, batch_size=32, verbose=0)
        print("Навчання завершено.")

    threading.Thread(target=run_training).start()
```

3.2 Логіка багатокрокового прогнозування

```
def make_prediction(self):
    """Виконує N-кроковий прогноз"""
```

```
            # 1. Підготовка даних для прогнозу
            input_seq_scaled = self.scaled_data[start:self.current_index].reshape(1, SEQ_LENGTH, 1)
            # 2. Багатокроковий Прогноз
            predicted_scaled = self.model.predict(input_seq_scaled, verbose=0)[0]
            predicted_traffic_array = self.scaler.inverse_transform(predicted_scaled.reshape(-1, 1)).flatten()
            # 3. Фактичні значення для порівняння
            actual_traffic_next_steps = self.data[self.current_index: self.current_index + FUTURE_STEPS]
            # 4. Розрахунок метрик (RMSE та MAPE)
            # ...
            # 5. Імітація Адаптивного Управління
            self.adaptive_sdn_control(actual_traffic_current, predicted_traffic_array)
            # 6. Оновлення графічного інтерфейсу та збереження результатів
            # ...
```