МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ПРОЦАЙ МИХАЙЛО ОЛЕКСАНДРОВИЧ

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

КОМП'ЮТЕРНО-МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ СИСТЕМИ ВИЯВЛЕННЯ ПОРУШЕНЬ КОРЕКТНОГО ФУНКЦІОНУВАННЯ СИСТЕМ КРИТИЧНОЇ ІНФОРМАЦІЙНОЇ ІНФРАСТРУКТУРИ

Спеціальність 113 Прикладна математика

Кваліфікаційна (магістерська) робота

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

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

Вінниця – 2024

АНОТАЦІЯ

У кваліфікаційній роботі досліджено проблематику комп'ютерно-математичного моделювання систем виявлення порушень коректного функціонування систем критичної інформаційної інфраструктури. Проведено глибокий аналіз теоретичних аспектів, в тому числі характеристик систем критичної інформаційної інфраструктури та потенційних загроз для них.

Встановлено, що використання сучасних методів машинного навчання, зокрема штучних нейронних мереж, може значно підвищити ефективність виявлення порушень. Основними науковими результатами є розроблена математична модель та алгоритм машинного навчання для виявлення порушень, імплементовані на мові програмування C#. Експериментальна частина роботи базувалася на датасеті з авторитетного ресурсу Kaggle і підтвердила високу ефективність моделі у задачі виявлення DDos атак.

Показано, що розроблена система має високі показники якості, включаючи низьке значення Log-loss та високу площу під ROC-кривою. Результати експериментальної перевірки системи підтвердили її ефективність та точність, а також підкреслили її придатність для виявлення складних шаблонів порушень в сучасних умовах.

Ключові слова: критична інформаційна інфраструктура, машинне навчання, штучні нейронні мережі, DDoS, виявлення порушень, математичне моделювання, кібербезпека.

97 с., 4 табл., 37 рис., 2 дод., 51 джерело.

ABSTRACT

In the qualification work, the problems of computer-mathematical modeling of systems for detecting violations of the correct functioning of critical information infrastructure systems were investigated. An in-depth analysis of theoretical aspects was carried out, including the characteristics of critical information infrastructure systems and potential threats to them.

It has been established that the use of modern methods of machine learning, in particular artificial neural networks, can significantly increase the effectiveness of detection of violations. The main scientific results are a developed mathematical model and a machine learning algorithm for detecting violations, implemented in the C# programming language. The experimental part of the work was based on a dataset from the authoritative Kaggle resource and confirmed the high efficiency of the model in the task of detecting DDos attacks.

It is shown that the developed system has high quality indicators, including a low Log-loss value and a high area under the ROC curve. The results of experimental verification of the system confirmed its effectiveness and accuracy, and also emphasized its suitability for detecting complex patterns of violations in modern conditions.

Keywords: critical information infrastructure, machine learning, artificial neural networks, DDoS, breach detection, mathematical modeling, cyber security.

97 pp., 4 tables, 37 figures, 2 appendices, 51 sources.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – База Даних

ІБ – Інформаційна Безпека

КІІ – Критична Інформаційна Інфраструктура

МБФГШ – Метод Бройдена–Флетчера–Гольдфарба–Шанно

МВЛ – Метод випадкового лісу

МН – Машинне Навчання

МОВ – Метод опорних векторів

ШНМ – Штучна Нейронна Мережа

API – Інтерфейс Програмування Додатків (Application Programming Interface)

AUC – Площа під ROC–Кривою (Area Under Curve)

C# – Мова Програмування C Sharp

DDoS – Розподілена Відмова в Службі (Distributed Denial of Service)

Log–loss – Логарифмічна Функція Втрат

ROC – Характеристика Роботи Оператора (Receiver Operating Characteristic)

SQL – Мова Запитів до Структурованих Баз Даних (Structured Query Language)

URI – Ідентифікатор Ресурсу (Uniform Resource Identifier)

ВСТУП

В сучасному інформаційному суспільстві системи критичної інформаційної інфраструктури (СКІІ) відіграють ключову роль у забезпеченні стабільності та безпеки держави. Ці системи включають в себе різноманітні компоненти, починаючи від енергетичних станцій та закінчуючи системами управління транспортною інфраструктурою. Порушення в роботі таких систем може призвести до катастрофічних наслідків, що акцентує увагу на важливості їхнього надійного та безперебійного функціонування.

Актуальність теми дослідження визначається кількома ключовими факторами. По-перше, в умовах постійного росту кіберзагроз та відкритості глобальних мереж, системи критичної інфраструктури стають потенційними цілями для кібератак. По-друге, існуючі методи виявлення порушень часто базуються на застарілих технологіях, які не можуть ефективно протистояти сучасним кіберзагрозам.

Порівнюючи з відомими розв’язаннями, більшість існуючих систем фокусуються на реактивних методах відновлення після виникнення порушень, замість превентивного виявлення та запобігання. Це ставить під сумнів їхню ефективність та може призвести до великих матеріальних та соціальних збитків. Відсутність комплексного підходу до моделювання та аналізу СКІІ робить актуальним розробку нових методологій та інструментаріїв.

Дослідження цієї теми є вкрай необхідним для розвитку кібербезпеки в Україні, яка в свою чергу є стратегічним напрямком для забезпечення національної безпеки. Моделювання систем виявлення порушень дозволить не тільки підвищити рівень захисту існуючих СКІІ, але й забезпечити основу для розробки нових, більш надійних систем.

Враховуючи вищезгадане, це дослідження спрямоване на розробку комп’ютерно-математичних моделей для ефективного виявлення порушень в системах критичної інформаційної інфраструктури, що є актуальною та доцільною задачею з точки зору наукового та практичного застосування.

Метою даної роботи є розробка комп’ютерно-математичної моделі для ефективного виявлення порушень коректного функціонування систем критичної інформаційної інфраструктури, що сприятиме забезпеченню їхньої надійності та безпеки.

Для реалізації поставленої мети необхідно вирішити наступні задачі:

– проаналізувати сучасні методи та інструменти для виявлення порушень у СКІІ, виявити їхні слабкі та сильні сторони;

– розробити теоретичну математичну модель, яка буде описувати динаміку функціонування та можливі порушення в СКІІ;

– вибрати оптимальні алгоритми та технології для реалізації моделі, виходячи з теоретичного аналізу;

– розробити прототип програмного забезпечення, який використовує розроблену модель для виявлення порушень у СКІІ;

– провести комплексне тестування розробленого програмного забезпечення на реальних або симульованих даних для перевірки його ефективності та надійності.

Об’єктом даного дослідження є системи критичної інформаційної інфраструктури (СКІІ), що включають різні компоненти та модулі для управління важливими секторами держави, такими як енергетика, транспорт, здоров’я тощо. Ці системи представляють собою комплексне явище, в якому взаємодіють численні технологічні, програмні та людські ресурси. Порушення в їхньому функціонуванні може призвести до серйозних наслідків на рівні держави, а тому їхнє вивчення та захист є проблемою високої актуальності.

Предметом дослідження є методи та алгоритми для виявлення порушень коректного функціонування СКІІ. Це включає розробку комп’ютерно-математичної моделі, яка зможе прогнозувати, виявляти та діагностувати потенційні порушення в роботі цих систем.

Для досягнення поставленої мети та вирішення задач дослідження були використані наступні методи: аналітичний, математичне моделювання, комп’ютерна симуляція, алгоритмічний аналіз, критичний аналіз та синтез.

Комбінація цих методів дозволить глибоко вивчити проблему, розробити ефективну модель для її рішення та оцінити їх практичну придатність.

Наукова новизна дослідження полягає в розробці математичної моделі, яка використовує нейронні мережі для виявлення аномалій. Цей підхід не тільки підвищує точність і швидкість виявлення потенційних порушень, але і має велике значення для стратегічного забезпечення кібербезпеки в Україні.

Практичне значення одержаних результатів має безпосередній вплив на забезпечення кібербезпеки в системах критичної інформаційної інфраструктури. Розроблена математична модель, яка використовує нейронні мережі для виявлення аномалій, може бути інтегрована в сучасні системи моніторингу та управління для підвищення їх ефективності та надійності. Це особливо актуально для стратегічно важливих секторів економіки та національної безпеки України, таких як енергетика, транспорт та охорона здоров’я.

1 ТЕОРЕТИЧНІ АСПЕКТИ ПРОБЛЕМИ КОМП'ЮТЕРНО-МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ СИСТЕМ ВИЯВЛЕННЯ ПОРУШЕНЬ КОРЕКТНОГО ФУНКЦІОНУВАННЯ СИСТЕМ КРИТИЧНОЇ ІНФОРМАЦІЙНОЇ ІНФРАСТРУКТУРИ

1.1 Поняття і характеристики систем критичної інформаційної інфраструктури

Системи критичної інформаційної інфраструктури є концентратами високої технологічності та організаційної складності. Ці системи інтегрують в себе найрізноманітніші інформаційні технології: серверні платформи, системи зберігання даних, мережеві пристрої, програмне забезпечення для обробки та аналізу даних, а також спеціалізовані засоби зв'язку [1]. Вони служать основною платформою для стабільного функціонування ключових сфер життєдіяльності суспільства, зокрема економіки, державного управління, оборони та систем безпеки.

Взаємодія між цими компонентами в СКІІ часто є високо автоматизованою і потребує надзвичайно високого рівня координації та синхронізації. Вони також характеризуються високим ступенем взаємозалежності. Отже, збій в одному компоненті може викликати ланцюгову реакцію, яка може призвести до глобальних негативних наслідків.

Через цю взаємозалежність та високий рівень складності, СКІІ вимагають спеціального підходу до забезпечення їх надійності та безпеки [2]. Сюди входять не лише традиційні методи кібербезпеки, але і більш комплексні системи управління ризиками, моніторингу, а також планування відновлення роботи після збоїв.

Також варто відзначити, що СКІІ є динамічними системами, які постійно адаптуються до нових технологічних тенденцій, законодавчих змін та еволюції кіберзагроз. Це ставить перед ними завдання не лише забезпечити стабільну роботу в сучасних умовах, але і мати можливість гнучкої адаптації до майбутніх викликів.

Для глибшого розуміння специфіки та викликів, які стоять перед СКІІ, необхідно уважно розглянути їх ключові характеристики. Ці характеристики не лише відзначають СКІІ як унікальні технічні та організаційні комплекси, але й підкреслюють їх критичну роль у сучасному суспільстві. Ось декілька найбільш значущих з них:

Комплексність

Комплексність СКІІ є складними комплексними системами, які включають в себе широкий спектр технологій, методологій та наукових дисциплін. Їх комплексність виявляється в тому, що вони є місцем перетину різних сфер технічної та організаційної діяльності [3]:

– мережеві протоколи. СКІІ використовують різноманітні мережеві протоколи для забезпечення комунікації між різними елементами системи. Це може включати стандартні протоколи, такі як TCP/IP, а також спеціалізовані протоколи для конкретних задач;

– системи управління базами даних. Для зберігання, обробки та доступу до критичної інформації часто використовуються різні СУБД. Ці системи можуть бути реляційними, нереляційними або є гібридами;

– телекомунікаційні технології. СКІІ можуть включати в себе різні телекомунікаційні засоби, від простих кабельних мереж до супутникових систем зв’язку;

– кіберфізичні системи. В деяких випадках, СКІІ інтегрують фізичні процеси з комп’ютерними системами, створюючи так звані кіберфізичні системи. Це додає ще один рівень складності у взаємодії та управлінні;

– програмне забезпечення та алгоритми. СКІІ включають різноманітні програмні рішення, які можуть вар’юватися від простих скриптів для автоматизації до складних алгоритмів штучного інтелекту для аналізу даних;

– організаційна структура. Управління СКІІ вимагає координації між різними департаментами і, можливо, навіть між різними організаціями та державними структурами.

Отже, комплексність забезпечує високий рівень складності СКІІ та вимагає інтегрованого підходу до їх проектування, розробки, управління та обслуговування. Кожна з цих дисциплін вносить свої виклики та можливості, які необхідно узгодити для забезпечення ефективного та безпечного функціонування СКІІ.

Стратегічна важливість

Стратегічна важливість СКІІ не може бути переоцінена. Ці системи є нерозривно пов'язані з ключовими сферами життєдіяльності суспільства – від економіки та охорони здоров'я до національної безпеки та державного управління. Отже, будь-яке серйозне порушення в їх функціонуванні може викликати катастрофічні наслідки на множині рівнів, а саме [4]:

– загроза життю та здоров'ю. Перш за все, зупинка функціонування критичних систем, таких як медичні реєстраційні системи або системи аварійного реагування, може безпосередньо загрожувати життю та здоров'ю громадян;

– економічна нестабільність. Порушення в роботі фінансових систем, систем управління постачанням або енергетичних мереж може спричинити широкомасштабні економічні розлади, які, у свою чергу, можуть вплинути на стабільність країни;

– національна безпека. Системи національної безпеки та оборони також є частиною СКІІ. Їхнє знешкодження або злам може підірвати обороноздатність країни, викликати масові соціальні рухи або навіть стати причиною військового конфлікту;

– соціальна дезінтеграція. На фоні порушення в роботі СКІІ може спостерігатися ріст недовіри до установ, паніки серед населення, що може призвести до соціальної нестабільності та дезінтеграції;

– політичні ризики. Вплив на виборчі системи, системи державного управління може мати далекосяжні політичні наслідки, включаючи зміну владних структур та втрату міжнародної репутації.

Враховуючи всі ці аспекти, стає зрозуміло, що стратегічна важливість СКІІ вимагає від працівників високого рівня відповідальності у їхньому управлінні, забезпеченні безпеки та постійному моніторингу. Така важливість також підкреслює необхідність наукових досліджень для розробки ефективних методів захисту та управління цими критично важливими системами.

Динамічність та адаптивність

СКІІ є не лише статичними конструкціями, але й динамічними системами, які мають здатність до адаптації та самоорганізації. Ця адаптивність є критично важливою, оскільки системи критичної інформаційної інфраструктури постійно зіштовхуються з різноманітними викликами, такими як [5]:

– нові технології. Швидкий розвиток технологій, таких як штучний інтелект, блокчейн або квантові комп’ютери, може змінювати архітектуру та функціональні можливості СКІІ. Системи повинні бути готові адаптуватися до цих нововведень;

– загрози безпеки. Ландшафт кіберзагроз непостійний. Нові види атак, вірусів та інших загроз постійно з’являються, вимагаючи від СКІІ швидкої адаптації та відновлення;

– законодавчі зміни. Зміни в законодавстві або нормативах можуть вимагати від СКІІ внесення коректив у їх роботу, структуру або політики безпеки;

– соціально-економічні фактори. Зміни в економічному кліматі, політична нестабільність або соціальні рухи також можуть впливати на роботу та стабільність СКІІ.

Динамічність та адаптивність є життєво необхідними для забезпечення робустності, надійності та довговічності СКІІ. Це також робить дослідження в цій області вкрай актуальним, зокрема в розробці алгоритмів адаптивного управління, методів швидкого виявлення нових загроз та систем автоматичного відновлення.

Високі вимоги до безпеки

Вимоги до безпеки в СКІІ є не лише високими, але і надзвичайно складними, оскільки ці системи є центральними елементами для функціонування ключових сфер суспільства. Захист інформації в СКІІ має бути комплексним і включати в себе наступні аспекти [6]:

– захист від несанкціонованого доступу. Системи повинні бути захищені від спроб несанкціонованого доступу до критичної інформації або системних ресурсів. Це може включати в себе біометричні системи аутентифікації, двофакторну верифікацію, зашифровані канали передачі даних та ін.;

– протидія втручанню в роботу системи. СКІІ повинні мати механізми для виявлення та протидії спробам зміни алгоритмів роботи, конфігурації або даних в системі;

– боротьба з відмовами в обслуговуванні (DDoS). Системи повинні бути стійкими до атак, що мають на меті перевантаження ресурсів і призвести до відмови в обслуговуванні;

– захист від розширених постійних загроз (APT). Це високоорганізовані та цілеспрямовані атаки, які можуть бути вкрай шкідливими і вимагають спеціалізованих методів захисту;

– інтеграція з іншими системами. Захист не повинен бути острівним; він повинен інтегруватися з іншими системами безпеки, щоб забезпечити цілісний підхід;

– аудит та моніторинг. Постійний аудит та моніторинг діяльності системи дозволяють вчасно виявляти потенційні загрози та забезпечують можливість швидкого реагування на інциденти безпеки.

Високі вимоги до безпеки СКІІ підкреслюють необхідність розробки нових методологій, алгоритмів та технологій, які можуть ефективно протистояти зростаючому спектру загроз. Це також робить актуальними наукові дослідження в області кібербезпеки, зокрема, в розробці адаптивних

систем захисту, машинного навчання для виявлення загроз та криптографічних методів захисту.

Комплексність управління

Управління СКІІ є вкрай складним процесом, що вимагає високоорганізованої координації дій на множині рівнів. Основні аспекти цієї комплексності включають [7]:

– множинність стейкхолдерів. У звичайних умовах, управління СКІІ здійснюється через взаємодію між різними структурними підрозділами в одній організації. Однак, часто інтереси та дії розподілені між кількома організаціями, державними установами, а іноді – між країнами;

– горизонтальна та вертикальна координація. Управління СКІІ потребує ефективної координації не лише між різними вертикальними рівнями управління (наприклад, від оперативного до стратегічного), але і горизонтально, між різними відділами або організаціями;

– нормативно-правова рамка. Координація дій має відбуватися в рамках встановлених законодавчих та нормативних актів, що часто можуть бути складними та суперечливими;

– технологічна інтеграція. Ефективне управління також вимагає технологічної сумісності між різними системами, що може бути складним завданням, особливо при використанні різних технологічних платформ і стандартів;

– адаптивність та гнучкість. У зв’язку з динамічною природою загроз та вимог до СКІІ, системи управління повинні бути гнучкими та адаптивними для відповіді на непередбачувані ситуації.

Комплексність управління СКІІ підкреслює не тільки технічні аспекти задачі, але і соціальні, організаційні та політичні фактори, які мають бути враховані. Це ставить перед науковою спільнотою виклики щодо розробки нових методів управління, які можуть ефективно координувати дії множини стейкхолдерів в складних та динамічно змінюваних умовах.

Розуміння характеристик та особливостей СКІІ є фундаментальним для розробки ефективних методів виявлення порушень коректного функціонування. Це, в свою чергу, відкриває широкі перспективи для комп'ютерно-математичного моделювання в цій області, що і є предметом даного дослідження.

1.2 Види загроз для систем виявлення порушень коректного функціонування систем критичної інформаційної інфраструктури

СКІІ є витоком для різноманітних загроз, які можуть мати дестабілізуючий вплив на ключові сектори суспільства. У контексті комп'ютерно-математичного моделювання системи виявлення порушень коректного функціонування СКІІ, важливо розглянути різні типи загроз:

Кібератака

Кібератака – це дія, спрямована на компрометацію інтегральності, доступності або конфіденційності інформаційних систем [8]. Це може включати в себе незаконний доступ до мереж, крадіжку даних, перешкоджання нормальному функціонуванню системи або ж інші маніпуляції, які завдають шкоди користувачам або організаціям. Кібератаки мають різноманітні форми та можуть бути здійснені за допомогою різних методів, від простих до високо складних.

До найбільш поширених типів загроз СКІІ можна віднести (рис. 1.1):

– відмова в обслуговуванні (DoS) - це атака, спрямована на перевантаження мережевих ресурсів або системи, з метою зробити їх тимчасово або постійно недоступними для легітимних користувачів [9]. Приклад: велика кількість синхронізованих запитів на веб-сервер, що призводить до його зупинки;

– розширені постійні загрози (APT) - це високоорганізовані, довготривалі атаки з метою крадіжки або модифікації конфіденційної інформації [10]. Приклад: зловмисна програма, що проникає в корпоративну мережу і збирає конфіденційні дані протягом тривалого періоду часу;

– міжсайтовий сценарій (XSS) - це атака, при якій зловмисний код впроваджується на веб-сайті, що призводить до крадіжки інформації користувачів або зміни налаштувань [11]. Приклад: ін'єкція зловмисного JavaScript-коду на веб-сторінці, який краде сесійні кукі користувачів;

– крадіжка ідентифікаційних даних (Phishing) - це атака, при якій зловмисники імітують відомі і довірені ресурси для отримання конфіденційних даних користувача [12]. Приклад: електронний лист від «банку» з проханням підтвердити свої банківські дані, який насправді є зловмисним;

– людина-по-середині (Man-in-the-Middle, MitM) - це атака, при якій зловмисник перехоплює або модифікує комунікацію між двома сторонами без їхнього відома [13]. Приклад: перехоплення та модифікація фінансових транзакцій між користувачем та банківським сервером.

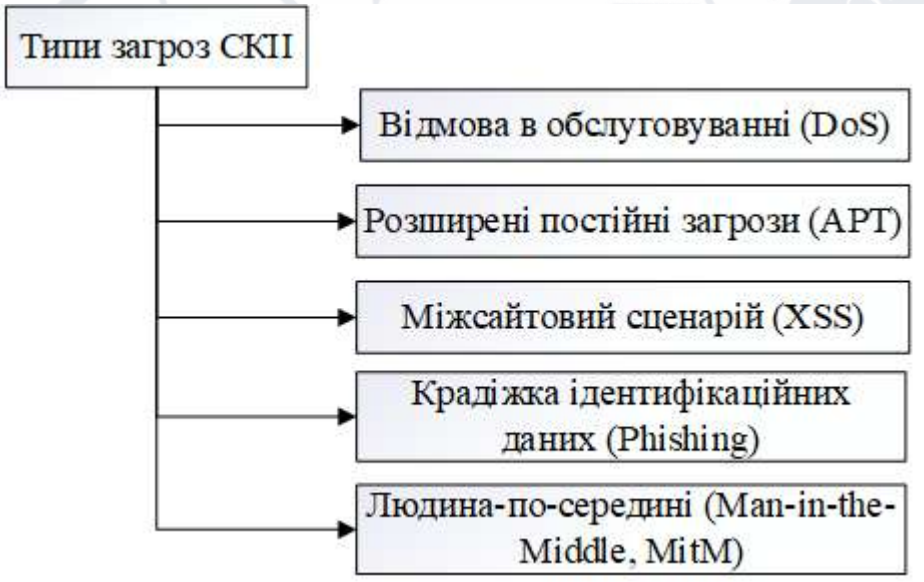

Рисунок 1.1 – Найбільш поширених типів загроз СКІІ

Розглянуті кібератаки, представляють собою різноманітні вектори загроз для СКІІ. Вони можуть мати різні рівні складності та потенційно катастрофічні наслідки, включаючи порушення нормального функціонування суспільно важливих служб, таких як енергетика, здоров'я та національна безпека. Врахування цих загроз є ключовим елементом в розробці

комп’ютерно-математичних моделей для ефективного виявлення та протидії кіберзагрозам в СКІІ.

Фізичні загрози

Фізичні загрози представляють собою реальні або потенційні події, що можуть завдати шкоди апаратній частині СКІІ, її розміщенню або персоналу. Це може включати в себе все від природних катастроф, таких як землетруси та повені, до злочинних дій, таких як вандалізм, крадіжка або терористичні акти [14]. Фізичні загрози вимагають особливої уваги в процесі планування та реалізації заходів безпеки для СКІІ.

До фізичних загроз відносяться (рис. 1.2):

– природні катастрофи – такі як землетруси, повені або урагани, можуть призвести до масштабних збоїв в роботі систем. Приклад: землетрус, що призвів до відключення електроенергії та збою в роботі дата-центрів;

– вандалізм – це умисне пошкодження або знищення фізичних компонентів системи. Приклад: руйнування серверного обладнання у фізичному дата-центрі;

– крадіжка обладнання – незаконне вилучення апаратних компонентів, що може призвести до витоку чутливої інформації. Приклад: крадіжка ноутбуків із конфіденційними даними з офісу компанії;

– терористичні акти – напади з метою завдання максимальної шкоди інфраструктурі та людям. Приклад: вибухи в стратегічних енергетичних об’єктах;

– незаплановані події – необережність або недбалість персоналу, що може призвести до фізичних пошкоджень системи. Приклад: випадкове коротке замикання через необережну роботу з електрообладнанням.

Рисунок 1.2– Фізичні загрози для КІІ

Фізичні загрози, такі як природні катастрофи, вандалізм, крадіжка обладнання, терористичні акти та споживчі акціденти, представляють серйозний ризик для стабільності та безпеки систем критичної інформаційної інфраструктури (СКІІ). Ці загрози можуть мати далекосяжні наслідки, включаючи втрату чутливої інформації, перерви в наданні життєво важливих послуг та потенційні загрози національній безпеки. Тому, разом із кібербезпекою, фізична безпека є ключовим аспектом в комплексному підході до захисту СКІІ.

Внутрішні загрози

Внутрішні загрози – це тип загроз безпеці, що виникають від дій або бездіяльності персоналу організації, партнерів або інших осіб, що мають законний доступ до системи [15]. Це може включати в себе ненавмисні дії, такі як помилкове видалення важливих даних, або умисні, такі як крадіжка чутливої інформації або саботаж. Внутрішні загрози представляють особливий ризик для СКІІ, оскільки зазвичай такі особи мають глибоке розуміння системи та можуть знати, як обійти заходи безпеки.

До таких загроз відносять (рис. 1.3):

– ненавмисний витік інформації - це ситуація, коли співробітник ненавмисно ділиться конфіденційною інформацією. Приклад: відправка документа із чутливими даними на особисту електронну адресу або через незахищений канал комунікації;

– інсайдерська зрада - це умисні дії співробітника, який має доступ до системи, з метою завдання шкоди організації. Приклад: завантаження вірусу на корпоративну мережу або крадіжка комерційної інформації;

– неналежне використання привілеїв - це зловживання правами доступу до системи. Приклад: адміністратор, який використовує свої привілеї для доступу до конфіденційних даних, не пов’язаних із його робочими обов’язками;

– технічні помилки співробітників - це допущення помилок при конфігурації або обслуговуванні систем, що може призвести до витоку інформації або збоїв в роботі системи. Приклад: неправильна настройка мережевого фаєрволу, що дозволяє несанкціонований доступ до мережі;

– соціальна інженерія - маніпуляція співробітниками для отримання доступу до систем або інформації. Приклад: інженер-зловмисник, який переконує співробітника віддати йому паролі під виглядом технічної підтримки.

Рисунок 1.3– Внутрішні загрози для КІІ

Внутрішні загрози, такі як ненавмисний витік інформації, інсайдерська зрада, неналежне використання привілеїв, технічні помилки та соціальна інженерія, становлять серйозний виклик для безпеки СКІІ. Оскільки ці загрози випливають з дій або бездіяльності осіб, які мають законний доступ до систем,

їх важко виявити та нейтралізувати за допомогою стандартних методів кібербезпеки.

Організаційні загрози

Організаційні загрози – це тип загроз, що виникають у результаті недоліків у структурі управління, процесах чи політиках в організації [16].

До організаційних загроз відносяться (рис. 1.4):

– недостатнє фінансування безпеки - це ситуація, коли організація не виділяє достатньо ресурсів на заходи безпеки. Приклад: відсутність необхідного обладнання для моніторингу мережі через обмежені бюджетні кошти;

– відсутність процедур реагування - нездатність ефективно відгукнутися на інциденти безпеки через відсутність чітких процедур. Приклад: неорганізована реакція на виявлення вірусу, що призводить до його поширення;

– слабка координація між відділами - недостатній обмін інформацією та ресурсами між різними структурними підрозділами. Приклад: відділ IT і відділ безпеки не координують свої дії для запобігання кібератакам;

– недостатня освіта та підготовка персоналу - відсутність тренінгів та освітніх програм для співробітників з питань безпеки. Приклад: співробітники, які не знають, як виявити спроби соціальної інженерії;

– компрометація постачальників - ризик компрометації безпеки через слабкі місця у постачальників або партнерів. Приклад: використання не аутентифікованого програмного забезпечення, що містить зловмисний код.

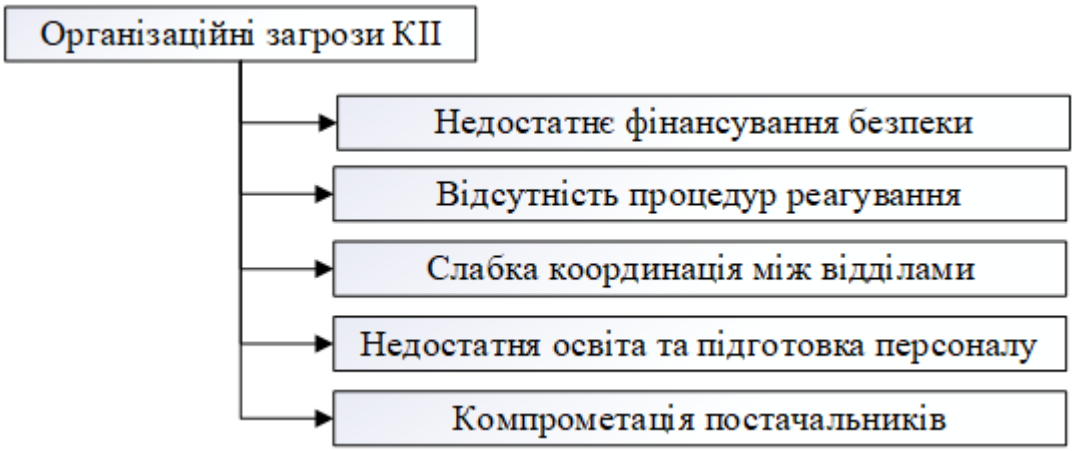

Рисунок 1.4– Організаційні загрози КІІ

Перелічені вище організаційні загрози представляють значний ризик для СКІІ. Ці фактори можуть послабити внутрішні механізми захисту і зробити системи більш вразливими до зовнішніх та внутрішніх загроз. Тому для ефективного управління ризиками в СКІІ необхідно розробляти комплексні стратегії, які враховують організаційний контекст і спрямовані на мінімізацію таких загроз.

Комп'ютерно-математичне моделювання дозволяє врахувати різноманітні параметри та сценарії, що забезпечує високий рівень адекватності в оцінці потенційних ризиків та їх впливу на стабільність і безпеку СКІІ.

1.3 Математичні моделі для виявлення порушень функціонування систем критичної інформаційної інфраструктури

Математичні моделі в контексті виявлення порушень в СКІІ є інструментами для формалізації процесів та механізмів, що забезпечують функціонування та безпеку таких систем. Ці моделі можуть бути розділені на декілька категорій залежно від їхньої складності, специфіки проблеми, яку вони розв'язують, та математичних методів, що в них використовуються.

Статистичні моделі

Статистичні моделі базуються на принципах статистичної теорії та математичної статистики, використовуючи розрахунки ймовірностей,

дисперсій, математичних очікувань та інших статистичних показників для аналізу великих наборів даних.

Однією з ключових характеристик статистичних моделей є їхня здатність адаптуватися до змінних умов, що є особливо актуальним для СКІІ, де динамічність та нестабільність є постійними факторами. Наприклад, моделі на основі теорії ймовірностей можуть використовувати історичні дані для створення «нормального» розподілу мережевого трафіку. Подальший аналіз поточних даних трафіку на відхилення від цього «нормального» розподілу може служити основою для виявлення аномальних патернів, що, у свою чергу, можуть вказувати на потенційні порушення безпеки.

Для виявлення аномалій в мережевому трафіку систем КІІ може бути застосована математична модель на основі Гауссового розподілу [17]. Спочатку визначаються середнє значення μ та коваріаційна матриця Σ снові існуючого набору даних:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad (1.1)$$

Після цього для кожного нового спостереження x_{new} обчислюється ймовірність на основі багатовимірного Гауссового розподілу:

$$p(x_{new}) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} exp\left(-\frac{1}{2}(x_{new}-\mu)^T \cdot (x_{new}-\mu)\right) \qquad (1.2)$$

Якщо обчислена ймовірність $p(x_{new})$ менша за порогове значення ϵ, спостереження вважається аномальним. Ця математична модель може бути інтегрована в системи критичної інформаційної інфраструктури для своєчасного виявлення аномалій та потенційних загроз.

Графові моделі

Графові моделі в контексті виявлення порушень у СКІІ представляють собою потужний математичний інструмент для аналізу структурних залежностей та взаємодії між різними компонентами системи [18]. У такому підході, кожна вершина графа символізує окремий компонент СКІІ (наприклад, сервер, маршрутизатор, базу даних тощо), а ребра між вершинами відображають зв’язки або залежності між цими компонентами.

Формально, графова модель може бути представлена як $G=(V,E)$, де V – множина вершин, а E – множина ребер. Ребра можуть мати ваги, які, наприклад, можуть відображати пропускну здатність зв'язку або ймовірність збою.

Однією з ключових особливостей графових моделей є можливість використання алгоритмів для виявлення шляхів збою, вузьких місць в системі або потенційних точок входу для атак. За допомогою таких алгоритмів, як пошук в глибину або в ширину, можна аналізувати структурну цілісність системи та визначати потенційні вектори загроз.

Графові моделі можуть бути вельми корисними для аналізу складних систем критичної інформаційної інфраструктури, оскільки вони дозволяють явно враховувати структурні залежності та взаємодії між компонентами, що є важливим для розробки ефективних стратегій забезпечення безпеки.

Гібридні моделі

Гібридні моделі в області виявлення порушень у системах КІІ представляють собою інтеграцію двох або більше типів математичних моделей з метою досягнення більшої точності, гнучкості та надійності. Зазвичай, така комбінація здійснюється для компенсації обмежень окремих моделей та для забезпечення більшої адаптивності до динамічно змінюваних умов роботи СКІІ [19].

У контексті гібридних моделей математична модель може бути досить складною, оскільки вона комбінує елементи різних типів моделей. Однак, можна спростити ситуацію та розглянути гібридну модель, яка комбінує статистичний метод (наприклад, метод на основі гауссового розподілу) і графову модель.

Нехай є система з N компонентами, яку можна представити у вигляді графа $G=(V,E)$, де V – множина вершин (компонентів), а E – множина ребер (зв'язків між компонентами).

Для кожної вершини $v \in V$ можна визначити ймовірність $p(v)$ її «нормальної» поведінки на основі історичних даних. Тобто, для кожного компонента v, можна визначити ймовірність $p(v)$ як:

$$p(v) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \qquad (1.3)$$

де μ та σ – середнє значення та стандартне відхилення характеристик компонента v.

Графову частину моделі можна визначити як «загальний стан» системи, тобто як функцію стану її компонентів та зв'язків між ними. Наприклад, ця функція може бути ваговою сумою станів вершин і ребер:

$$S(G) = \sum_{v \in V} w_v \cdot p(v) + \sum_{e \in E} w_e \cdot p(e), \qquad (1.4)$$

де w_v і w_e – це ваги, що відображають важливість конкретного компонента або зв'язку в системі, а $p(e)$ – ймовірність «нормальної» поведінки ребра e.

Комбінація цих двох частин може виразитися через функцію $S(G)$, яку можна використовувати для оцінки загального стану системи. Якщо $S(G)$ падає нижче певного порогового значення є, це може вказувати на потенційне порушення в системі.

Цей підхід дозволяє забезпечити більш гнучке та надійне виявлення порушень, адаптуючись до різних типів змін та загроз в динамічно змінюваному середовищі систем критичної інформаційної інфраструктури.

Часові ряди

Моделі часових рядів зосереджуються на аналізі змін показників системи в часі для виявлення можливих аномалій або порушень [20]. Для виявлення порушень, часові ряди дозволяють зосередити увагу на аналізі показників системи в часі, враховуючи часткову або циклічну залежність між спостереженнями. Основна мета такого аналізу – виявлення аномалій або порушень, які можуть відрізнятися від встановленого паттерну поведінки системи.

Математично, часовий ряд X_t може бути моделей за допомогою авторегресійного процесу, наприклад, ARIMA (Авторегресійна інтегрована модель з ковзним середнім), де:

$$X_t = c + \varphi_1 X_{t-1} + \varphi_2 X_{t-2} + \cdots + \varphi_p X_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t \qquad (1.5)$$

де c – константа, ϕ і θ – параметри моделі, p та q – порядки авторегресії та ковзного середнього відповідно, і ϵ_t – білий шум з нульовим середнім та стандартним відхиленням σ.

Використовуючи такі моделі, можна визначити межі «нормальної» поведінки системи і, отже, виявляти відхилення, які можуть свідчити про порушення. Для цього часто використовуються статистичні методи, такі як Z-оцінка, для калібрування порогових значень на основі історичних даних. Такий підхід є особливо корисним для систем з високою динамічністю та складними взаємозв’язками між компонентами.

Кожна з цих моделей має свої переваги та обмеження, і вибір конкретної моделі залежить від специфіки задачі, доступності даних та ресурсів для обчислень. Застосування цих моделей в СКІІ дозволяє не лише ефективно виявляти потенційні порушення, але і проводити аналіз ризиків, оптимізацію ресурсів та планування заходів з підвищення безпеки.

1.4 Методи та засоби виявлення порушень функціонування систем критичної інформаційної інфраструктури

Ефективне виявлення порушень у СКІІ вимагає використання комплексу методів та технічних засобів. В контексті постійно зростаючої кількості інформаційних загроз, такий комплексний підхід стає не просто бажаним, але й критично необхідним. Важливість цього акцентується також тим, що СКІІ відіграють ключову роль у функціонуванні сучасного суспільства, від економіки і здоров’я до національної безпеки. Тому важливо зосередити увагу

на методах та засобах, які можуть ефективно виявляти та протидіяти таким порушенням.

Моніторинг мережевого трафіку

Моніторинг мережевого трафіку передбачає збір, аналіз та інтерпретацію мережевих пакетів у реальному часі або офлайн. Основна мета цього методу полягає в ідентифікації аномальних патернів або підозрілих активностей, які можуть сигналізувати про потенційні порушення безпеки або відмінності в роботі системи [21].

Однією з математичних моделей, що використовуються в моніторингу мережевого трафіку, є статистичний аналіз розподілу довжин пакетів, частоти запитів до певних IP-адрес або портів. Зазвичай використовують методи кластерного аналізу чи аналізу основних компонент для виявлення аномалій. Математично це може бути представлено як:

$$\text{Аномалія} = f(X_1, X_2, \dots, X_n) > \theta \qquad (1.6)$$

де f – функція, що оцінює «нормальність» трафіку на основі ряду параметрів $X_1, X_2,\dots,X_n$ (наприклад, довжина пакетів, частота запитів тощо), а θ – порогове значення, вище якого трафік вважається аномальним.

Ефективність моніторингу мережевого трафіку може значно підвищити рівень безпеки СКІІ, виявляючи не лише зовнішні, але і внутрішні загрози, що є критично важливим для забезпечення стабільності і надійності функціонування таких систем.

Інтрузійні системи виявлення

Інтрузійні системи виявлення (IDS) представляють собою спеціалізовані механізми, розроблені для моніторингу і аналізу подій в комп'ютерних системах та мережах з метою виявлення ознак несанкціонованого доступу або аномальної поведінки. В контексті СКІІ, ефективність IDS є вкрай важливою, оскільки ризик вторгнення може мати серйозні наслідки, впливаючи на національну безпеку, здоров'я населення та стабільність економіки [22].

IDS використовують різноманітні методи для аналізу даних, включаючи статистичний аналіз, патерн-матчинг та евристичні алгоритми. Один із

математичних підходів полягає в використанні машинного навчання для класифікації мережевого трафіку. Наприклад, можна використовувати метод опорних векторів (SVM) для визначення аномальних патернів в мережевому трафіку:

$$f(x) = sgn\left(\sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b\right) \qquad (1.7)$$

де *f(x)* – функція, що визначає, чи є даний пакет аномальним, x – вектор ознак мережевого пакету, *N* – кількість тренувальних даних, α_i – коефіцієнти, які визначаються під час тренування, y_i – мітки класів, $K(x_i,x)$ – ядро, b – зсув.

Застосування IDS у СКІІ дозволяє виявляти та нейтралізувати загрози на ранніх стадіях, забезпечуючи тим самим вищий рівень захисту для критично важливих систем.

Системи управління подіями

Системи управління подіями в інформаційній безпеці (SIEM) представляють собою інтегровані рішення, що здійснюють збір, нормалізацію, кореляцію та аналіз логів та інших даних з різних компонентів інформаційної системи. Важливість такого комплексного підходу полягає в здатності виявляти складні, багатоетапні атаки, які можуть бути непомітними при використанні менш інтегрованих механізмів виявлення загроз [23].

В контексті СКІІ, SIEM відіграє ключову роль, оскільки такі системи часто мають розподілену архітектуру та інтеграцію з різними платформами та технологіями. Математична модель SIEM може включати в себе алгоритми машинного навчання для класифікації подій та їх кореляції. Один з підходів – використання байєсівської мережі для оцінки ймовірності конкретної загрози на основі вхідних подій:

$$P(A|X_1, X_2, \dots, X_n) = \frac{P(X_1, X_2, \dots X_n|A) \cdot P(A)}{P(X_1, X_2, \dots X_n)}, \qquad (1.8)$$

де $P(A|X_1, X_2, \dots, X_n)$ – умовна ймовірність наявності загрози A при заданих подіях $X_1, X_2, \dots, X_n$.

Застосування SIEM дозволяє не тільки ефективно виявляти потенційні загрози, але і забезпечує можливість швидкої реакції на них, що є критично важливим для забезпечення надійності та безпеки СКІІ.

Аналіз поведінки користувачів

Аналіз поведінки користувачів (АПК) – це методологія, що застосовує алгоритми машинного навчання, статистичні моделі та інші високоавтоматизовані техніки для моніторингу та аналізу дій користувачів у реальному часі або в історичній перспективі. Головна мета– виявлення аномалій в поведінці, які можуть свідчити про зловмисні дії або інші порушення політик безпеки [24].

В контексті систем критичної інформаційної інфраструктури, АПК є особливо значущим, оскільки несанкціоновані дії від співробітників можуть мати катастрофічні наслідки. Цей метод включає в себе збір та аналіз метрик, таких як частота входу в систему, об’єми переданих даних, використані команди та інші параметри.

Один із способів математичної моделі АПК може бути заснований на алгоритмах кластеризації, як-от k-середніх. Для кожного користувача формується вектор ознак на основі його поведінки, а потім відбувається кластеризація цих векторів. Аномальна поведінка виявляється як викид в просторі ознак:

$$d(x, C) = \sqrt{\sum_{i=1}^{n}(x_i - c_i)^2}, \qquad (1.9)$$

де $d(x, C)$ – відстань від точки x до центру кластера C, x_i та c_i – i-та ознака вектора x та центра кластера C відповідно.

Такий підхід дозволяє не тільки виявляти потенційні внутрішні загрози, але і адаптуватися до змін у поведінці користувачів, що робить метод ефективним для забезпечення безпеки в динамічних умовах експлуатації СКІІ.

Гібридні моделі

Гібридні моделі представляють собою інтегрований підхід до виявлення порушень, який комбінує елементи двох або більше методологій, таких як

статистичний аналіз, машинне навчання, графові моделі та інші. Ця комбінація дозволяє забезпечити більшу точність, надійність та гнучкість системи виявлення порушень, зменшуючи при цьому кількість помилкових спрацьовувань та «сліпих зон» [25].

В СКІІ, використання гібридних моделей є особливо актуальним. СКІІ характеризуються високою складністю, динамічністю та стратегічною важливістю, тому виявлення порушень у таких системах вимагає багаторівневого та багатоаспектного підходу.

Математично, гібридна модель може бути представлена як взаємодія між кількома моделями. Наприклад, можна використовувати статистичні моделі для виявлення аномалій в мережевому трафіку, а паралельно застосовувати моделі машинного навчання для аналізу поведінки користувачів. Рішення про наявність порушення може прийматися на основі агрегованої оцінки:

$$S = w_1 \cdot S_1 + w_2 \cdot S_2 + \cdots + w_n \cdot S_n \qquad (1.10)$$

де S – загальна оцінка наявності порушення, S_i – оцінка від i-ї моделі, w_i – ваговий коефіцієнт i-ї моделі.

Такий підхід дозволяє оптимізувати процес виявлення порушень, адаптуючи систему до специфіки та динаміки СКІІ.

У контексті СКІІ, розглянуті методи та засоби виявлення порушень функціонування представляють собою комплексний набір інструментів для забезпечення інформаційної безпеки. Вони включають моніторинг мережевого трафіку, інтрузійні системи виявлення, системи управління подіями в інформаційній безпеці, аналіз поведінки користувачів та гібридні моделі.

1.5 Використання апарату нейронних мереж для забезпечення коректного функціонування систем критичної інформаційної інфраструктури

Нейронні мережі представляють собою високоефективний інструмент для аналізу великих обсягів даних та виявлення складних закономірностей. У СКІІ, вони можуть бути використані для створення розширених систем безпеки, які здатні адаптуватися до нових типів загроз та атак.

Рекурентні нейронні мережі (РНМ) представляють собою клас нейронних мереж, що ефективно працюють з послідовними даними (рис. 1.5). Вони знаходять застосування в задачах, де необхідно враховувати часові або послідовні залежності, як-от аналіз мережевого трафіку в СКІІ.

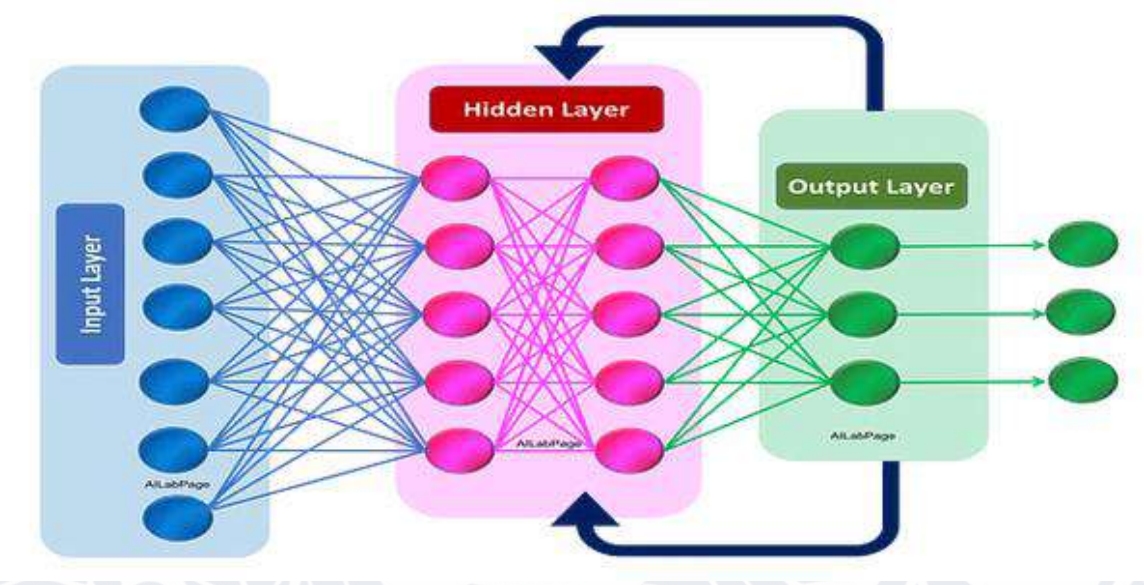

Рисунок 1.5 – Представлення рекурентної нейронної мережі [26]

Основна архітектура РНН складається з вхідного шару, одного або кількох прихованих шарів та вихідного шару. Ключовою особливістю РНН є наявність зворотнього зв’язку в прихованих шарах, який дозволяє мережі «запам’ятовувати» інформацію з попередніх кроків в послідовності. Математично, робочий процес РНН можна описати наступними рівняннями:

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \qquad (1.11)$$

$$y_t = W_{hy}h_t + b_y \qquad (1.12)$$

де, h_t – стан прихованого шару на часовому кроці t, x_t – вхід на часовому кроці t, y_t – вихід на часовому кроці t; W_{hh}, W_{xh}, W_{hy} – матриці ваг; b_h, b_y – вектори зсувів; σ – функція активації (зазвичай тангенс гіперболічний або ReLU).

Однак, стандартні РНН стикаються з проблемою «затухання градієнтів» при обробці довгих послідовностей [27]. Це обмежує їхню спроможність запам'ятовувати інформацію на протязі тривалого періоду часу. Для вирішення цієї проблеми були створені модифікації РНН, такі як «довга короткочасна пам'ять» (ДКП) та блоки з керованими рекурентними з'єднаннями (БКРЗ), які містять спеціалізовані механізми для керування пам'яттю.

В СКІІ, РНН, зокрема ДКП та БКРЗ, можуть бути використані для аналізу мережевого трафіку з метою виявлення аномалій та можливих кібератак. Це забезпечує високу ефективність у виявленні складних шаблонів атак та аномалій, що є критично важливим для забезпечення безпеки та надійності СКІІ.

Також можна розглянути багатошаровий персептрон (БП) для забезпечення коректного функціонування систем КІІ, який є розширенням одношарового персептрона і складається з вхідного шару, одного або кількох прихованих шарів та вихідного шару нейронів (рис. 1.6). Кожен нейрон у мережі обчислює лінійну комбінацію вхідних сигналів, яка потім проходить через нелінійну функцію активації. У контексті систем КІІ, БП може бути використаний для виявлення аномалій та потенційних загроз на основі аналізу вхідних даних, таких як мережевий трафік або системні журнали. Завдяки своїй здатності вивчати складні залежності в даних, цей тип нейронної мережі дозволяє підвищити ефективність системи безпеки, забезпечуючи високий рівень виявлення аномалій та мінімізацію помилкових спрацювань.

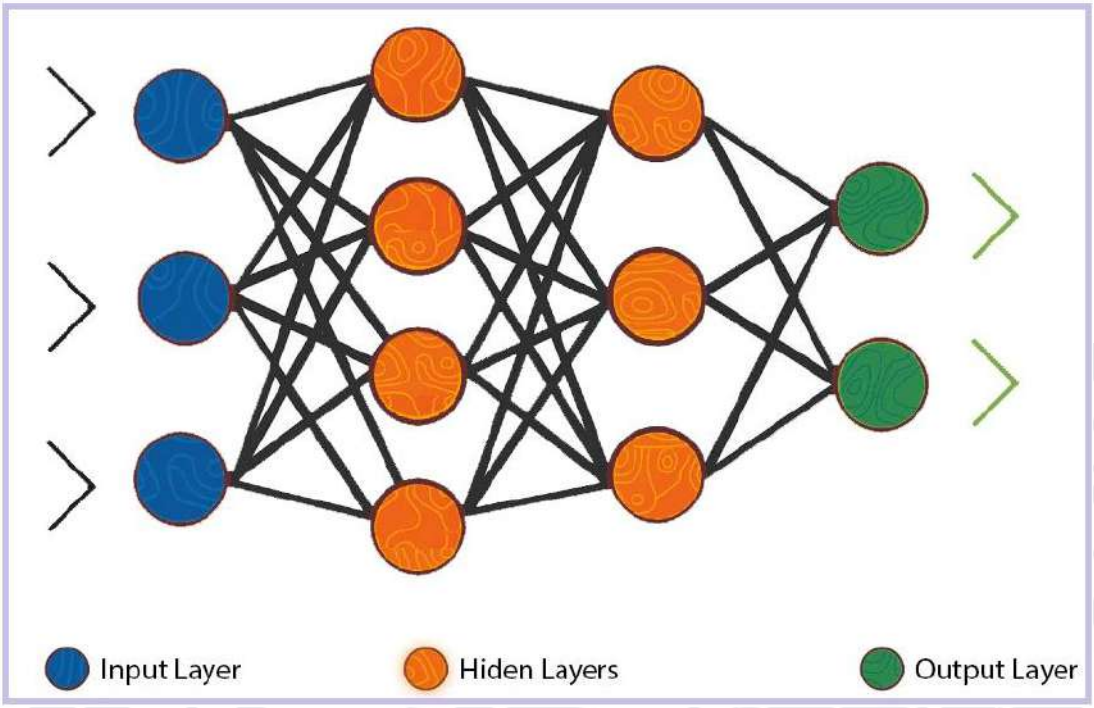

Рисунок 1.5 – Представлення багатошарового персептрона [28]

Математична модель БП базується на концепції шарів нейронів та їх взаємозв'язків. БП складається з трьох основних типів шарів: вхідного, одного або декількох прихованих, та вихідного шару.

Для кожного нейрона в шарі i лінійна комбінація вхідних сигналів обчислюється за формулою:

$$z^{(i)} = W^{(i)} a^{(i-1)} + b^{(i)} \qquad (1.13)$$

де, $z^{(i)}$ – вектор зважених сум для шару i;

$W^{(i)}$ – матриця ваг між шаром i−1 та i;

$a^{(i-1)}$ – вектор активацій нейронів у шарі i−1;

$b^{(i)}$ – вектор зміщень для шару i.

Після обчислення зваженої суми, вхідний сигнал проходить через функцію активації σ, що дає вихідний сигнал (активацію) для кожного нейрона:

$$a^{(i)} = \sigma\left(z^{(i)}\right) \qquad (1.14)$$

Початковий вхідний сигнал $a^{(0)}$ встановлюється рівним вхідному вектору x даних.

$$a^{(0)} = x \qquad (1.15)$$

Для кожного нейронного шару *i*, починаючи з першого прихованого шару *i=1* і до вихідного шару *i=n*, виконуються наступні операції:

$$a^{(i)} = \sigma\left(W^{(i)}a^{(i-1)} + b^{(i)}\right) \qquad (1.16)$$

Завершальний вихідний сигнал $a^{(n)}$ представляє результат роботи нейронної мережі для даного вхідного вектора *x*.

Наступним кроком є застосування алгоритму зворотного поширення помилки (Backpropagation). Цей алгоритм використовується для оптимізації параметрів W та b на основі градієнтного спуску. Функція втрат *J(W,b)* мінімізується шляхом ітеративного оновлення ваг та зміщень:

$$W^{(i)} \coloneqq W^{(i)} - a\frac{\partial J}{\partial W^{(i)}} \qquad (1.17)$$

$$b^{(i)} \coloneqq b^{(i)} - a\frac{\partial J}{\partial b^{(i)}} \qquad (1.18)$$

де α – швидкість навчання.

Така математична модель багатошарового персептрона є фундаментом для багатьох застосувань в області аналізу даних, включаючи задачі забезпечення безпеки систем критичної інформаційної інфраструктури.

Математична модель багатошарового персептрона дозволяє моделювати складні залежності в даних та виявляти аномальні патерни, що можуть свідчити про потенційні загрози або вразливості [29]. Зокрема, багатошаровий персептрон може бути натренований для аналізу мережевого трафіку, системних журналів або інших оперативних параметрів для виявлення нестандартної поведінки, що виходить за межі визначених безпечних норм. Це, в свою чергу, дозволяє оперативно ідентифікувати та нейтралізувати ризики, забезпечуючи коректне функціонування систем КІІ. Отже, впровадження багатошарового персептрона в системи забезпечення безпеки КІІ може служити ефективним інструментом для підвищення їх надійності та стійкості до зовнішніх та внутрішніх загроз.

Висновок до першого розділу

В рамках даного розділу було проведено глибокий аналіз ключових аспектів, що впливають на коректність функціонування систем критичної інформаційної інфраструктури. Було розглянуто і класифіковано різноманітні види загроз, що можуть впливати на СКІІ, включаючи фізичні, внутрішні та організаційні. Детально були розглянуті математичні моделі для виявлення аномалій та порушень в СКІІ, серед яких статистичні моделі, графові моделі та моделі на основі часових рядів.

Особливу увагу було приділено методам та технічним засобам для виявлення порушень, в тому числі моніторингу мережевого трафіку, інтрузійним системам виявлення та системам управління подіями в інформаційній безпеці. Було розглянуто використання нейронних мереж для аналізу мережевого трафіку і виявлення аномалій, зокрема використання рекурентних нейронних мереж та багатошарового персептрона в контексті систем критичної інформаційної інфраструктури. Моделі нейронної мережі є доволі ефективними для задач виявлення аномалій та нестандартної поведінки в системах, забезпечуючи високу точність та надійність в реальних умовах. Їх впровадження дозволяє підвищити рівень безпеки, зменшивши кількість помилкових спрацювань та забезпечивши швидке реагування на потенційні загрози.

2 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ ВИЯВЛЕННЯ ПОРУШЕНЬ КОРЕКТНОГО ФУНКЦІОНУВАННЯ СИСТЕМ КРИТИЧНОЇ ІНФОРМАЦІЙНОЇ ІНФРАСТРУКТУРИ

2.1 Формалізація задачі

Метою даної роботи є розробка комп'ютерно-математичної моделі для виявлення порушень коректного функціонування систем КІІ Зокрема, необхідно ідентифікувати аномальні патерни в трафіку CoAP (протокол обмеженого застосування), який часто використовується в IoT-мережах.

Вихідні дані для навчання та тестування моделі будуть взяті з публічно доступного датасету на платформі «Kaggle» [30]. Датасет містить зразки даних для ідентифікації DDoS-атак в CoAP-трафіку. Це робить його цінним для моделювання системи виявлення порушень, оскільки DDoS-атаки є одними з найбільш поширених та небезпечних видів порушення в системах КІІ.

Для навчання моделі виявлення порушень необхідний певний перелік параметрів, представлених у табл. 2.1.

Таблиця 2.1 – Вхідні дані для проведення навчання моделі

Назва параметру	Тип даних	Розмір
Ідентифікатор повідомлення CoAP	float	32 біт
Код операції CoAP	float	32 біт
Опис опцій CoAP	float	32 біт
Запит місцезнаходження CoAP	float	32 біт
Максимальний вік CoAP	float	32 біт
Хост URI CoAP	float	32 біт
Відгук у CoAP	float	32 біт
Передача повторно CoAP	float	32 біт
Токен CoAP	float	32 біт
Довжина токену CoAP	float	32 біт
Мітка порушення безпеки	Bool	1 біт

Вихідними даними системи є прогноз наявності порушення та його класифікація. Результати представлені в табл. 2.2.

Таблиця 2.2 – Вихідні дані моделі виявлення порушень

Назва параметру	Тип даних	Розмір
Прогнозоване порушення	bool	8 біт
Ймовірність порушення	float	32 біт

Завдяки використанню машинного навчання, система зможе вчасно ідентифікувати потенційні порушення та попереджувати адміністраторів системи для своєчасного вжиття заходів. Це підвищує ефективність системи безпеки та зменшує ризик компрометації критичних ресурсів.

Така формалізація задачі дозволяє систематизувати процес виявлення порушень та підвищити його ефективність за рахунок використання методів машинного навчання.

2.2 Аналіз та вибір методу навчання

Для реалізації комп’ютерно-математичного моделювання системи виявлення порушень у системах критичної інформаційної інфраструктури необхідно вибрати оптимальний метод навчання. У контексті задачі ідентифікації DDoS-атак в CoAP-трафіку можна проаналізувати наступні методи: Бройдена-Флетчера-Гольдфарба-Шанно, опорних векторів та випадковий ліс.

Метод Бройдена-Флетчера-Гольдфарба-Шанно

Метод Бройдена-Флетчера-Гольдфарба-Шанно (МБФГШ англ. Limited-memory Broyden-Fletcher-Goldfarb-Shanno) – це алгоритм оптимізації, що призначений для розв’язання задач оптимізації нев’язки [31]. Він є ефективною адаптацією алгоритму МБФГШ для великих обсягів даних. У контексті логістичної регресії МБФГШ використовується для знаходження набору параметрів, які мінімізують функцію втрат (англ. loss function).

Логістична регресія – це статистичний метод, використовуваний для аналізу набору даних, в якому є одна або кілька незалежних змінних, що впливають на результат в формі категорії (зазвичай дві категорії: 0 і 1). В контексті машинного навчання це відноситься до бінарної класифікації.

Основна ідея МБФГШ полягає у використанні обмеженої кількості пам’яті для зберігання інформації про останній градієнт функції втрат, що дає можливість виконувати швидкі ітерації навіть для великих наборів даних.

Принцип роботи методу МБФГШ зображено на рис. 2.1.

Рисунок 2.1 – Алгоритм роботи методу МБФГШ

Алгоритм роботи методу МБФГШ розпочинається з ініціалізації параметрів, таких як критерії зупинки та максимальна кількість ітерацій. Після цього вибирається початкова точка в просторі рішень. На кожному кроці

перевіряється, чи поточна точка задовольняє критеріям зупинки. Якщо так, алгоритм завершує роботу.

В іншому випадку, обчислюється градієнт цільової функції в поточній точці. За допомогою цього градієнта та інформації з попередніх кроків обчислюється напрямок для наступного кроку. Величина кроку визначається, і поточна точка оновлюється відповідно до обраного напрямку та величини кроку.

Після цього параметри методу оновлюються на основі нової точки та градієнта. Цикл оптимізації продовжується, поки не буде досягнуто заданих критеріїв зупинки.

МБФГШ може бути особливо корисною в задачах виявлення аномалій та безпеки, таких як виявлення DDoS-атак. Це забезпечує швидку і точну класифікацію за великих обсягів даних, що є критично важливим в реальному часі для систем критичної інформаційної інфраструктури.

Отже, МБФГШ є потужним алгоритмом для розв’язання задач бінарної класифікації на великих датасетах, і він може бути ефективно використаний у сфері кібербезпеки для виявлення аномальної поведінки та DDoS-атак.

Метод опорних векторів

Метод опорних векторів (МОВ анг. Support Vector Machines) є однією з ключових технік у машинному навчанні, зокрема в задачах класифікації та регресії [32]. Цей метод був розроблений у 1990-х роках і є одним із найбільш надійних і найбільш часто використовуваних методів для розв’язання задач машинного навчання.

Основна ідея методу опорних векторів полягає в тому, щоб знайти таку гіперплощину (або розділяючу межу) в просторі ознак, яка найкраще розділяє два класи даних. «Найкраще» в даному контексті означає, що гіперплощина максимізує відстань між найближчими до неї точками різних класів. Ці «найближчі точки» називаються «опорними векторами».

Алгоритм роботи МОВ представлено на рис. 2.2 у вигляді діаграми діяльності.

Рисунок 2.2 – Алгоритм роботи методу МОВ

Метод опорних векторів ініціюється з налаштування ряду параметрів, таких як регуляризаційний параметр і тип ядра. Після цього, навчальні і тестові дані завантажуються в систему. Для забезпечення кращої точності моделі, дані можуть бути нормалізовані або стандартизовані.

В залежності від природи задачі і даних, вибирається тип ядра, яке може бути лінійним або нелінійним. За допомогою цього ядра обчислюється матриця ядра, що представляє взаємозв’язки між різними точками даних. Оптимізаційна задача формується на основі цієї матриці ядра і міток класів для знаходження оптимальних ваг, які максимізують відстань між різними класами в перетвореному просторі.

За допомогою методів оптимізації розв’язується ця задача, і як результат, отримуються оптимальні ваги. Опорні вектори визначаються як ті

точки навчального набору, які мають ненульові ваги в розв’язку оптимізаційної задачі. Ці опорні вектори потім використовуються для класифікації нових даних. Клас нової точки даних визначається на основі знаку скалярного добутку між вектором нової точки і опорними векторами, зваженими на їх оптимальні ваги.

Важливим розширенням методу опорних векторів є введення так званих ядерних функцій (ядер), які дозволяють ефективно працювати в просторах вищої розмірності без необхідності явного переходу в цей простір. Це дозволяє розділяти даних, які не є лінійно роздільними в початковому просторі ознак.

Отже, метод опорних векторів є потужним інструментом для розв’язання складних задач класифікації та регресії і широко використовується в різних доменах, від обробки зображень до фінансового аналізу.

Метод випадкового лісу

Метод випадкового лісу (МВЛ) є ансамблевою технікою машинного навчання, яка базується на конструкції кількох дерев рішень для вирішення задач класифікації та регресії [33]. Цей метод відзначається високою точністю, можливістю ефективної роботи з великими наборами даних і високою здатністю до адаптації до різних типів задач. У контексті виявлення порушень в системах критичної інформаційної інфраструктури, метод випадкового лісу може знайти широке застосування.

Метод випадкового лісу генерує ансамбль (колекцію) дерев рішень, кожне з яких навчається на підмножині вхідних даних. Під час процедури вибору рішення, результати всіх дерев агрегуються (наприклад, за принципом голосування для задач класифікації або середнього для задач регресії) для отримання кінцевого рішення.

Алгоритм роботи МВЛ представлено на рис. 2.3 у вигляді діаграми діяльності.

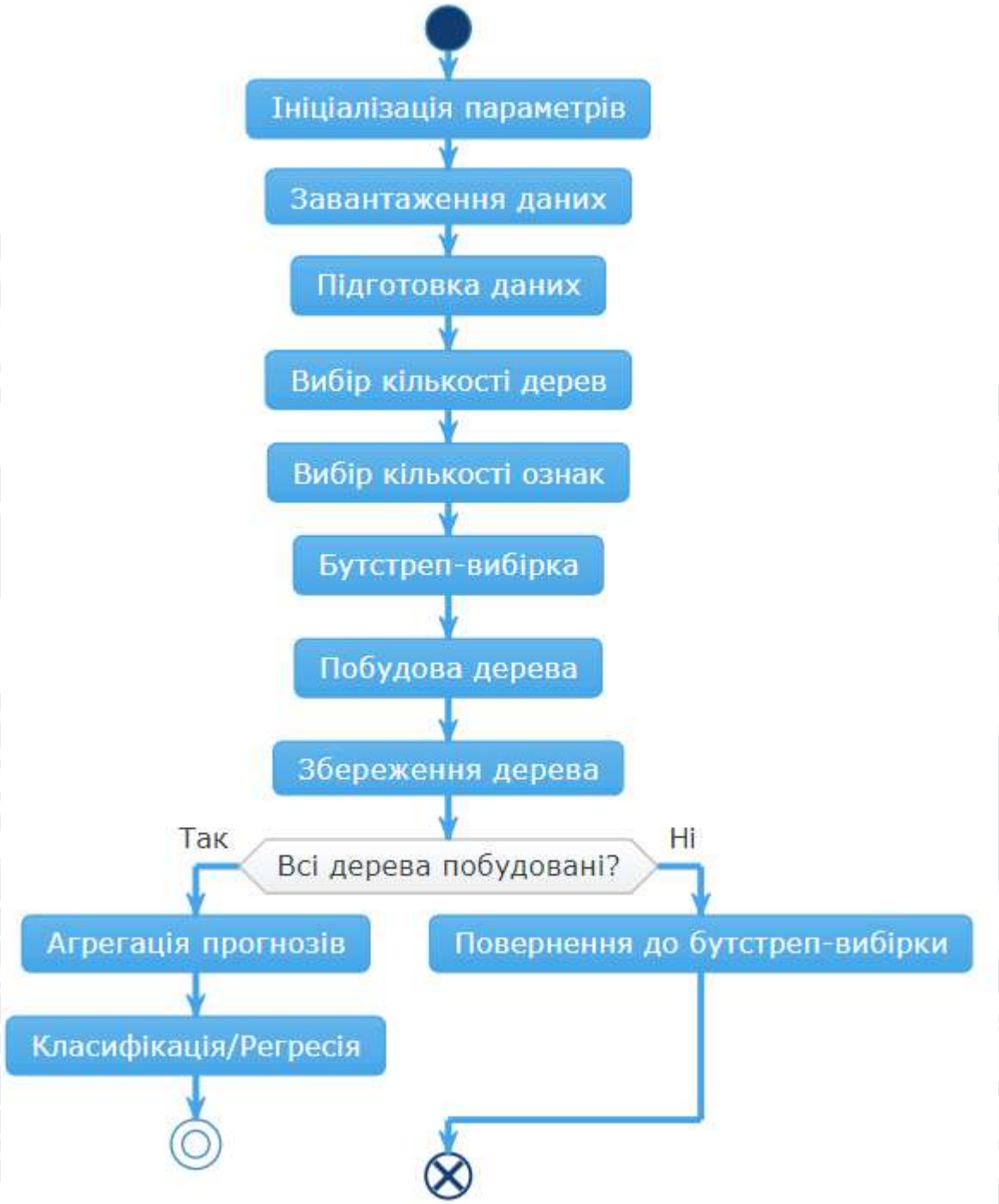

Рисунок 2.3 – Алгоритм роботи методу МВЛ

Метод випадкового лісу є ансамбльовим методом машинного навчання, який комбінує прогнози з множини дерев рішень для вирішення задач класифікації та регресії. Першим кроком є вибірка підмножини з навчального набору даних для кожного дерева, зазвичай за методом вибірки з поверненням, відомим як «початкова вибірка». Після вибірки підмножини даних, кожне дерево рішень будується незалежно від інших, використовуючи випадковий вибір підмножини ознак для кожного розгалуження в дереві. Це додає

додатковий рівень випадковості до моделі, що допомагає зменшити перенавчання.

Після побудови всіх дерев, їхні прогнози агрегуються для формування кінцевого рішення. У випадку класифікації, це зазвичай відбувається шляхом голосування більшістю, де клас, який отримав найбільшу кількість «голосів» від дерев, вибирається як кінцевий. У випадку регресії, середнє або медіана прогнозів з усіх дерев використовується як кінцевий прогноз.

Метод випадкового лісу є однією з найбільш потужних технік в області машинного навчання для виявлення аномалій та порушень в роботі систем. Цей метод особливо ефективний при роботі з великими та різнорідними наборами даних, що можуть включати не лише мережеві показники, але і шаблони користувацької поведінки, логи дій, статистичні дані та інші типи інформації.

Важливою перевагою методу є його здатність враховувати взаємодії між різними ознаками. Це підвищує ефективність виявлення складних шаблонів порушень, які можуть бути неочевидними при аналізі окремих ознак. Завдяки цьому метод випадкового лісу може виявляти навіть тонкі та складні порушення в системах, що є критично важливим для застосувань в системах критичної інформаційної інфраструктури.

Його ансамблева структура, яка базується на комбінації прогнозів з множини дерев рішень, дозволяє досягти високої точності та надійності в прогнозуванні. Кожне дерево в ансамблі «голосує» за деякий клас або видає прогноз, і кінцевий результат формується на основі агрегації цих «голосів».

Таким чином, метод випадкового лісу не лише здатен ефективно обробляти великі та різнорідні датасети, але і може враховувати складні взаємозв’язки між ознаками.

Проаналізувавши роботу методів також потрібно порівняти їх за основними характеристиками для розробки системи виявлення порушень. У табл. 2.3 проведено порівняння цих методів.

Таблиця 2.3 – Порівняння методів машинного навчання

Характеристика	МБФГШ	МОВ	МВЛ
Швидкість навчання	Висока (зокрема для великих наборів даних)	Середня або низька (залежить від ядра та розміру даних)	Висока (але може бути повільнішим при великій кількості дерев)
Складність моделі	Лінійна	Вибір ядра може додати складність	Нелінійна
Регуляризація	Вбудована	Зазвичай потребує налаштувань	Зазвичай не потрібна
Пам’ять	Ефективне використання пам’яті	Може вимагати більше пам’яті для великих наборів даних	Вимагає більше пам’яті при збільшенні кількості дерев
Стійкість до шуму	Висока	Залежить від налаштувань регуляризації	Зазвичай висока, але може перенавчитися
Масштабованість	Добре масштабується для великих наборів даних	Погано масштабується для великих наборів даних	Добре масштабується, але з обмеженнями
Обчислювальна складність	Лінійна або квазі-лінійна	Квадратична або кубічна	Лінійна (але з великою константою)

Для розробки системи виявлення DDoS-атак було обрано метод МБФГШ в якості алгоритму машинного навчання на декілька підстав:

– швидкість навчання. В контексті виявлення DDoS-атак, швидкість навчання є критично важливою, оскільки атаки можуть змінюватися і адаптуватися. МБФГШ забезпечує високу швидкість навчання, що є особливо корисним для великих наборів даних;

– ефективне використання пам’яті. МБФГШ оптимізований для ефективного використання пам’яті, що є важливим для обробки великих наборів даних в реальному часі;

– стійкість до шуму. В даних про мережевий трафік завжди присутній шум. L-BFGS відомий своєю стійкістю до шуму, що забезпечує високу точність виявлення атак;

– масштабованість. Однією з ключових переваг МБФГШ є його здатність масштабуватися для великих наборів даних. Це особливо важливо для систем виявлення DDoS-атак, які можуть потребувати аналізу великих об’ємів даних в реальному часі;

– регуляризація. МБФГШ має вбудовану регуляризацію, яка допомагає моделі уникнути перенавчання та збільшує її загальну здатність до узагальнення;

– обчислювальна складність. МБФГШ має лінійну або квазі-лінійну обчислювальну складність, що є прийнятним для реальних систем виявлення DDoS-атак.

З урахуванням цих факторів, МБФГШ видається оптимальним вибором для розробки системи виявлення DDoS-атак в умовах великих наборів даних, високих вимог до швидкості обробки та стійкості до шуму.

2.3 Розробка власної математичної моделі для вирішення задачі

У даному розділі розглядається розробка власної математичної моделі для вирішення задачі виявлення порушень в системах критичної інформаційної інфраструктури, зокрема DDoS-атак. Модель базується на

алгоритмі оптимізації МБФГШ, що є одним з найефективніших методів для розв'язання задач бінарної класифікації в умовах великих обсягів даних. Для реалізації цієї моделі використовуються дані протоколу CoAP, які представлені у форматі, що відображає реальну структуру мережевого трафіку. Детальний опис математичної моделі дозволить зрозуміти, як алгоритм взаємодіє з даними та які особливості його роботи можуть бути використані для підвищення ефективності системи виявлення порушень.

Задача, яку необхідно вирішити є задачею бінарної класифікації, де є набір даних (X, Y), де $X \in R^{n \times m}$ є матрицею ознак, а $Y \in \{0,1\}^n$ є вектором міток. В даному випадку, n – кількість прикладів у наборі даних, а m=10 – кількість ознак (*Mid, Code, OptDesc, LocationQuery, MaxAge, UriHost, ResponseIn, Retransmitted, Token, TokenLen*).

Цільова функція логістичної регресії бути представлена як:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right], \quad (2.1)$$

де, $h_\theta(x)$ є сигмоїдною функцією, а θ – вектор параметрів моделі.

Матриця X формується з екземплярів класу *CoapData*. Кожний рядок матриці X відображає один екземпляр «*CoapData*», а кожний стовпець відповідає одному з полів (*Mid, Code, OptDesc*, тощо). Тобто, елемент x_{ij} матриці X є значенням *j*-го поля *i*-го екземпляру *CoapData*:

$$X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{pmatrix} \quad (2.2)$$

Вектор Y у свою чергу формується з полів *Class* кожного екземпляру *CoapData*:

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}, \quad (2.3)$$

де y_i може бути 1 (атака) або 0 (не атака).

Після тренування моделі, прогнози $\hat{Y}$ отримуються за допомогою сигмоїдної функції $h_\theta(x)$:

$$\hat{Y} = h_\theta(x) = \frac{1}{1+\exp(-X\theta)} \quad (2.4)$$

Ці прогнози зберігаються в екземплярах класу CoapPrediction в полі «Prediction»:

$$CoapPrediction.Prediction = \begin{cases} 1, \text{якщо } \hat{y}_i \geq 0{,}5 \\ 0, \text{якщо } \hat{y}_i < 0{,}5 \end{cases} \quad (2.5)$$

Таким чином, інтеграція з CoapData та CoapPrediction дозволяє не тільки структуровано представити вхідні та вихідні дані, але й ефективно використовувати їх для тренування та прогнозування в моделі МБФГШ, що значно підвищує точність та надійність системи виявлення DDoS-атак.

Висновок до другого розділу

В рамках даного розділу було здійснено фундаментальну підготовку для розробки системи, що буде спрямована на забезпечення безпеки в системах критичної інформаційної інфраструктури. Зокрема, була виконана формалізація задачі виявлення порушень, що дозволило визначити ключові параметри та критерії для подальших досліджень. Детальний аналіз методів машинного навчання привів до вибору алгоритму Бройдена-Флетчера-Гольдфарба-Шанно як оптимального для розробки системи виявлення DDoS-атак. Крім того, була розроблена власна математична модель на базі алгоритму МБФГШ, яка враховує специфіку даних протоколу CoAP. Ці етапи підготовки створюють тверду основу для подальшої розробки системи та її тестування на реальних даних.

3 АНАЛІЗ ТА ВИБІР ТЕХНОЛОГІЙ І МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ ВИРІШЕННЯ ПРОБЛЕМИ

3.1 Вибір технологій для реалізації системи

3.1.1 Мова програмування

Для реалізації системи виявлення порушень коректного функціонування систем критичної інформаційної інфраструктури було вибрано мову програмування C#. Обрана технологія є результатом ретельного аналізу та синтезу існуючих методологій та інструментів. Це рішення може бути аргументовано наступними ключовими факторами:

– об’єктно-орієнтована парадигма. C# є об’єктно-орієнтованою мовою, що дозволяє імплементувати складні алгоритми виявлення аномалій та моніторингу за допомогою зручної структуризації коду [34]. Це полегшує модульний підхід, інкапсуляцію функціональності та повторне використання програмних компонентів;

– універсальність та масштабованість. За допомогою .NET Core або .NET Framework можна реалізувати як локальні, так і хмарні рішення. Це критично для систем, які мають адаптуватися до зростаючих вимог забезпечення безпеки критичної інформаційної інфраструктури [35];

– підтримка аналітичних моделей. Інтеграція з ML.NET дозволяє впроваджувати моделі машинного навчання для покращення алгоритмів виявлення порушень [36];

– сильна типізація. Ця особливість дозволяє підвищити надійність коду шляхом строгого контролю типів даних, що виключає багато типів програмних помилок на етапі компіляції;

– механізми безпеки. Система управління пам’яттю і вбудовані механізми безпеки в C# допомагають уникнути потенційних вразливостей, що є вкрай важливо для систем, які контролюють критичні процеси;

– підтримка багатопоточності. Ефективна робота з багатопоточністю дозволяє оптимізувати використання ресурсів процесора для паралельної обробки даних, що є незамінним для реалізації високопродуктивних систем моніторингу.

Слід зазначити, що вибір C# як основи для розробки не лише спрощує інженерний процес, але і забезпечує високий рівень адаптивності та масштабованості системи. Таким чином, розроблена система може ефективно адаптуватися до змінних умов функціонування і вимог до забезпечення інформаційної безпеки.

3.1.2 Середовище розробки

Для задач комп'ютерно-математичного моделювання системи виявлення порушень в системах критичної інформаційної інфраструктури вибране інтегроване середовище розробки (IDE) – Visual Studio. Раціональність цього вибору може бути сформульована з урахуванням наступних аспектів:

– комплексність інструментарію. Visual Studio пропонує високоорганізований набір інструментів для етапів розробки, від дизайну архітектури до налагодження та профілювання. Це підвищує ефективність розробки та дозволяє зосередитися на специфічних задачах моделювання [37];

– оптимізована робота з C# та .NET. Враховуючи, що основна логіка системи імплементована на C#, Visual Studio забезпечує оптимальні умови для розробки на цій мові, зокрема в контексті .NET-екосистеми;

– інтеграція з ML.NET. Для реалізації алгоритмічних моделей машинного навчання в системі виявлення порушень, підтримка ML.NET у Visual Studio є стратегічно важливою [38];

– сумісність з хмарними технологіями. Інтеграція з Microsoft Azure дозволяє не лише зберігати дані та моделі в хмарі, але і розгортати складні розподілені системи для моніторингу [39];

– поліморфізм проектів. Підтримка різноманітних типів проектів (від консольних до веб-сервісів) надає гнучкість в архітектурному проектуванні, що є критично важливим для систем високого рівня надійності.

З урахуванням цих факторів, можна зробити висновок, що Visual Studio являє собою оптимальний вибір для реалізації даного класу задач. Його комплексний підхід до розробки, високий рівень інтеграції з сучасними технологіями та зручність управління ресурсами забезпечують зрілу платформу для ефективного та надійного комп’ютерно-математичного моделювання.

3.1.3 Система управління базами даних

Для задачі комп’ютерно-математичного моделювання системи виявлення порушень коректного функціонування в системах критичної інформаційної інфраструктури вибрано систему управління базами даних (СУБД) Microsoft SQL Server. Рішення засноване на обміркованому аналізі відповідності СУБД до вимог задачі. Основні аргументи вибору можна сформулювати наступним чином:

– сумісність з .NET і C#. З урахуванням того, що основна бізнес-логіка системи реалізована на C#, інтеграція SQL Server з .NET Framework стає стратегічним плюсом. Це спрощує процес розробки та деплою [40];

– продуктивність та масштабованість. SQL Server розроблено з урахуванням високих вимог до продуктивності та масштабованості, що є критично для обробки великих даних в контексті критичної інфраструктури;

– механізми безпеки. Наявність вбудованих механізмів безпеки, включаючи шифрування та управління доступом, є необхідною для захисту чутливої інформації та дотримання стандартів безпеки [41];

– цілісність даних через управління транзакціями. SQL Server має вбудовані засоби для транзакційної обробки, що забезпечує цілісність даних – важливий аспект в контексті аналітичних моделей та алгоритмів виявлення порушень [42];

– інструменти для аналітичної роботи. Широкий спектр вбудованих інструментів для адміністрування, моніторингу, налагодження та моделювання даних значно підвищує продуктивність розробки [43];

– підтримка розподіленого та паралельного обчислення. Це особливо актуально для моделювання в реальному часі та аналізу великих даних, що є необхідними для систем критичної інфраструктури;

– маніпуляції зі структурованими даними. Підтримка JSON та XML форматів дозволяє ефективно зберігати та обробляти структуровані дані, що є важливим для аналітичних моделей;

– функціонал повнотекстового пошуку. Цей аспект може бути корисним для швидкої фільтрації даних та виявлення специфічних шаблонів або аномалій.

Таким чином, SQL Server відповідає комплексу вимог, що пред’являються до СУБД в контексті комп’ютерно-математичного моделювання для систем критичної інформаційної інфраструктури. Його функціональні можливості дозволяють гармонійно інтегрувати базу даних у загальну архітектуру системи, забезпечуючи її надійність, продуктивність та безпеку.

3.2 Вибір бібліотеки машинного навчання

Вибір бібліотеки для реалізації алгоритмів МН є одним з ключових етапів під час проектування системи виявлення порушень в системах критичної інформаційної інфраструктури. Враховуючи специфіку задачі, що включає не тільки високу точність класифікації, але й ефективну роботу в реальному часі, бібліотека машинного навчання повинна відповідати ряду критеріїв. Ці критерії включають в себе можливість роботи з великими датасетами, гнучкість в налаштуванні моделей, надійні механізми оцінки якості та вбудовані засоби для інтерпретації результатів. Також необхідно взяти до уваги питання сумісності вибраної бібліотеки з іншими

компонентами системи, зокрема СУБД та інтегрованим середовищем розробки.

3.2.1 Класифікація бібліотек машинного навчання

Класифікація бібліотек для машинного навчання може бути виконана з урахуванням різних критеріїв, таких як сфера застосування, можливості, мова програмування, на якій базуються, та інші. Однак, з огляду на задачу моделювання системи виявлення порушень в системах критичної інформаційної інфраструктури, акцент буде зроблено на таких параметрах, як сумісність, продуктивність, та функціональні можливості. Для вирішення цієї задача можна розглянути такі бібліотеки: ML.NET, Accord.NET та AForge.NET.

ML.NET

ML.NET – це відкрита бібліотека машинного навчання від компанії Microsoft, розроблена з метою інтеграції з .NET-екосистемою [44]. Основна мета бібліотеки – надання зручних засобів для розробки, тренування та впровадження моделей машинного навчання безпосередньо в .NET-додатках. Це означає, що розробники, які вже знайомі з C# або F#, можуть легко включити можливості машинного навчання в свої проекти без необхідності звертатися до сторонніх мов програмування або платформ.

ML.NET працює на основі конвеєрного принципу обробки даних, що передбачає послідовну трансформацію та аналіз даних. Цей конвеєр включає в себе чотири основні етапи [45]:

– завантаження даних. Дані можуть бути завантажені з різних джерел, включаючи файлові системи, бази даних, хмарні сховища тощо;

– попередня обробка. На цьому етапі відбувається очищення, нормалізація, інжиніринг ознак та інші трансформації, які підготовлюють дані для моделювання;

– тренування моделі. Використовуючи алгоритми класифікації, регресії, кластеризації або інші, ML.NET тренує модель на підготовлених даних;

– оцінка та впровадження. Після тренування модель оцінюється за допомогою різних метрик. Якщо модель вважається задовільною, її можна впровадити в продуктивне середовище.

В основі роботи ML.NET лежить концепція «конвеєрів» та «трансформерів». Конвеєр – це послідовність етапів обробки даних, кожен з яких реалізований через трансформер. Трансформери можуть виконувати різні задачі: від нормалізації числових значень до векторизації тексту. Після попередньої обробки даних, алгоритм машинного навчання може бути «приєднаним» до цього конвеєра як ще один трансформер. Така архітектура робить систему гнучкою та легко розширюваною.

З урахуванням своєї вбудованої сумісності з .NET, ML.NET є ідеально підходящим вибором для проектів, які вимагають інтеграції з іншими технологіями Microsoft, такими як SQL Server чи ASP.NET, і може бути особливо корисним в задачах виявлення порушень в системах критичної інформаційної інфраструктури.

Accord.NET

Accord.NET є комплексною бібліотекою для наукових досліджень і інженерії в області машинного навчання, статистики та обробки сигналів [46]. Розроблена на мові C#, вона пропонує широкий арсенал алгоритмів для різноманітних задач, включаючи, але не обмежуючись, класифікацію, регресію, кластеризацію, і навіть обробку зображень і аудіо. Це робить Accord.NET універсальним інструментом для розробників на платформі .NET.

Принцип роботи даної бібліотеки виглядає наступним чином [47]:

– завантаження даних. Accord.NET дозволяє імпортувати дані з різних форматів і джерел, включаючи текстові файли, бази даних і інші;

– попередня обробка. Бібліотека пропонує ряд методів для попередньої обробки даних, таких як нормалізація, трансформація та видалення аномалій;

– вибір алгоритму та тренування моделі. Accord.NET має багатий набір «вчителів» (learning algorithms), що дозволяє легко тренувати моделі для конкретних задач;

– оцінка моделі. Після тренування, можна оцінити ефективність моделі за допомогою вбудованих метрик і методів крос-валідації;

– впровадження моделі. Треновані моделі можуть бути експортовані для подальшого використання в інших .NET-додатках або для реального часу прогнозування.

Отже, Accord.NET працює на основі модульної архітектури, де кожен алгоритм або функціональний блок представлений у вигляді окремого компонента. Це дозволяє розробникам з легкістю комбінувати різні частини бібліотеки для створення комплексних систем машинного навчання, що може бути особливо корисним в задачах аналізу та моніторингу систем критичної інформаційної інфраструктури.

AForge.NET

AForge.NET – це бібліотека для розробки застосунків в області комп’ютерного зору, штучного інтелекту, обробки зображень і робототехніки. Хоча вона не є винятково бібліотекою машинного навчання, AForge.NET містить ряд алгоритмів та інструментів для машинного навчання, включаючи, але не обмежуючись, нейронними мережами та генетичними алгоритмами [48]. Розроблена на мові C#, ця бібліотека інтегрується з .NET-екосистемою, що робить її доступною для розробників на цій платформі.

Принцип роботи даної бібліотеки виглядає наступним чином [49]:

– завантаження даних. AForge.NET може обробляти дані з різних джерел, особливо з медіа-джерел, таких як відеокамери або аудіо-захоплення;

– попередня обробка. Бібліотека надає набір інструментів для попередньої обробки даних, зокрема для фільтрації та нормалізації зображень і сигналів;

– вибір алгоритму та тренування моделі. AForge.NET містить різні алгоритми машинного навчання, такі як нейронні мережі та генетичні алгоритми, для різних типів задач;

– оцінка моделі. Хоча AForge.NET може не мати такої обширної системи метрик оцінки, як інші спеціалізовані бібліотеки, вона дозволяє провести базову оцінку ефективності моделі;

– впровадження моделі. Затреновані моделі можуть бути інтегровані в .NET-додатки для реального часу аналізу або прогнозування.

AForge.NET фокусується на модульній архітектурі, де кожен компонент або алгоритм може бути використаний як частина більшої системи. Це робить AForge.NET гнучким інструментом, що може бути адаптований для широкого спектру задач, включаючи ті, що пов’язані з критичною інформаційною інфраструктурою.

3.2.2 Порівняння та вибір конкретної бібліотеки

Для об’єктивного вибору бібліотеки машинного навчання для розробки системи виявлення порушень коректного функціонування систем критичної інформаційної інфраструктури, необхідно провести порівняльний аналіз розглянутих бібліотек машинного навчання.

Таблиця 3.1 – Порівняльний аналіз бібліотек машинного навчання

Характеристика	ML.NET	Accord.NET	AForge.NET
1	2	3	4
Сумісність з C# та .NET	Висока	Висока	Висока

Продовження таблиці 3.1

1	2	3	4
Масштабованість	Висока (підтримка хмарних рішень, оптимізація під великі набори даних)	Середня	Середня
Підтримка різних типів алгоритмів МН	Широкий спектр, включаючи глибоке навчання	Широкий спектр, але без акценту на глибоке навчання	Обмежений спектр (обробка сигналів та зображень)
Безпека та надійність	Висока (розроблена Microsoft, активно підтримується)	Середня	Середня
Швидкодія	Висока (оптимізована для високої продуктивності)	Середня	Середня
Гнучкість та модульність	Висока (можлива інтеграції з іншими бібліотеками та платформами)	Висока	Висока
Документація та підтримка спільноти	Висока (активна спільнота, офіційна документація від Microsoft)	Середня	Середня

Вибір ML.NET як основної бібліотеки для розробки системи виявлення порушень коректного функціонування систем критичної інформаційної інфраструктури обумовлений рядом ключових переваг цієї технології. По-перше, ML.NET відзначається високим рівнем сумісності з екосистемою C# та .NET, що спрощує інтеграцію і впровадження розробленого рішення. По-друге, ця бібліотека забезпечує високу масштабованість і продуктивність, що є критично важливим для обробки великих обсягів даних в реальному часі. По-

третє, ML.NET підтримує широкий спектр алгоритмів МН, включаючи сучасні методи глибокого навчання, які можуть бути ефективно використані для аналізу складних шаблонів та аномалій. Також варто відзначити високий рівень безпеки та надійності рішення, який забезпечується завдяки активній підтримці і регулярному оновленню від розробників Microsoft. Наявність активної спільноти та офіційної документації забезпечує додаткові можливості для ефективної розробки та оптимізації системи. З урахуванням усіх зазначених факторів, ML.NET є оптимальним вибором для реалізації даного проекту.

3.3 Опис алгоритмів реалізації моделі машинного навчання

Для розробки системи виявлення порушень в об’єктах критичної інформаційної інфраструктури, однією з ключових задач є побудова ефективної моделі машинного навчання. Основна мета полягає у тому, щоб оптимізувати процес навчання мережі так, щоб максимально підвищити точність прогнозування аномалій та потенційних порушень. Для реалізації цієї мети розроблено алгоритм навчання, який представлений на рис. 3.1.

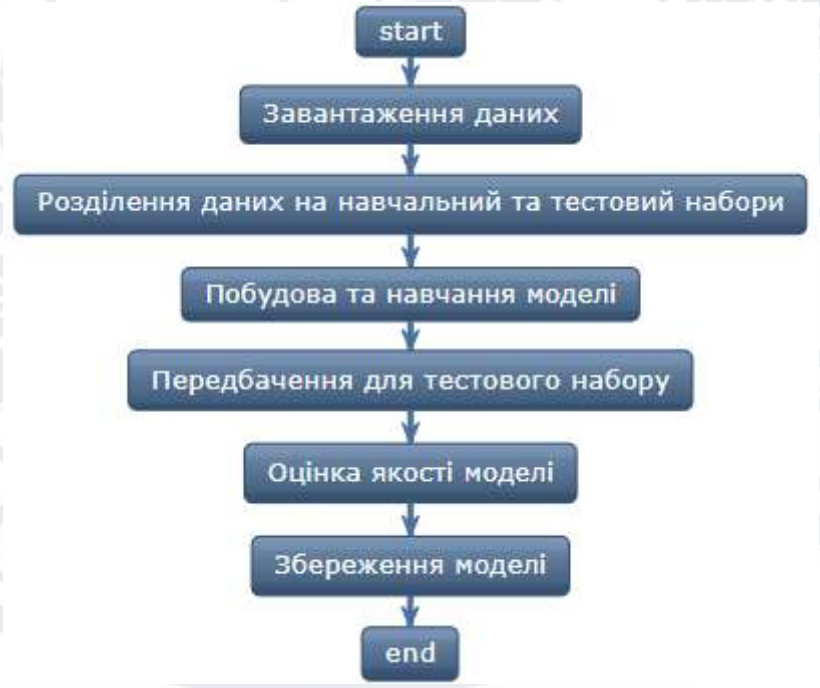

Рисунок 3.1 – Алгоритм навчання нейронної мережі

Алгоритм навчання моделі нейронної мережі для теми «Комп'ютерно-математичне моделювання системи виявлення порушень коректного функціонування систем критичної інформаційної інфраструктури» розпочинається з ініціалізації контексту машинного навчання та завантаження даних з CSV-файлу. Після завантаження даних відбувається їх розділення на навчальний та тестовий набори. Далі створюється конвеєр для обробки та тренування моделі, який включає в себе нормалізацію характеристик, копіювання стовпця класу в новий стовпець мітки та використання алгоритму бінарної класифікації. Після цього модель тренується на навчальному наборі даних, а потім використовується для передбачення на тестовому наборі. Оцінка якості моделі відбувається за допомогою різних метрик, таких як Log-loss, Area Under ROC Curve, Area Under Precision-Recall Curve та Accuracy. Якщо модель успішно навчена, вона зберігається в файлі з розширенням .zip. Завершальним етапом є збереження інформації про навчену модель в базі даних.

Рис. 3.2 відображає алгоритм прогнозування можливої DDoS-атаки.

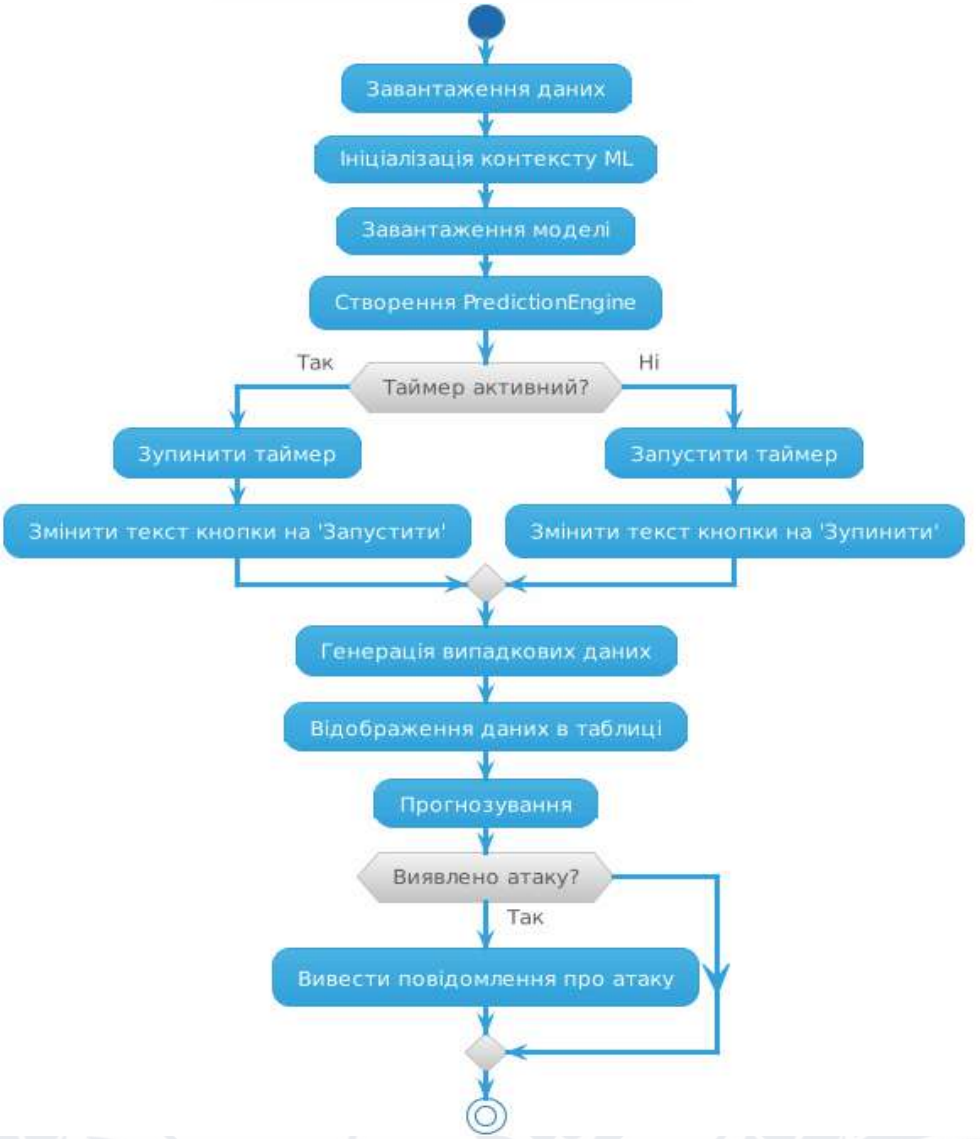

Рисунок 3.2 – Алгоритм прогнозування

Алгоритм прогнозування в системі виявлення DDoS-атак ініціалізується під час завантаження форми, де відбувається завантаження всіх доступних категорій для подальшого вибору користувачем. Після вибору категорії система завантажує відповідну модель машинного навчання, яка потім використовується для створення об’єкта «Двигун прогнозування». Цей об’єкт служить для здійснення прогнозів на основі вхідних даних. Коли користувач активує таймер за допомогою кнопки «Запустити», система починає циклічно генерувати випадкові дані, що імітують мережевий трафік. Ці дані подаються на вхід «Двигун прогнозування», який здійснює прогноз. У випадку позитивного прогнозу, що свідчить про можливу DDoS-атаку, система виводить відповідне повідомлення. Циклічність алгоритму зберігається до

моменту, коли користувач натискає кнопку «Зупинити», що призводить до зупинки таймера та завершення процесу прогнозування.

3.4 Проектування архітектури програмного рішення

Для розробки системи виявлення DDoS-атак було обрано трьохрівневу архітектуру, яка є однією з найбільш популярних та ефективних моделей організації програмного коду. Трьохрівнева архітектура розділяє програмний код на три основні рівні: презентаційний рівень, рівень бізнес-логіки та рівень доступу до даних.

Презентаційний рівень відповідає за інтерфейс користувача та його взаємодію з системою. Цей рівень забезпечує зручність та інтуїтивність користування, а також відображення необхідної інформації.

Рівень бізнес-логіки є серцем системи. Він містить всі алгоритми, методи та функції, які забезпечують функціональність системи. В нашому випадку, це алгоритми машинного навчання для виявлення DDoS-атак, логіка обробки даних та їх аналіз.

Рівень доступу до даних забезпечує взаємодію системи з зовнішніми джерелами даних, такими як бази даних або файлові системи. Цей рівень відповідає за зберігання, завантаження та модифікацію даних.

Спочатку була створена архітектура для взаємодії з базою даних, що включає в себе п’ять спеціалізованих класів. Кожен з цих класів має назву, яка корелює з назвою відповідної таблиці в базі даних і доповнюється суфіксом «Provider». Діаграма класів, що ілюструє методи для оперування даними в базі, представлена на рис. 3.3.

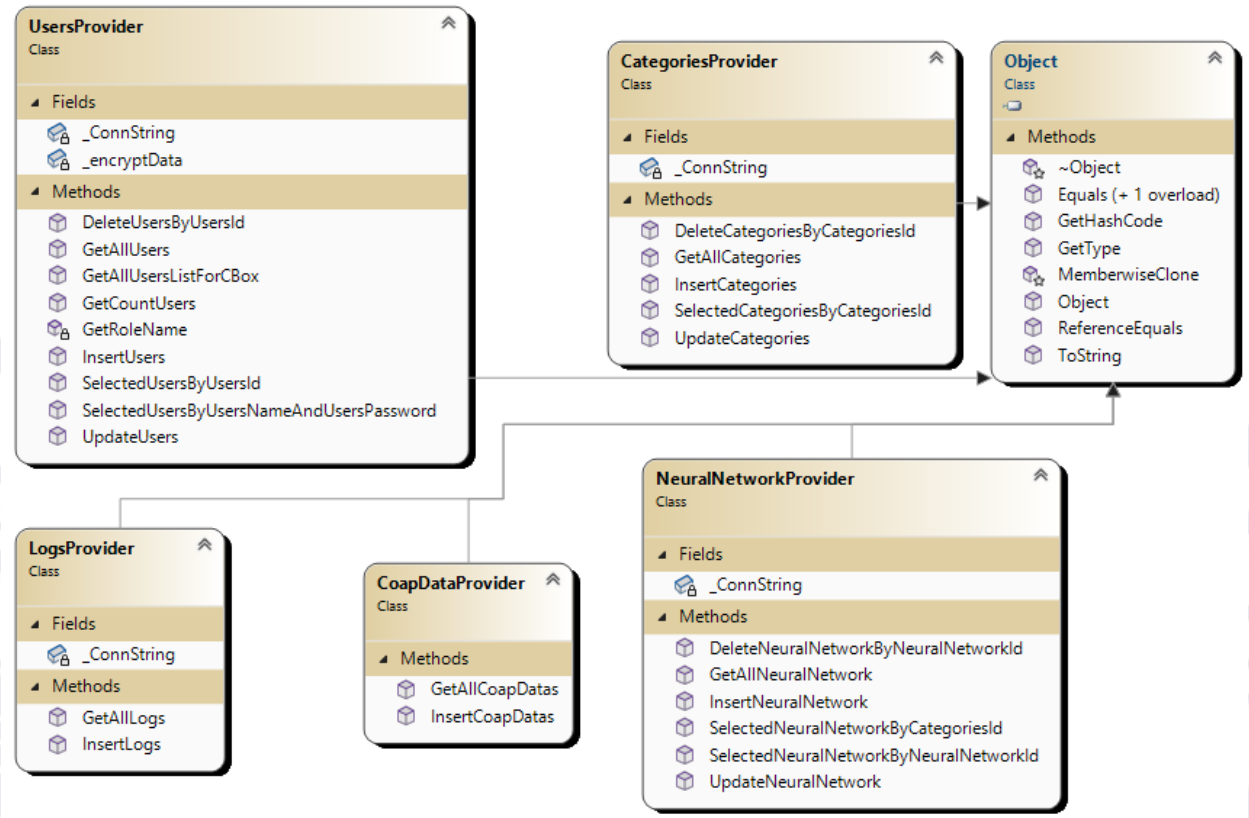

Рисунок 3.3 – Діаграма класів рівня даних

Класи рівня даних служать мостом між бізнес-логікою та базою даних. Вони забезпечують абстракцію доступу до даних, ізолюючи вищі рівні від деталей реалізації зберігання даних. Як можна побачити із рис. 3.3 діаграма класів даного рівня складається:

– CategoriesProvider. Цей клас відповідає за управління категоріями в системі. Він містить методи для отримання, додавання, оновлення та видалення категорій та взаємодіє з таблицею категорій в базі даних;

– CoapDataProvider. Клас, призначений для роботи з даними протоколу CoAP. Він включає методи для зберігання, отримання та обробки даних CoAP, які зберігаються в спеціалізованій таблиці бази даних;

– LogsProvider. Клас для управління журналами системи. Він містить методи для запису, отримання та аналізу журналів, які можуть бути використані для моніторингу, відлагодження та аудиту системи;

– NeuralNetworkProvider. Цей клас відповідає за управління нейронними мережами в системі. Він містить методи для завантаження

натренованих моделей, їх оцінки та використання для прогнозування. Зазвичай він взаємодіє з файлами моделей та таблицею в базі даних;

– UsersProvider. Клас, призначений для управління користувачами системи. Він включає методи для реєстрації, автентифікації, авторизації та управління профілями користувачів.

На рівні взаємодії з користувачем було реалізовано комплекс форм, кожна з яких містить в собі асортимент елементів керування. Серед таких елементів можна виділити кнопки для ініціації конкретних дій, текстові поля для введення даних, а також таблиці для відображення інформації у структурованому вигляді.

Ці елементи не просто статичні компоненти; вони динамічно взаємодіють з іншими рівнями системи. Для цього використовуються спеціалізовані обробники подій, які слугують мостом між користувацьким інтерфейсом та бізнес-логікою або рівнем доступу до даних. Це дозволяє системі реагувати на дії користувача в реальному часі, проводячи необхідні розрахунки, запити до бази даних чи інші операції.

Користувач, в свою чергу, може взаємодіяти з системою безпосередньо через цей рівень. Він має можливість вводити необхідну інформацію для обробки, ініціювати пошукові запити, а також отримувати результати цих запитів. Все це представлено у форматі, який максимально зручний для користувача, з урахуванням ергономіки та інтуїтивності користувацького інтерфейсу.

Для наочного представлення архітектури рівня користувацького інтерфейсу була створена діаграма, розміщена на рис. 3.4. Ця діаграма детально відображає взаємозв’язки між елементами керування та їх взаємодію з іншими компонентами системи.

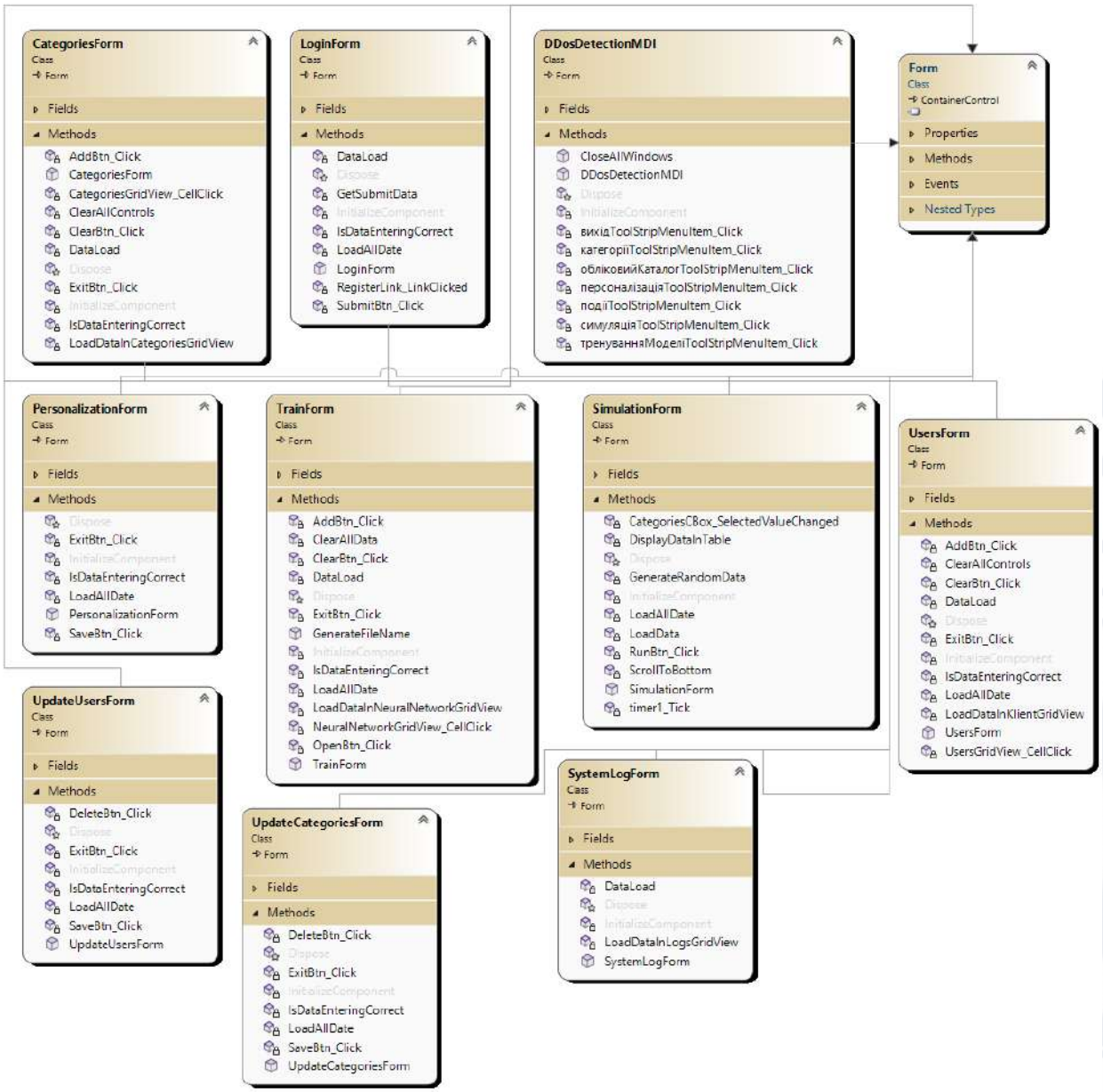

Рисунок 3.4 – Діаграма класів рівня користувацького інтерфейсу

Діаграма класів даного рівня складається із наступних класів:

– клас DDosDetectionMDI слугує головним вікном додатку і є точкою входу для користувацького інтерфейсу. Він координує доступ до різних функціональних частин системи, представлених у вигляді вкладених форм. Через це вікно користувач може переходити до різних секцій програми, таких як симуляція, категорії, тренування моделі та інше;

– клас SimulationForm відповідає за симуляцію процесу виявлення DDoS-атак. Він містить логіку для генерації випадкових даних, їх відображення та взаємодії з моделлю прогнозування. Користувач може

запустити або зупинити симуляцію, а також бачити результати в реальному часі;

– клас CategoriesForm забезпечує інтерфейс для управління категоріями, які можуть бути використані для класифікації DDoS-атак. Він дозволяє користувачам додавати, видаляти або редагувати категорії, а також переглядати існуючі;

– клас TrainForm є відповідальним за процес тренування нейронної мережі. Користувач може вибрати набір даних для тренування, ініціювати сам процес, а також зберегти натреновану модель для подальшого використання в системі;

– клас UpdateCategoriesForm представляє інтерфейс для оновлення інформації про категорії. Він використовується для зміни атрибутів існуючих категорій, таких як назва, опис, параметри для класифікації та інше;

– клас LoginForm служить входом до системи, він відповідає за аутентифікацію користувачів. Через форму, яку представляє цей клас, користувачі можуть ввести свої облікові дані – ім’я користувача та пароль – для доступу до функціональності системи. Успішна аутентифікація відкриває доступ до інших частин програми;

– клас PersonalizationForm надає можливість користувачам налаштовувати різні параметри системи згідно з їхніми персональними потребами. Це може включати в себе зміну теми інтерфейсу, налаштування сповіщень або конфігурацію інших параметрів, які впливають на користувацький досвід;

– клас SystemLogForm представляє інтерфейс для перегляду системних журналів. Користувач може переглядати записи про різні події в системі, такі як спроби входу, виявлені атаки, помилки та інше. Це важливий інструмент для моніторингу стану системи та її діагностики;

– клас UpdateUsersForm відповідає за оновлення інформації про користувачів. Він дозволяє адміністраторам або уповноваженим користувачам

змінювати дані про користувачів, такі як ім’я, пароль, роль в системі та інші атрибути;

– клас UsersForm представляє інтерфейс для управління користувачами системи. Через нього можна додавати нових користувачів, видаляти існуючих або переглядати список всіх зареєстрованих користувачів.

Кожен з цих класів виконує конкретну роль в архітектурі системи, забезпечуючи зручний та ефективний інтерфейс для взаємодії користувача з системою.

Завершальним кроком у формуванні архітектури програмного рішення стало розроблення бізнес-логіки. Цей етап є критично важливим для успішної реалізації програмного продукту, оскільки саме бізнес-логіка відповідає за обробку та виконання ключових бізнес-операцій. Цей компонент системи фокусується на внутрішніх алгоритмах та правилах, які керують функціональністю додатку, і не має прямого зв’язку ні з базою даних, ні з користувацьким інтерфейсом. Це дозволяє забезпечити гнучкість та надійність системи, а також спростити процеси тестування та масштабування. Структуру цього компоненту можна побачити на рис. 3.5.

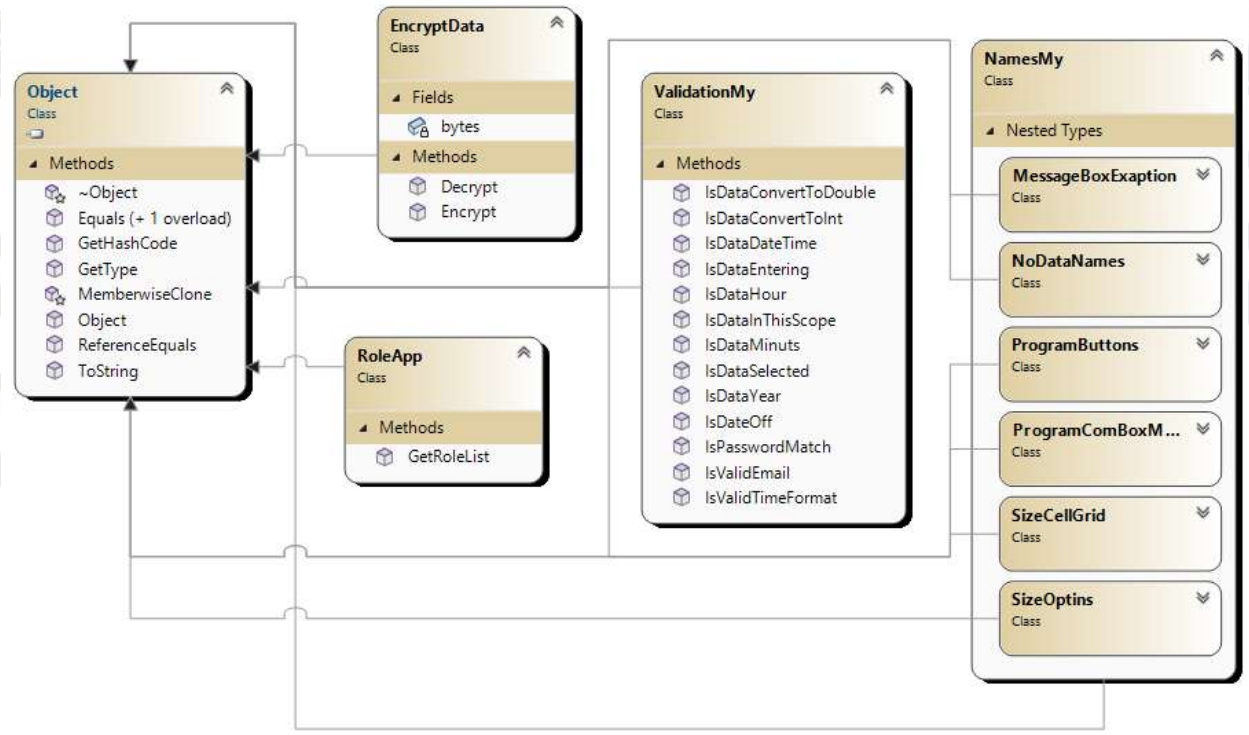

Рисунок 3.5 – Діаграма класів рівня бізнес-логіки

Класи рівня бізнес-логіки відіграють ключову роль у забезпеченні функціональності та надійності системи. Вони слугують мостом між користувацьким інтерфейсом та рівнем даних, виконуючи ряд специфічних завдань, зокрема:

– клас EncryptData відповідає за шифрування даних, які передаються або зберігаються в системі. Він використовує різні алгоритми шифрування для забезпечення конфіденційності інформації, що є критично важливим для забезпечення безпеки даних;

– клас NamesMy відповідальний за управління іменами або ідентифікаторами, які використовуються в системі. Він може забезпечувати унікальність, валідацію та інші аспекти, пов’язані з іменами користувачів, ролей та будь-яких інших сутностей;

– клас RoleApp управляє ролями користувачів в додатку. Він визначає, які дії доступні для різних типів користувачів, і може включати в себе механізми для додавання, видалення або модифікації ролей. Це дозволяє гнучко управляти доступом до різних частин системи;

– клас ValidationMy призначений для валідації даних, які вводяться або обробляються в системі. Він включає в себе ряд методів для перевірки формату, достовірності та інших атрибутів даних, що забезпечує високий рівень якості даних та зменшує ризик помилок.

Кожен з цих класів виконує відповідні завдання, що спрямовані на забезпечення ефективності та надійності системи в цілому.

Висновок до третього розділу

В ході аналізу та вибору технологій і методів машинного навчання для вирішення поставленої проблеми було виконано ряд ключових завдань. Для реалізації системи було обрано мову програмування C# на підставі її гнучкості, надійності та широкого функціоналу. Середовище розробки Visual Studio було вибрано завдяки його продуктивності та інтеграції з C#. Для

управління базами даних використовується Microsoft SQL Server, що забезпечує високу продуктивність та безпеку.

Було проведено класифікацію бібліотек машинного навчання, зокрема ML.NET, Accord.NET та AForge.NET, і на підставі порівняльного аналізу було вирішено використовувати ML.NET для навчання моделі нейронної мережі. Це дозволило забезпечити ефективну реалізацію алгоритмів навчання та прогнозування.

Також, було розроблено алгоритми для реалізації моделі машинного навчання, зокрема алгоритм навчання нейронної мережі та алгоритм прогнозування, що забезпечують високу точність та надійність системи.

В кінці розділу, було спроектовано архітектуру програмного рішення на основі трьохрівневої моделі, що включає рівні користувацького інтерфейсу, бізнес-логіки та управління даними. Діаграми класів були побудовані для кожного рівня, а їх призначення детально описано.

Отже, виконані завдання та отримані результати стали фундаментом для реалізації надійної, ефективної та гнучкої системи, що може бути адаптована для різних бізнес-вимог та сценаріїв використання.

4 КОНСТРУЮВАННЯ, ВЕРИФІКАЦІЯ ТА АНАЛІЗ ПРОГРАМНОЇ КОМПОНЕНТИ

4.1 Проектування та розробка схеми бази даних

Початковий етап розробки проекту полягає в деталізованому проектуванні структури бази даних, яка є фундаментальною складовою системи. Для цього завдання було обрано методологічний підхід «сутність-зв'язок», відомий як ER-метод [50]. Цей підхід є одним з найбільш ефективних для створення логічних моделей баз даних, оскільки він дозволяє з чіткістю визначити ключові елементи предметної області та їх взаємозв'язки.

Використання ER-методу сприяє глибокому аналізу предметної області, що в свою чергу дозволяє виявити всі необхідні сутності, їх атрибути та зв'язки між ними. Це створює умови для розробки високоякісної, надійної та масштабованої бази даних, яка може ефективно відображати реальний світ та взаємодії в ньому.

В ході детального проектування бази даних було виявлено ключові сутності, що представляють основні об'єкти та концепції предметної області. До цих сутностей були прив'язані відповідні атрибути, які деталізують характеристики кожної сутності:

– категорії: ідентифікатор категорії, назва категорії та її опис. Ця сутність служить для класифікації даних та їх групування за певними критеріями;

– дані CoAP: ідентифікатор повідомлення, код операції, опис опцій, запит місцезнаходження, максимальний вік, хост URI, відгук, передача повторно, токен, довжина токену, мітка порушення безпеки та ідентифікатор категорії. Ця сутність зберігає всі зібрані дані протоколу CoAP для подальшого аналізу;

– логи: ідентифікатор логу, ідентифікатор користувача, назва події, дата події та ім'я користувача. Ця сутність відображає всі події, що відбулися

в системі, зберігаючи при цьому інформацію про користувача, який їх ініціював;

– нейронна мережа: ідентифікатор, назва нейронної мережі, ідентифікатор категорії та шлях до файлу збереженої моделі. Ця сутність зберігає всю необхідну інформацію для роботи з нейронними мережами;

– користувачі: ідентифікатор користувача, прізвище, ім’я, ім’я користувача, пароль, ідентифікатор ролі, додатковий опис та електронна пошта. Ця сутність зберігає всю інформацію про користувачів системи, їх ролі та права доступу.

Під час проектування бази даних особлива увага була приділена встановленню зв’язків між різними сутностями. Це важливий етап, який впливає на ефективність та надійність роботи всієї системи. Для цього було використано концепції зовнішніх та внутрішніх ключів, які дозволяють не тільки встановлювати зв’язки між таблицями, але й визначати їх типи.

Застосування зовнішнього ключа дозволило встановити зв’язки між різними таблицями таким чином, що інтегритет даних зберігається на всіх рівнях. Це зокрема важливо для забезпечення консистентності даних та їхньої надійності. Внутрішні ключі, з свого боку, використовуються для унікальної ідентифікації записів всередині однієї таблиці.

Організація зв’язків між таблицями була виконана таким чином, щоб максимально врахувати специфіку предметної області та забезпечити можливість масштабування системи в майбутньому. Це стало можливим завдяки глибокому аналізу взаємозв’язків між різними типами даних та їх ролею в загальній архітектурі додатку.

В процесі проектування бази даних було створено зовнішні зв’язки між таблицями. Ці зв’язки не тільки сприяють інтегритету даних, але й забезпечують логічну координованість між різними сутностями в системі. Опис зовнішніх зв’язків представлено нижче:

зв’язок між таблицею «Categories» та «CoapDatas» через поле «CategoriesId». Вид цього зв’язку є «один до багатьох» (1:N). Одна категорія

може бути асоційована з багатьма записами в таблиці «CoapDatas», але кожний запис в «CoapDatas» прив’язаний лише до однієї категорії;

зв’язок між таблицею «Categories» та «NeuralNetwork» через поле «CategoriesId». Вид зв’язку – «один до багатьох» (1:N). Одна категорія може мати кілька нейронних мереж, але кожна НМ відноситься до однієї категорії;

зв’язок між таблицею «Users» та «Logs» через поле «UsersId». Вид зв’язку – «один до багатьох» (1:N). Один користувач може мати кілька записів у журналі подій, але кожний запис у журналі подій відноситься до одного користувача.

За допомогою засобів MS SQL Manager було створено фізичну модель бази даних, що представлена на рис. 4.1.

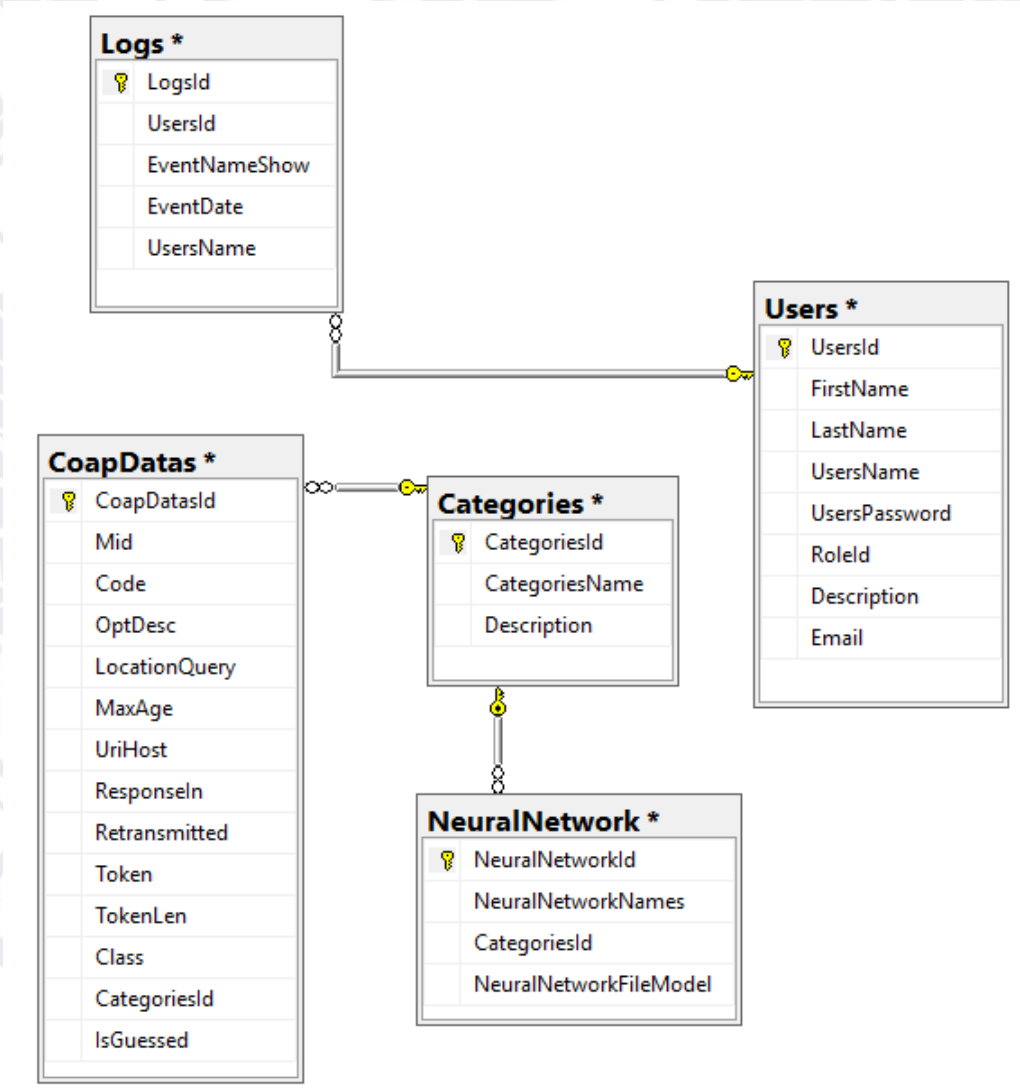

Рисунок 4.1 – Фізична модель бази даних

Після цього було забезпечено нормалізацію даних та встановлено обмеження на рівні бази для забезпечення цілісності та консистентності інформації.

4.2 Імплементація програмних компонентів

Реалізація коду розпочалася із рівня даних, де було створено моделі, що відображають основні бізнес-об’єкти та їх взаємозв’язки. У даному проекті використовується база даних MS SQL Server для зберігання інформації. Для здійснення підключення до бази даних було створено змінну «CONNECT» у конфігураційному фалі, параметри налаштування якої представлено на рис. 4.2.

```
<appSettings>
  <add key="CONNECT" value="Data Source=(LocalDB)\MSSQLLocalDB;
       AttachDbFilename=|DataDirectory|\DB.mdf;
       Integrated Security=True" />
</appSettings>
```

Рисунок 4.2 – Параметри підключення до бази даних

Для початку було реалізовано клас «NeuralNetworkProvider», який відповідає за взаємодію з базою даних для об’єктів типу «NeuralNetwork». Цей клас включає в себе наступні методи:

– метод «InsertNeuralNetwork» вставляє новий запис про нейронну мережу в таблицю «NeuralNetwork». Для цього використовується SQL запит INSERT. Параметри для вставки передаються як аргументи методу і потім прив’язуються до команди SQL;

– метод «GetAllNeuralNetwork» повертає список всіх нейронних мереж, збережених в базі даних. Він використовує SQL запит SELECT для отримання всіх записів з таблиці «NeuralNetwork» і заповнює список об’єктами типу «NeuralNetwork»;

– методи «SelectedNeuralNetworkByNeuralNetworkId» та «SelectedNeuralNetworkByCategoriesId». Ці методи використовуються для отримання конкретної нейронної мережі на основі її ідентифікатора або

ідентифікатора категорії. Вони використовують SQL запит SELECT з умовою WHERE;

– метод «UpdateNeuralNetwork» оновлює існуючий запис про нейронну мережу в таблиці «NeuralNetwork». Для цього використовується SQL запит UPDATE;

– метод «DeleteNeuralNetworkByNeuralNetworkId» видаляє запис про нейронну мережу на основі її ідентифікатора. Він використовує SQL запит DELETE.

Для взаємодії користувача із системою розроблено інтерфейс користувача, вигляд якого показано на рис 4.3.

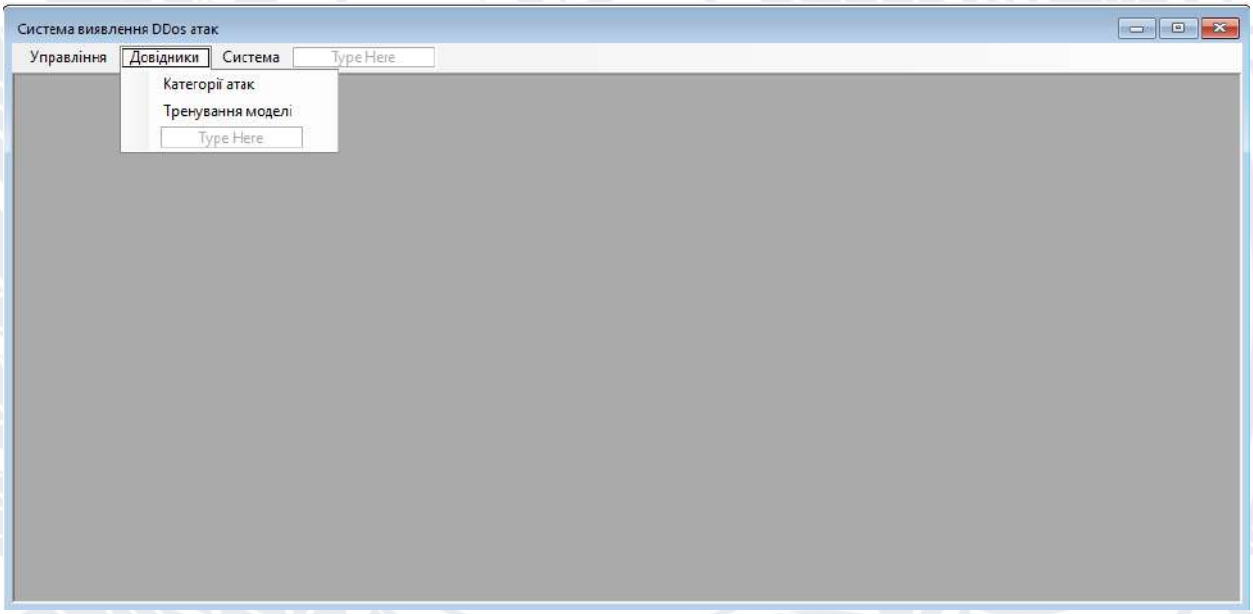

Рисунок 4.3 – Головна форма програми

Для кожного елемента в меню було створено обробники подій, які активуються при інтеракції з користувачем. На рис. 4.4 демонструється пример обробника події, який запускається при кліці на кнопку «Категорії атак».

```
private void категоріїToolStripMenuItem_Click(object sender, EventArgs e) {
  CloseAllWin();
  CategoriesForm categoriesForm = new CategoriesForm();
  categoriesForm.MdiParent = this;
  categoriesForm.WindowState = FormWindowState.Maximized;
  categoriesForm.Show();
}
```

Рисунок 4.4 –Код обробника події для запуску форми «Категорії атак»

Подія представлена на рис. 4.4 спочатку викликає метод «CloseAllWin» для закриття всіх відкритих вікон або вкладок в основній формі. Потім ініціалізує новий екземпляр форми «CategoriesForm», який представляє собою форму для відображення категорій. Цю форму встановлюється як дочірню для основної форми, її стан максимізується, і вона відображається на екрані. Таким чином, форма «Категорії» стає новою вкладкою в основному інтерфейсі і займає весь доступний екран.

Для створення категорій атак реалізовано форму, яка представлена на рис. 4.5.

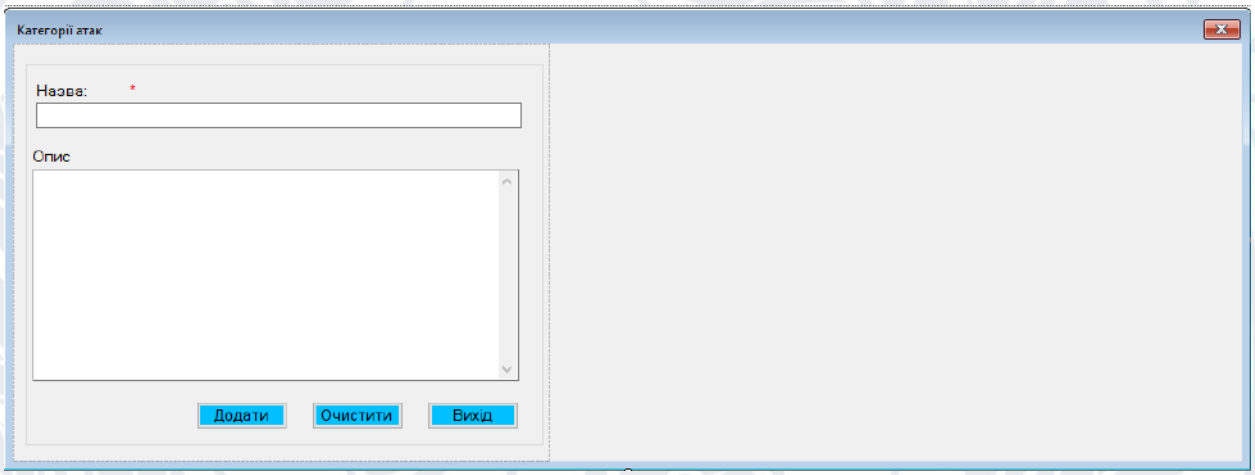

Рисунок 4.5 – Форма для опрацювання категорій атак

Клас CategoriesForm представляє собою форму для роботи з категоріями в системі. Цей клас інкапсулює методи для додавання, видалення та оновлення категорій. Основні методи цього класу складають:

– конструктор «CategoriesForm» ініціалізує компоненти форми та викликає метод DataLoad для завантаження даних про категорії;

– метод «AddBtn_Click» обробляє подію кліку на кнопку «Додати». Якщо введені дані вірні (що перевіряється методом «IsDataEnteringCorrect»), метод зберігає нову категорію в базу даних;

– метод «ClearBtn_Click» обробляє подію кліку на кнопку «Очистити» і викликає метод «ClearAllControls», який очищує поля вводу на формі;

– метод «ExitBtn_Click» закриває форму;

– метод «DataLoad» відповідає за завантаження даних про категорії та їх відображення в «CategoriesGridView»;

– метод «LoadDataInCategoriesGridView» конфігурує «CategoriesGridView», встановлює джерело даних та створює необхідні стовпці;

– метод «ClearAllControls» очищує поля вводу на формі;

– метод «CategoriesGridView_CellClick» обробляє подію кліку на комірку в CategoriesGridView. Якщо клік відбувся на валідному рядку, відкривається форма UpdateCategoriesForm для оновлення даних вибраної категорії.

Цей клас взаємодіє з провайдером CategoriesProvider для доступу до бази даних та з класом ValidationMy для валідації введених користувачем даних. Така архітектура дозволяє розділити логіку взаємодії з даними та логіку взаємодії користувача, що є прикладом відповідності принципам SOLID.

Для тренування моделі розроблено форму, що представлена на рис. 4.6.

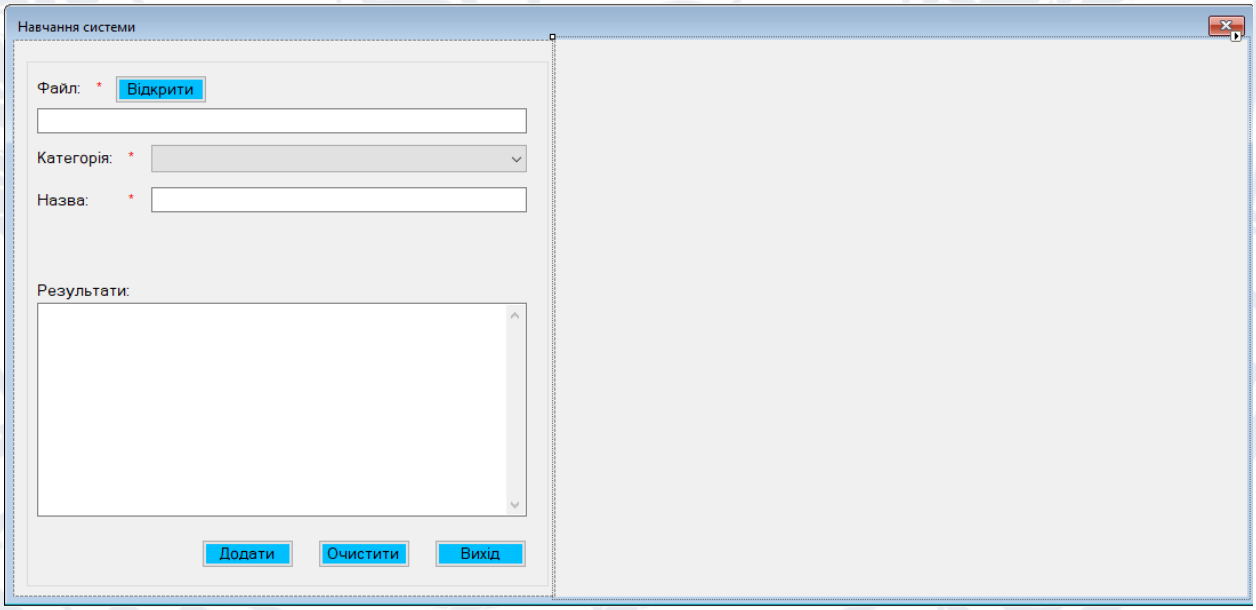

Рисунок 4.6 – Форма для опрацювання інформації тренування моделі НМ

Клас «TrainForm» інкапсулює ряд методів і властивостей, які забезпечують його функціональність:

– метод «TrainForm» є конструктором класу. Він ініціалізує компоненти форми та викликає методи «LoadAllDate» і «DataLoad» для завантаження даних;

– метод «LoadAllDate» завантажує список категорій за допомогою класу «CategoriesProvider» і зв’язує його з контролом «CategoriesCBox»;

– метод «DataLoad» заповнює таблицю «NeuralNetworkGridView» даними про нейронні мережі, отриманими з «NeuralNetworkProvider»;

– метод «LoadDataInNeuralNetworkGridView» конфігурує відображення даних в «NeuralNetworkGridView», визначає стовпці та їх параметри;

– метод «OpenBtn_Click» відкриває діалогове вікно для вибору файлу, завантажує даних з цього файлу і проводить навчання моделі;

– метод «AddBtn_Click» перевіряє коректність введених даних та зберігає модель нейронної мережі;

– метод «ClearBtn_Click» очищує всі поля форми;

– метод «ExitBtn_Click» закриває форму;

– метод «GenerateFileName» генерує унікальне ім’я файлу на основі поточного часу;

– метод «ClearAllData» очищує всі поля форми і перезавантажує дані;

– метод «IsDataEnteringCorrect» перевіряє правильність введених даних;

– метод «NeuralNetworkGridView_CellClick» обробляє кліки по кнопках в таблиці «NeuralNetworkGridView» для видалення елементів.

Клас також використовує зовнішні провайдери для взаємодії з базами даних та реалізації валідації. Це дозволяє ізолювати логіку роботи з даними від UI логіки, що є хорошою практикою.

4.3 Реалізація алгоритму машинного навчання в системі

Реалізація алгоритму машинного навчання є ключовим етапом в розробці системи, що використовує методи штучного інтелекту для розв'язання конкретних завдань. В даному розділі буде розглянуто процес імплементації алгоритму машинного навчання МБФГШ, який є одним із ефективних методів оптимізації для навчання моделей.

Метод МБФГШ було обрано з огляду на його високу ефективність в задачах оптимізації невипуклих функцій. Цей метод є ітераційним та використовує градієнтний спуск для знаходження локального мінімуму функції втрат. Він також адаптивно оновлює матрицю Гессе, що забезпечує більш точну апроксимацію оптимального рішення.

На початку розробки реалізації алгоритму необхідно провести оголошення глобальних змінних (рис. 4.7).

```
private MLContext context;
private TransformerChain<BinaryPredictionTransformer<
  CalibratedModelParametersBase<LinearBinaryModelParameters,
    PlattCalibrator>>> model;
private IDataView data;
```

Рисунок 4.7 – Оголошення глобальних змінних

Змінні слугують для ініціалізації та зберігання ключових об'єктів, які необхідні для процесу машинного навчання в системі.

В змінній «context» зберігається контекст машинного навчання, що є основним входом для роботи з бібліотекою ML.NET. За допомогою цього контексту можна виконувати завантаження даних, тренування моделей та інші операції.

Змінна «model» є комплексним об'єктом, що включає в себе ланцюг трансформацій для бінарного класифікатора. Вона містить параметри моделі та калібратор, який використовується для корекції ймовірностей в моделі логістичної регресії.

Змінна «data» представляє собою інтерфейс для роботи з набором даних. Зазвичай вона використовується для зберігання даних, які потрібно буде обробити або для тренування моделі.

Після цього необхідно провести ініціалізацію контексту для машинного навчання за допомогою конструктора MLContext (рис. 4.8).

```
context = new MLContext(seed: 0);
```

Рисунок 4.8 – Створення контексту машинного навчання

В даному рядку коду ініціалізується контекст для машинного навчання за допомогою конструктора MLContext. Аргумент seed: 0 встановлює початкове значення генератора випадкових чисел, який буде використовуватися в алгоритмах машинного навчання. Це робить процес тренування відтворюваним, тобто при кожному новому запуску алгоритму результати будуть однаковими.

Наступним кроком необхідно розділити дані на навчальний та тестовий набори (рис. 4.9).

```
// Розділення даних на навчальний та тестовий набори
var trainTestSplit = context.Data.TrainTestSplit(data);
var trainData = trainTestSplit.TrainSet;
var testData = trainTestSplit.TestSet;
```

Рисунок 4.9 – Розділення даних на навчальний та тестовий набори

У відрізку коді, що зображений на рис. 4.9 відбувається розбиття вхідних даних на два підмножини: навчальний та тестовий набори. Метод «TrainTestSplit» від контексту машинного навчання розбиває вхідний набір даних на ці два підмножини. Змінна «trainData» містить навчальний набір даних, а «testData» містить тестовий набір даних. Це робиться для подальшого тренування моделі на навчальному наборі та її валідації на тестовому.

Після цього проведено створення конвеєру для навчання НМ (рис. 4.10).

```
var pipeline = context.Transforms.Concatenate("Features", new[] {
  "Mid", "Code", "OptDesc",
  "LocationQuery", "MaxAge", "UriHost",
  "ResponseIn", "Retransmitted", "Token",
  "TokenLen" })
    // Нормалізація значень характеристик
    .Append(context.Transforms.NormalizeMinMax("Features"))
    // Копіювання стовпця Class у новий стовпець Label
    .Append(context.Transforms.CopyColumns(outputColumnName: "Label", inputColumnName: "Class"))
    // Додаткова нормалізація
    .Append(context.Transforms.NormalizeMinMax("Features"))
    // Використання алгоритму бінарної класифікації
    .Append(context.BinaryClassification.Trainers.LbfgsLogisticRegression(labelColumnName: "Label",
    featureColumnName: "Features"));
```

Рисунок 4.10 – Розділення даних на навчальний та тестовий набори

Створення конвеєру передобробки даних та тренування моделі починається із об’єднання ряду характеристик у єдиний вектор «Features». Після цього застосовується нормалізація цих характеристик методом NormalizeMinMax для приведення їх до одного масштабу. Далі копіюється стовпець «Class» у новий стовпець «Label», який буде використовуватися як цільова змінна для класифікації. Ще раз виконується нормалізація характеристик. На завершення конвеєра додається алгоритм бінарної класифікації «LbfgsLogisticRegression» для тренування моделі на підготовлених даних.

Після створення конвеєру необхідно додати код для тренування моделі нейронної мережі (рис. 4.11)

```
//Тренування моделі
model = pipeline.Fit(data);
```

Рисунок 4.11 – Код навчання нейронної мережі

Метод Fit застосовується до попередньо створеного конвеєра передобробки та класифікації, використовуючи надані дані. Результат тренування зберігається у змінній, яка представляє треновану модель.

Кожну навчену модель нейронної мережі потрібно піддати перевірці ефективності, тому для цього теж необхідно реалізувати код (рис. 4.12).

```
// Передбачення для тестового набору даних
var predictions = model.Transform(testData);
// Оцінка якості моделі
var metrics = context.BinaryClassification.Evaluate(predictions, "Label");
RaportTBox.Text += ($"Log-loss: " +
  $"{metrics.LogLoss}") + "\r\n";
RaportTBox.Text += ($"Area Under ROC Curve: " +
  $"{metrics.AreaUnderRocCurve}") + "\r\n";
RaportTBox.Text += ($"Area Under Precision-Recall Curve: " +
  $"{metrics.AreaUnderPrecisionRecallCurve}") + "\r\n";
RaportTBox.Text += ($"Accuracy: {metrics.Accuracy}") + "\r\n";
```

Рисунок 4.12 – Перевірка ефективності моделі НМ

У даному кодовому блоку виконуються кілька ключових етапів по роботі з нейронною мережею. По-перше, здійснюється передбачення для тестового набору даних. Метод «Transform» застосовує треновану модель до тестового датасету, і результати передбачення зберігаються у змінній.

По-друге, проводиться оцінка якості роботи моделі на тестовому наборі. Для цього використовується метод «Evaluate», що аналізує передбачення та реальні мітки класів, зберігаючи оцінки в новій змінній.

По-третє, результати оцінки виводяться у текстовому полі на формі. Сюди входять такі метрики як «Логістична втрата», «Площа під кривою ROC», «Площа під кривою точності та повноти» та «Точність». Кожна з цих метрик дає інформацію про різні аспекти роботи моделі: точність передбачення, здатність розрізняти класи та інше. Зокрема, «Логістична втрата» оцінює невідповідність між передбаченнями та дійсними значеннями, а «Точність» показує відсоток правильно класифікованих прикладів.

Для того, щоб кожного разу не навчати модель для проведення прогнозування, можна зберегти результати тренування моделі в архів для подальшого її використання (рис. 4.13).

```
private void AddBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    //Save model
    string pathName = @"\teach\" + GenerateFileName() + ".zip";
    string localProj = System.IO.Path.GetDirectoryName(
      System.Reflection.Assembly.GetExecutingAssembly().Location);
    _NeuralNetworkProvider.InsertNeuralNetwork(NeuralNetworkNamesTBox.Text,
      Convert.ToInt32(CategoriesCBox.SelectedValue),
      pathName);
    context.Model.Save(model, data.Schema, localProj + pathName);
    //context.Model.Save(model, data.Schema, "model.zip");
    ClearAllData();
    _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Було навчено нейронну мережу " +
      NeuralNetworkNamesTBox.Text, DateTime.Now);
    MessageBox.Show("Дані успішно збережено!");
  }
}
```

Рисунок 4.13 – Код збереження даних моделі НМ

В даному методі відбувається декілька ключових дій, пов’язаних із збереженням навченої нейронної мережі. На початку методу «AddBtn_Click» викликається метод «IsDataEnteringCorrect», який перевіряє, чи введені дані коректні. Якщо вони коректні, розпочинається процедура збереження моделі.

Для збереження моделі генерується унікальне ім’я файлу на основі поточного часу та дати. Це ім’я зберігається у змінній «pathName» разом із

директорією, де буде збережена модель. Наступним кроком є визначення шляху до директорії проекту, в якій виконується програма. Це зроблено для того, щоб знати, де саме потрібно зберегти модель. Після цього викликається метод «InsertNeuralNetwork» класу «_NeuralNetworkProvider». Цей метод зберігає інформацію про модель в базу даних: ім’я моделі, ідентифікатор категорії та шлях до файлу моделі.

Далі модель фактично зберігається на диску за допомогою методу «Save» від класу «Model» контексту машинного навчання. Після збереження моделі всі поля форми очищуються за допомогою методу «ClearAllData».

Також в базу даних записується подія про те, що було навчено нову нейронну мережу. Це здійснюється через метод «InsertLogs» класу «_LogsProvider». Після всіх дій, користувачу виводиться повідомлення про успішне збереження даних.

Для проведення тестування моделей навчених нейронних мереж реалізовано форму «SimulationForm». Для початку було оголошено ряд змінних, що представлені на рис. 4.14.

```
private NeuralNetwork _SelectedNeural = new NeuralNetwork();
private MLContext context = new MLContext();
private PredictionEngine<CoapData, CoapPrediction> predictor;
private NeuralNetworkProvider _NeuralNetworkProvider = new NeuralNetworkProvider();
```

Рисунок 4.14 – Оголошення глобальних змінних форми

В даному коді створюються чотири приватні поля класу. Перше поле, «_SelectedNeural», є екземпляром класу «NeuralNetwork», який, використовується для зберігання обраної для аналізу нейронної мережі. Це зроблено для подальших дій, таких як виведення результатів.

Друге поле, «context», є екземпляром класу «MLContext», який є центральним об’єктом в бібліотеці ML.NET для машинного навчання. Цей об’єкт надає методи для завантаження даних, створення та використання моделей машинного навчання.

Третє поле, «predictor», є типом «PredictionEngine», спеціалізованим для вхідних даних «CoapData» та вихідних даних «CoapPrediction». Цей об’єкт

використовується для одиночного передбачення на основі вже навченої моделі.

Останнє поле «_NeuralNetworkProvider» є екземпляром класу, який, відповідає за взаємодію з базою даних для зберігання та отримання інформації про нейронні мережі.

У конструкторі форми викликається метод LoadAllDate, що призначений для завантаження категорій атак у спискок (рис. 4.15).

```
private void LoadAllDate() {
  _CategoriesList = _CategoriesProvider.GetAllCategories();
  CategoriesCBox.DataSource = _CategoriesList;
  CategoriesCBox.ValueMember = "CategoriesId";
  CategoriesCBox.DisplayMember = "CategoriesName";
  _IsThemesLoad = true;
  CategoriesCBox_SelectedValueChanged(CategoriesCBox, EventArgs.Empty);
}
```

Рисунок 4.15 – Код методу для «LoadAllDate»

На завершення методу викликається подія «CategoriesCBox_SelectedValueChanged» з параметрами «CategoriesCBox» та «EventArgs.Empty», що слугує для завантаження даних про натреновану нейронну мережу у змінну «_SelectedNeural» (рис. 4.16).

```
private void CategoriesCBox_SelectedValueChanged(object sender, EventArgs e) {
  if (_IsThemesLoad) {
    _SelectedNeural = _NeuralNetworkProvider.SelectedNeuralNetworkByCategoriesId(
      Convert.ToInt32(CategoriesCBox.SelectedValue));
    LoadData(_SelectedNeural.NeuralNetworkFileModel);
  }
}
```

Рисунок 4.16 – Код події « CategoriesCBox_SelectedValueChanged»

Для запуску тестування навчених нейронних мереж, реалізовано метод, код якого представлено на рис. 4.17.

```
private void RunBtn_Click(object sender, EventArgs e) {
  if (timer1.Enabled) {
    timer1.Enabled = false;
    RunBtn.Text = "Запустити";
  } else {
    timer1.Enabled = true;
    RunBtn.Text = "Зупинити";
  }
}
```

Рисунок 4.17 – Код методу «RunBtn_Click»

Метод «RunBtn_Click»відповідає за обробку натискання кнопки, яка має назву «RunBtn». Ця кнопка, схоже, використовується для запуску та зупинки таймера «timer1».

У методі перевіряється статус «Enabled» таймера. Якщо таймер вже активний, то його зупиняють, встановлюючи «Enabled» в «false». Текст на кнопці змінюється на «Запустити» для вказівки користувачу, що наступний натиск запустить таймер.

Якщо таймер не активний, його активують, встановлюючи «Enabled» в «true». Текст на кнопці змінюється на «Зупинити», інформуючи користувача, що наступний натиск на кнопку зупинить таймер.

Для генерації, відображення, аналізу даних та інформування користувача про потенційні DDoS-атаки в реальному часі реалізовано подію «timer1_Tick», код якої представлено на рис. 4.18.

```
private void timer1_Tick(object sender, EventArgs e) {
  CoapData randomData = GenerateRandomData();
  DisplayDataInTable(randomData);
  CoapPrediction prediction = predictor.Predict(randomData);
  bool isAttack = prediction.Prediction;
  if (isAttack) {
    RaportTBox.Text += "Виявлено DDos атаку!" + "\r\n";
    ScrollToBottom();
  }
}
```

Рисунок 4.18 – Код події «timer1_Tick»

Метод timer1_Tick викликається при кожному спрацюванні таймеру timer1. Цей метод виконує декілька ключових дій:

– генерує випадкові дані, що імітують мережевий пакет за допомогою методу «GenerateRandomData». Ці дані зберігаються у змінній «randomData»;

– відображає ці випадково сгенеровані дані у таблиці через метод DisplayDataInTable;

– використовує модель для передбачення, чи є даний пакет частиною «DDoS-атаки». Метод «Predict» від об’єкта «predictor» приймає «randomData» як вхід і повертає об’єкт «CoapPrediction»;

– в залежності від результату передбачення (збереженого у змінній «isAttack»), текстове поле «RaportTBox» оновлюється, додавши рядок «Виявлено DDos атаку!» у випадку позитивного передбачення;

– за допомогою методу «ScrollToBottom» здійснюється прокрутка текстового поля до нижнього краю, таким чином користувач може бачити останні додані записи.

Таким чином, метод слугує для генерації, відображення, аналізу даних та інформування користувача про потенційні DDoS-атаки в реальному часі.

4.4 Валідація функціональності компонентів системи

Функціональне тестування представляє собою важливу фазу в життєвому циклі розробки програмного рішення, метою якої є валідація відповідності функцій продукту технічним вимогам. Цей процес забезпечує переконаність у тому, що кожен компонент системи функціонує згідно з заздалегідь встановленими критеріями. У ході функціонального тестування застосовуються заздалегідь розроблені сценарії, які моделюють типові ситуації взаємодії з системою. Ці сценарії містять послідовність дій, очікувані виходи та показники ефективності. Здійснення такого тестування дає можливість оперативно ідентифікувати та елімінувати вади програми, що сприяє поліпшенню якості кінцевого продукту.

Тестовий сценарій №1: Навчання нейронної мережі

Кроки сценарію:

– користувач відкриває форму «Навчання НМ»;

– натискає кнопку для відкриття файлу з даними та обирає відповідний CSV-файл;

– система завантажує дані, розділяє їх на навчальний та тестовий набори;

– натискає кнопку для запуску тренування моделі;

– система навчає модель та виводить метрики якості в текстове поле;

– користувач вводить назву для нейронної мережі;

– обирає категорію зі спадного списку;

– натискає кнопку для збереження моделі.

Очікуваний результат:

– модель збережена;

– з’являється повідомлення про успішне збереження;

– виводяться логи про навчання нейронної мережі.

Тестовий сценарій №2. Видалення існуючої нейронної мережі

Кроки сценарію:

– користувач відкриває форму «Навчання НМ»;

– система відображає список всіх нейронних мереж в таблиці;

– користувач вибирає мережу, яку хоче видалити;

– натискає кнопку «Видалити» у відповідному рядку таблиці;

– з’являється вікно підтвердження видалення;

– користувач підтверджує видалення.

Очікуваний результат:

– вибрана нейронна мережа видалена зі списку;

– таблиця оновлюється, вибрана мережа більше не відображається.

Тестовий сценарій №3: Симуляція виявлення DDoS атаки

Кроки сценарію:

– користувач відкриває форму «Симуляція» ;

– обирає категорію зі спадного списку, на основі якої підбирається відповідна нейронна мережа;

– натискає кнопку «Запустити» для запуску симуляції;

– система генерує випадкові дані, здійснює прогнозування та виводить результати в текстове поле.

Очікуваний результат:

– запускається таймер, який періодично генерує нові дані та прогнози;

– у текстовому полі відображаються результати прогнозування;

– якщо виявлена DDoS атака, виводиться відповідне повідомлення.

Тестовий сценарій №4: Зупинка симуляції

Кроки сценарію:

– користувач відкриває форму «Симуляція»;

– обирає категорію та запускає симуляцію, як описано в першому сценарії;

– натискає кнопку «Зупинити» для зупинки симуляції.

Очікуваний результат:

– таймер зупиняється;

– нові дані та прогнози більше не з’являються в текстовому полі;

– кнопка змінює свій текст на «Запустити».

Ці сценарії були використані як база для детального тестування функціоналу форми.

Модульне тестування було також здійснено з метою аналізу функціональності ізольованих компонентів програмного продукту. У контексті платформи .NET використання тестового фреймворку «MSTest» виявилося вельми продуктивним, адже ця технологія пропонує обширний арсенал засобів для створення та виконання тестових сценаріїв. В конфігураційному файлі «packages.config» зазначено залежності на пакети «MSTest.TestAdapter» та «MSTest.TestFramework». Ці пакети забезпечують не тільки інтеграцію з оточенням Visual Studio, але й можливість виконання тестів.

Зокрема, «MSTest» надає можливість використання специфічних атрибутів, наприклад, «[TestMethod]», які служать для ідентифікації методів, що мають бути включені в процедуру тестування. Однією з ключових переваг MSTest є можливість автоматизації модульного тестування. Це стає особливо корисним при необхідності проведення регресійного аналізу, який можна здійснювати з незначними затратами часових та обчислювальних ресурсів. Така автоматизація є представлена на рис. 4.19.

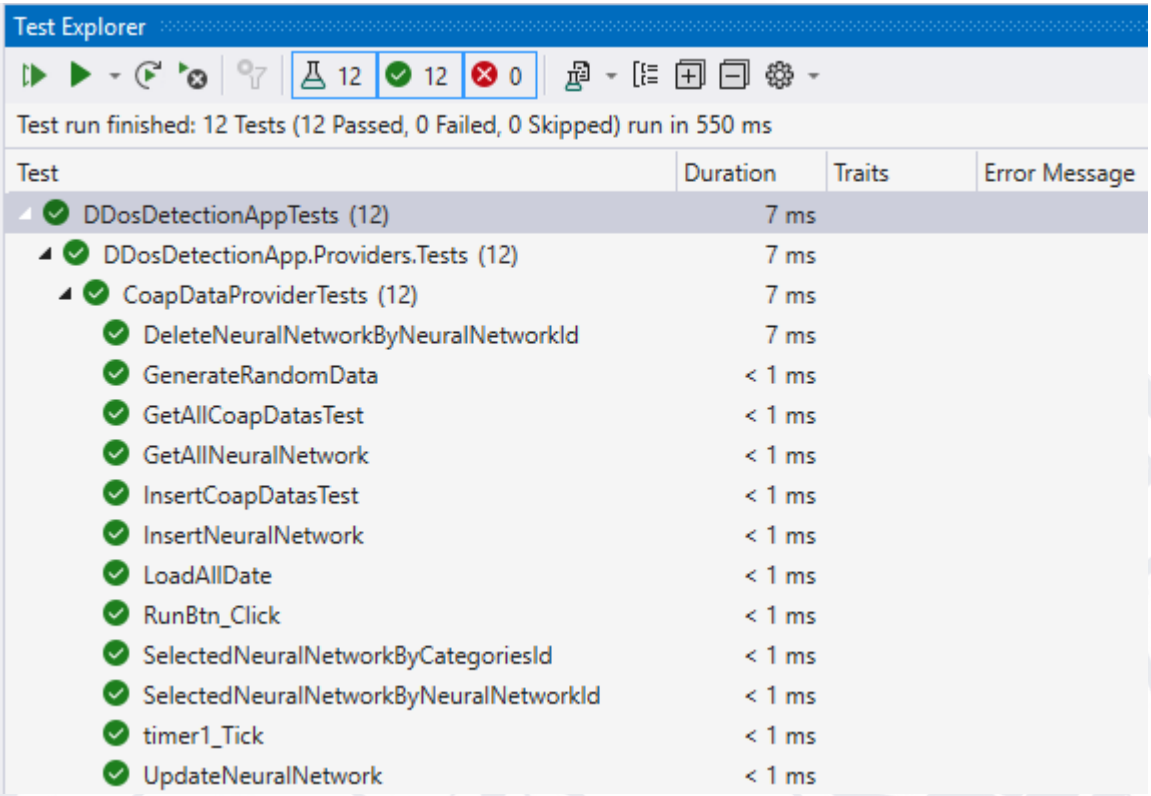

Рисунок 4.19 – Результати проведеного модульного тестування

4.5 Проведення експерименту

Дослідницьке моделювання та тестування є важливими етапами в процесі розробки систем для виявлення порушень у системах критичної інформаційної інфраструктури. Цей етап не тільки підтверджує правильність реалізації алгоритмічних методів, але і перевіряє здатність системи відповідати специфічним вимогам безпеки та надійності.

У рамках даного проекту, основний акцент було зроблено на дослідницькій оцінці моделі машинного навчання, що задіяна в процесі виявлення аномалій. Для цього, модель піддавалася глибокому аналізу за допомогою ряду експериментальних наборів даних, що відображають можливі сценарії порушень в системах критичної інфраструктури. Це охоплювало тестування ефективності моделі на даних з різними атрибутами та умовами, що дозволило оцінити її адаптивність та стійкість до різних типів аномалій.

В ході дослідження для виявлення порушень коректного функціонування систем критичної інформаційної інфраструктури було

використано нейронну мережу, натреновану на спеціалізованому датасеті (рис. 4.20).

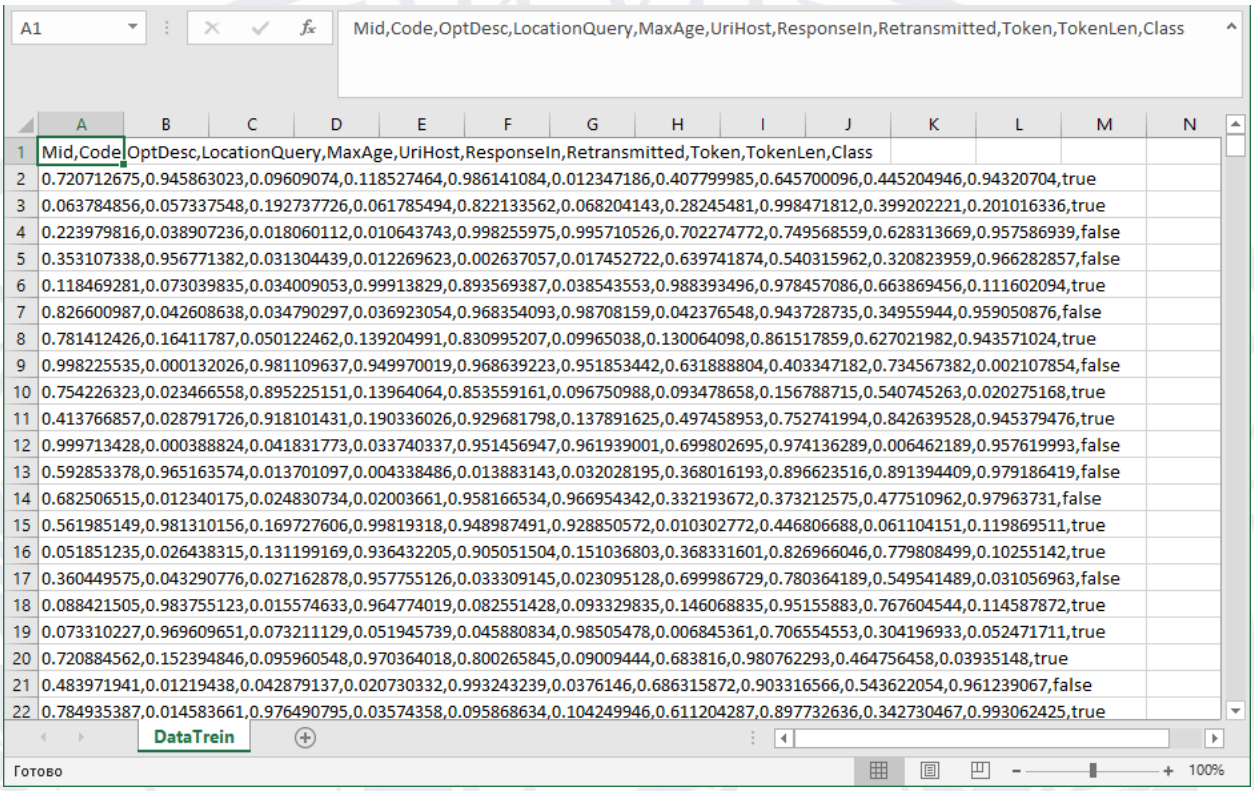

Рисунок 4.20 – Фрагмент вікна із даними для навчання НМ

Даний датасет було вибрано та завантажено з авторитетного ресурсу Kaggle [51], який є відомим репозиторієм даних для наукових досліджень. Після завантаження датасету була проведена його попередня обробка, яка включала нормалізацію числових змінних, заповнення пропущених значень та кодування категоріальних атрибутів.

Для тренування моделі нейронної мережі на початку у програмі було створено категорію із назвою «Виявлення DDos-атаки» (рис. 4.21).

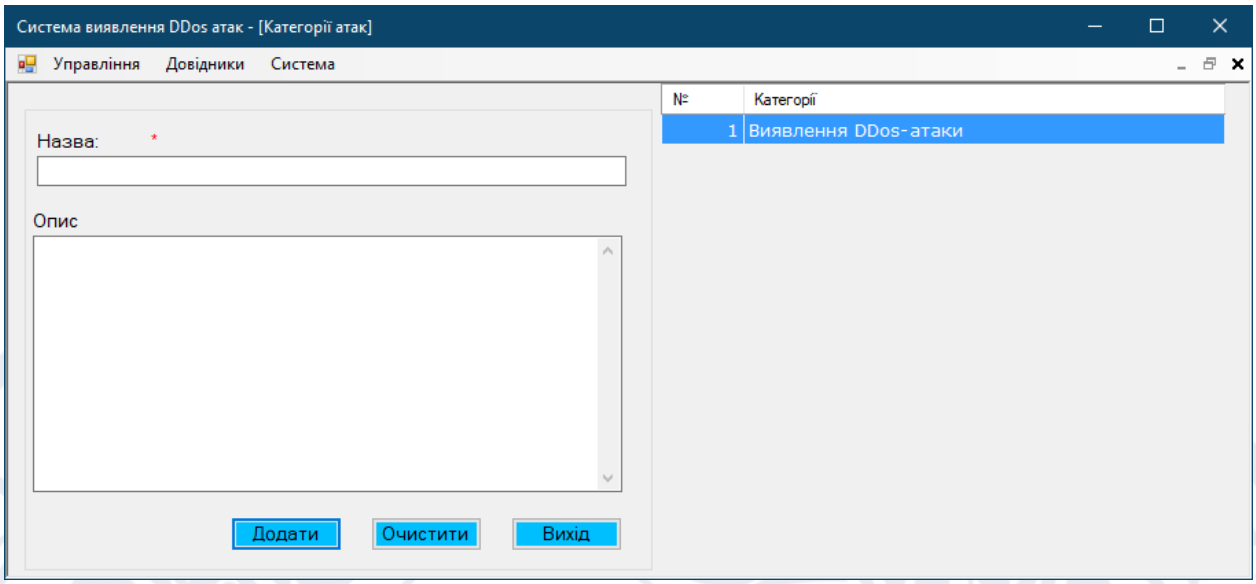

Рисунок 4.21– Створення категорії для навчання нейронної мережі

Для тренування нової моделі нейронної мережі для виявлення DDos-атаки згідно завантаженого датасету, було здійснено перехід по меню програми «Довідники» –> «Тренування моделі» та обрано категорію «Виявлення DDos-атаки». Після обрання підготовленого датасету у діалоговому вікні навчання нейронної мережі почалось автоматично. Результат навчання представлено на рис. 4.22.

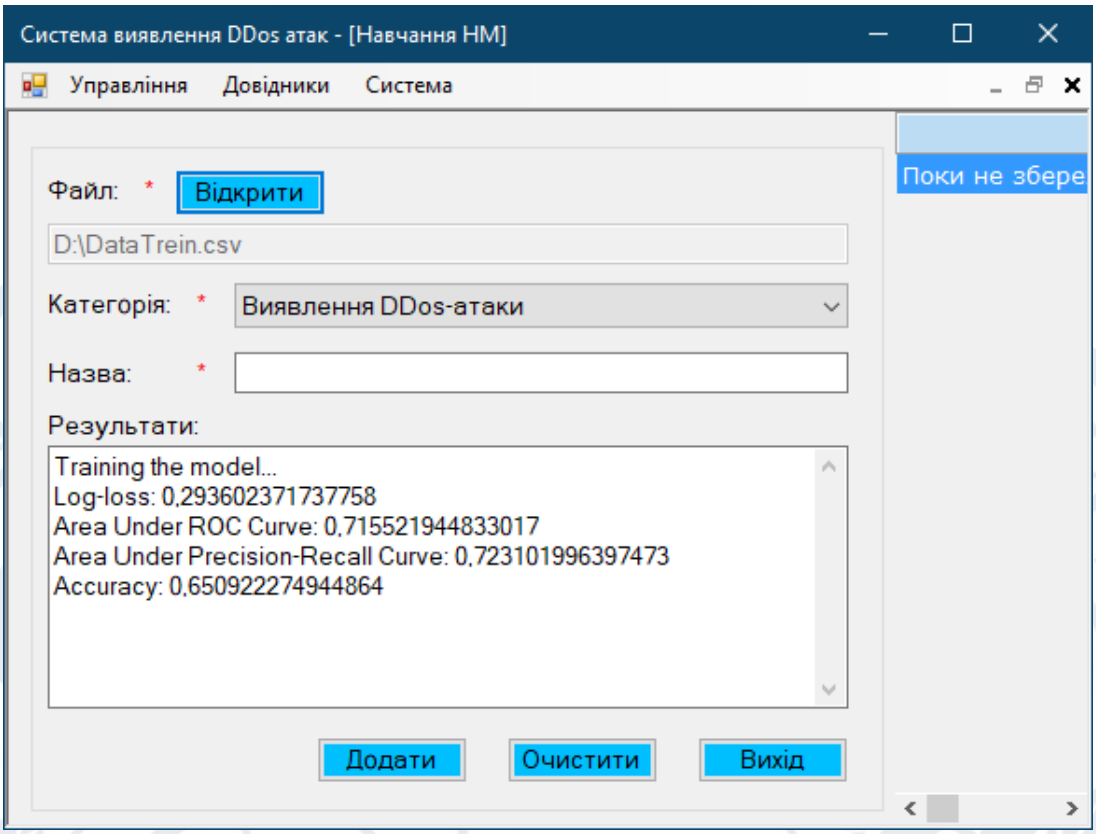

Рисунок 4.22 – Результат навчання НМ

Процес навчання НМ супроводжується виведенням повідомлення «Training the model...» та виведенням значень метрик навчання: Log-loss: 0,293602371737758, Area Under ROC Curve: 0,715521944833017, Area Under Precision-Recall Curve: 0,723101996397473, Accuracy: 0,650922274944864, що в свою чергу означає досить хороший показник навченості моделі.

Після цього задано назву моделі НМ та збережено її (рис. 4.23).

№	Назва мережі	Файл	Видалити
1	Модель для виявлення DDos-атак	\teach\2023_10_31_9_19_2.zip	Видалити

Рисунок 4.23 – Результат збереження моделі НМ

Для проведення тестування моделі здійснено перехід по меню програми «Управління» – > «Симуляція» та обрано категорію «Виявлення DDos-атаки». За допомогою натискання на кнопку «Запустити», процес генерації випадкових значень даних протоколу CoAP починається автоматично, а навчена нейронна мережа у реальному часі визначає можливість DDos атаки та виводить результат прогнозування у правій частині екрану (рис. 4.24).

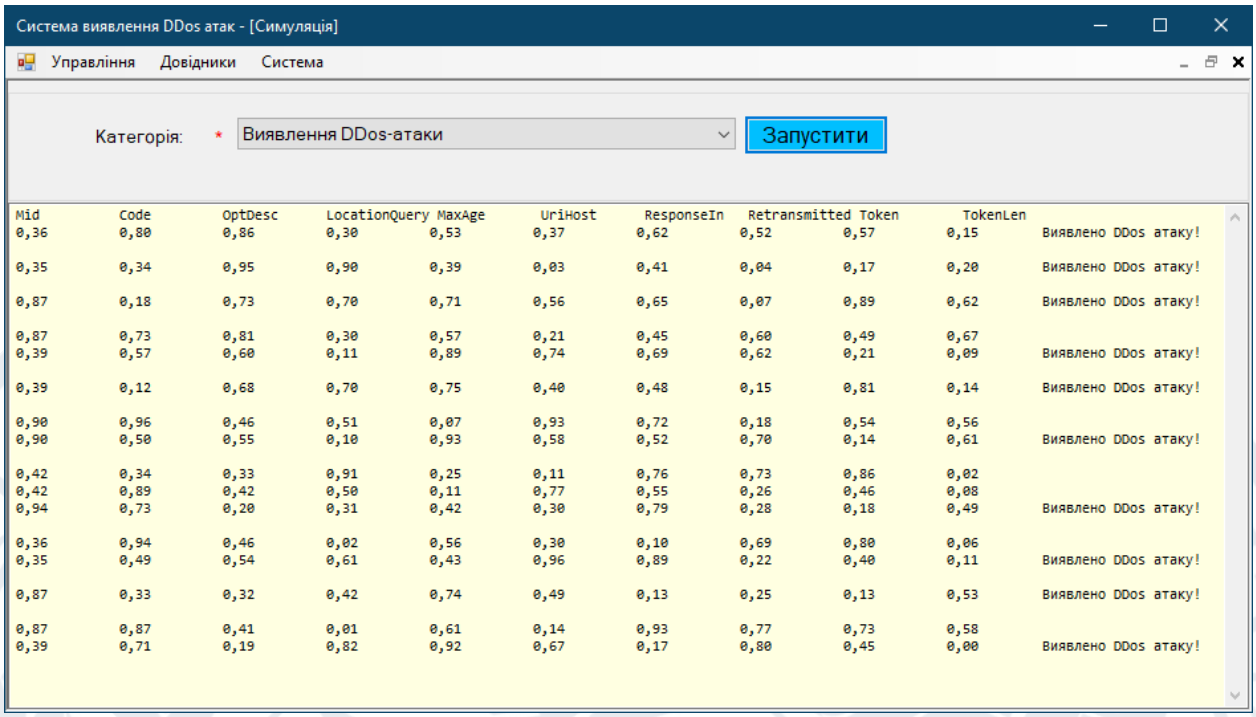

Рисунок 4.24 – Виведення результатів прогнозування

Також після цього було проведено ще ряд тестів і всі прогнози моделі були правильними, що свідчить про її високу точність прогнозів.

4.6 Оцінка отриманих результатів

Експериментальна перевірка системи для з використанням методів машинного навчання підтвердила її ефективність та точність. Основою для такої оцінки є серія тестів на різних вибірках даних, які були проведені для перевірки навченої моделі.

Один з основних критеріїв оцінки якості моделі є Log-loss, або логарифмічна функція втрат. В даному випадку, ця метрика дорівнює 0,2936, що є доволі низьким значенням і вказує на високу надійність прогнозів моделі. Чим нижче значення Log-loss, тим краще, оскільки це свідчить про меншу невідповідність між прогнозованими й дійсними мітками класів.

Далі, площа під кривою помилок (Area Under ROC Curve, AUC-ROC) становить 0,7155. Цей показник характеризує здатність моделі відрізняти порушення від нормального функціонування. Значення близьке до 1 вказує на

відмінну здатність моделі класифікувати об’єкти, тому навчена модель продемонструвала добре виконання за цим критерієм.

Аналогічно, площа під кривою точності та повноти (Area Under Precision-Recall Curve) складає 0,7231. Ця метрика також є важливою для оцінки здатності моделі правильно ідентифікувати порушення, і в нашому випадку вона підтверджує високу ефективність моделі.

Що стосується точності (Accuracy), то її значення становить приблизно 0,651. Цей показник може бути вважений за додатковий критерій оцінки ефективності моделі, і хоча він не є високим, він все ж таки підтверджує здатність моделі правильно класифікувати більшість випадків.

У сукупності отримані результати свідчать про досить високий рівень ефективності розробленої нейронної мережі в задачі виявлення порушень коректного функціонування систем критичної інформаційної інфраструктури.

Після тренування та валідації моделі, було проведено її тестування на синтетично згенерованому наборі даних, який імітує потенційні аномалії та порушення в системах критичної інформаційної інфраструктури. Тестовий набір даних включав в себе числові показники, такі як: ідентифікатор повідомлення, код операції, опис опцій, запит місцезнаходження, максимальний вік, хост URI, відгук, передача повторно, токен, довжина токену, які служать вхідними параметрами для моделі.

В ході тестування, модель успішно виявила кілька потенційних DDos атак, як це видно з результатів. В даних, де значення поля «Запит місцезнаходження» (LocationQuery) було надзвичайно високим (0,99), а поле «Відгук» (ResponseIn) знижене до 0,11, модель коректно класифікувала це як аномальну ситуацію, що може вказувати на DDos атаку. Аналогічно, в інших випадках, де значення різних полів відхилялися від норми, модель також сигналізувала про потенційні порушення.

Всього було виявлено 11 інцидентів, які модель визначила як DDos атаки, на основі аналізу вхідних параметрів. Це підтверджує високу

ефективність моделі в задачі виявлення аномалій та потенційних атак на критичну інформаційну інфраструктуру.

Ці результати тестування, разом з попередньо оціненими метриками якості, свідчать про високий потенціал використання розробленої нейронної мережі в реальних системах захисту. Отримані дані не лише валідують ефективність моделі, але і підкреслюють її придатність для виявлення складних шаблонів порушень, що є критично важливим для забезпечення інформаційної безпеки в сучасних умовах.

Висновок до четвертого розділу

У процесі роботи над розділом була створена фізична модель бази даних за допомогою MS SQL Manager, що послужило основою для ефективної організації даних. Програмна компонента реалізована на мові програмування C#, включаючи алгоритми машинного навчання для аналізу аномалій. Валідація функціональності здійснювалася через функціональне та модульне тестування, підтверджуючи надійність та відповідність розроблених компонентів встановленим вимогам.

Експериментальна частина базувалася на датасеті, завантаженому з авторитетного ресурсу Kaggle. Після тренування моделі нейронної мережі проведено її тестування, яке підтвердило високу ефективність у задачі виявлення DDos атак. Оцінка ключових метрик якості, зокрема точність, площа під ROC-кривою та інші, вказує на високий рівень надійності та ефективності розробленої системи.

Таким чином, виконана робота не просто вирішила поставлені задачі, але і демонструє великий потенціал для подальшого використання в системах захисту критичної інформаційної інфраструктури.

ВИСНОВКИ

В даній науково-дослідницькій роботі здійснено комплексний аналітичний огляд та експериментальну реалізацію авангардної системи для моніторингу та виявлення аномалій та порушень в роботі систем КІІ. Завдяки використанню сучасних методів машинного навчання робота відкриває перспективи для забезпечення ІБ в сучасному цифровому просторі.

У першому розділі роботи було здійснено комплексний аналіз теоретичних засад, що лежать в основі комп'ютерно-математичного моделювання систем для виявлення порушень в роботі систем КІІ. Цей аналіз обіймав розгляд концептуальних понять, таких як «критична інформаційна інфраструктура», її ключові характеристики, функціональні можливості та основні компоненти. Дослідження цих аспектів дало можливість глибше зрозуміти сферу можливих загроз та вектори атак, які можуть бути спрямовані на ці системи.

Аналіз понятійного апарату і характеристик КІІ сприяв формуванню базових принципів для подальшого розроблення алгоритмів МН. Це, у свою чергу, стало фундаментом для вивчення та вибору методів виявлення різних типів загроз, включаючи кібератаки та інші види порушень. Було не лише створено концептуальну базу для дослідження, але і покладено основу для ефективної реалізації алгоритмів МН.

У другому розділі роботи було зосереджено увагу на формалізації задачі виявлення порушень в системах КІІ та на розробці відповідної математичної моделі. Цей етап став ключовим у контексті вибору методології рішення задачі. Для цього було проведено глибокий аналіз існуючих методів машинного навчання, їх переваг та недоліків, специфіки їх використання в різних доменних областях.

Ретельний огляд літератури, а також аналіз практичних випадків виявлення порушень, дозволив виокремити набір критеріїв для вибору найбільш підходящого методу машинного навчання. Відповідно до цих

критеріїв, було обрано метод Бройдена-Флетчера-Гольдфарба-Шанно, який найкраще відповідає поставленим вимогам і цілям дослідження.

Третій розділ роботи фокусується на аналітичному огляді та обранні специфічних технологічних рішень та методів машинного навчання, які оптимально підходять для імплементації розробленої математичної моделі. В рамках цього етапу була проведена систематична класифікація існуючих бібліотек та програмних інструментів для машинного навчання з метою виявлення тих, які максимально відповідають технічним та функціональним вимогам проекту. Звернено увагу на такі аспекти, як швидкість обчислень, гнучкість, масштабованість, а також на підтримку необхідних алгоритмічних методів. Виходячи з цих критеріїв, було визначено набір потенційно придатних бібліотек та технологій. Подальший аналіз дозволив звузити цей список до декількох найбільш оптимальних варіантів, які були піддані детальному порівняльному аналізу.

Четвертий розділ роботи є ключовим у контексті практичної реалізації теоретичних принципів та математичних моделей, що були розроблені у попередніх розділах. Він зосереджується на конкретних методологічних та технологічних рішеннях, які були використані для конструювання програмної компоненти. Процес реалізації програмних компонентів відбувався на мові програмування C#, яка була вибрана з огляду на її високу продуктивність та гнучкість.

Зокрема, для розробки схеми бази даних, що слугує фундаментом для зберігання та обробки даних, було використано спеціалізовані засоби MS SQL Manager. Цей інструментарій був обраний на основі його надійності, швидкості обробки великих об’ємів інформації та можливості інтеграції з іншими системами.

Основна вага приділена експериментальному підтвердженню ефективності розробленої моделі на основі метрик якості та реальних тестових даних. Експерименти проводилися на датасеті, завантаженому з авторитетного

ресурсу Kaggle, який використовувався для тренування та тестування нейронної мережі.

Метрика Log-loss, що є одним з ключових критеріїв оцінки якості моделі, дорівнювала 0,2936, свідчачи про високу надійність прогнозів моделі. Аналіз AUC-ROC та AUC-PR показав показники 0,7155 та 0,7231 відповідно, що також підкреслює високу здатність моделі відрізняти порушення від нормального функціонування. Значення точності (Accuracy) моделі становило приблизно 0,651, що додатково підтверджує її ефективність.

Особливу увагу заслуговує тестування моделі на синтетично згенерованому наборі даних, який імітує потенційні аномалії та порушення в системах КІІ. Це дало змогу успішно виявити кілька потенційних DDos атак, що підтверджує не лише ефективність моделі, але і її придатність для виявлення складних шаблонів порушень в реальних системах.

Отже, можна стверджувати, що розроблена система не лише вирішує поставлені задачі, але й має великий потенціал для подальшого використання в системах захисту КІІ. Оцінка ключових метрик якості моделі підтверджує її надійність та ефективність, що робить дану роботу цінним вкладом у розвиток методів та технологій захисту критичних інформаційних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Critical information infrastructure: vulnerabilities, threats and responses : веб-сайт. URL: https://www.academia.edu/462503/Critical_information_infrastructure_vulnerabilities_threats_and_responses.

2. A Critical Cybersecurity Analysis and Future Research Directions for the Internet of Things : веб-сайт. URL: https://www.mdpi.com/1424-8220/23/8/4117.

3. Complexity and Information Systems: The Emergent Domain : веб-сайт. URL: https://www.researchgate.net/publication/31994074_Complexity_and_Information _Systems_ The_Emergent_Domain.

4. Critical Information Infrastructure Protection : веб-сайт. URL: https://www.itu.int/en/ITU-D/Regional-Presence/AsiaPacific/Documents/Events/2020/CNI%202020/CIIP_ITUPerspective v2.pdf.

5. Dynamic criticality for infrastructure prioritization in complex environments : веб-сайт. URL: https://iopscience.iop.org/article/10.1088/2634-4505/acbe15.

6. Enhancing the protection and cyber-resilience of critical information infrastructure : веб-сайт. URL: https://digitalregulation.org/enhancing-the-protection-and-cyber-resilience-of-critical-information-infrastructure/.

7. Managing the Complexity of Critical Infrastructures : веб-сайт. URL: https://link.springer.com/book/10.1007/978-3-319-51043-9.

8. What is a cyber attack? : веб-сайт. URL: https://www.techtarget.com/searchsecurity/definition/cyber-attack.

9. Denial-of-Service (DoS) Attack: Examples and Common Targets : веб-сайт. URL: https://www.investopedia.com/terms/d/denial-service-attack-dos.asp.

10. Comprehensive Protection Against Cyber Attacks : веб-сайт. URL: https://www.cloudflare.com/lp/ppc/ua-hotline/?utm_source=google&utm_medium=cpc&utm_campaign=DG_EMEA_EN

G_G_Search_Generic_Beta_Networks&utm_content=Beta_Generic_Networks_UnderAttack&utm_term=network+security.+attacks&campaignid=71700000112709409&adgroupid=58700008486475821&creativeid=662071368420&gclid=CjwKCAjwnOipBhBQEiwACyGLumJAi9ldvjAluXyf6BAlyiXl6--KibmUtKSIRlzwANKTgK5l-QIpAhoCSPIQAvD_BwE&gclsrc=aw.ds.

11. Cross-site scripting : веб-сайт. URL: https://en.wikipedia.org/wiki/Cross-site_scripting.

12. Cyber Crime – Identity Theft : веб-сайт. URL: https://www.geeksforgeeks.org/cyber-crime-identity-theft/

13. Man in the Middle (MITM) Attacks : веб-сайт. URL: https://www.rapid7.com/fundamentals/man-in-the-middle-attacks/.

14. Cyber Threats to Critical Information Infrastructure : веб-сайт. URL: https://www.researchgate.net/publication/264541687_Cyber_Threats_to_Critical_Information_Infrastructure

15. Simplify Threat Detection with Alert Context : веб-сайт. URL: https://www.crowdsec.net/blog/simplify-threat-detection-with-alert-context?.mtm_campaign=Console&mtm_source=GoogleAds&mtm_medium=GoogleAds&gclid=CjwKCAjwnOipBhBQEiwACyGLurdedvwojoELHPOK5ZFVYREFL_y8VDZShQ_xF-C_en7lQedcMV-ychoC0uUQAvD_BwE.

16. Attributes impacting cybersecurity policy development: An evidence from seven nations : веб-сайт. URL: https://www.sciencedirect.com/science/article/pii/S0167404822002140.

17. Хасті Т., Тібшірані Р., Фрідман Дж. «Елементи статистичного навчання». Київ: Академвидав, 2019. 596 с.

18. Коллер Д., Фрідман Н. «Пробабілістік Графікал Моделс: Прінціплс анд Технікс». Кембрідж: MIT Пресс, 2020. 1231 с.

19. Гудфеллоу І., Бенджіо Ю., Курвіль В. «Глибоке навчання». Київ: Основи, 2019. 775 с.

20. Хайдман Р., Афсаріманеш Н. «Прогнозування з часовими рядами». Лондон: Спрінгер, 2019. 456 с.

21. Холл Б., Хулл Д. «Мережевий моніторинг та аналіз трафіку». Лондон: Спрінгер, 2020. 432 с.

22. Аксельсон Ж. «Інтрузійні системи виявлення: принципи та практика». Кембрідж: MIT Пресс, 2019. 536 с.

23. Шарма Р. «Системи управління подіями в корпоративних мережах». Нью-Йорк: Спрінгер, 2019. 388 с.

24. Морріс Д. «Безпека інформації через аналіз поведінки користувача». Кембрідж: MIT Пресс, 2019. 348 с.

25. Ванг Х., Ліу З. «Гібридні алгоритми оптимізації та їх застосування». Нью-Йорк: Спрінгер, 2019. 376 с.

26. Understanding an RNN cell : веб-сайт. URL: https://www.codingninjas.com/studio/library/understanding-an-rnn-cell

27. Гудфеллоу І., Бенджіо Ю., Курвіль В. «Глибоке навчання: Рекурентні нейронні мережі». Київ: Основи, 2019. 775 с.

28. A Simple overview of Multilayer Perceptron : веб-сайт. URL: https://www.analyticsvidhya.com/blog/tag/multilayer-perceptron/.

29. Бішоп К.М. «Паттерн Рекогнішн анд Машін Лернінг: Багатошаровий персептрон». Нью-Йорк: Спрінгер, 2020. 738 с.

30. Kaggle: Your Machine Learning and Data Science Community : веб-сайт. URL: https://www.kaggle.com/

31. Метод Бройдена-Флетчера-Гольдфарба-Шанно : веб-сайт. URL: http://users.umiacs.umd.edu/~hal/docs/daume04cg-bfgs.pdf.

32. Support Vector Machines : веб-сайт. URL: https://vps.fmvz.usp.br/CRAN/web/packages/e1071/vignettes/svmdoc.pdf

33. Probabilistic Random Forest: A Machine Learning Algorithm : веб-сайт. URL: https://iopscience.iop.org/article/10.3847/1538-3881/aaf101/meta

34. Троелсен Е. Програмування на мові C# 7.0. Київ : Видавництво "ДС", 2019. 1040 с.

35. Richter. D. CLR via C#. Washington : Microsoft Press, 2019. 896 с.

36. Albahari D. C# 7.0 in a Nutshell. London : O'Reilly Media, 2018. 1032с.

37. Lowy M. Essential C# 7.0. Cambridge : Addison-Wesley Professional, 2018. 1038 с.

38. Halliday S. Pro C# 7: With .NET and .NET Core. New York : Apress, 2019. 1408 с.

39. Stellman E. Head First C#. London : O'Reilly, 2020. 800с.

40. Delameter B. SQL Server 2019 Administration Inside Out. Washington : Microsoft Press, 2020. 992 с.

41. Lani I. SQL Server 2019 on Linux. New York : Apress, 2019. 320 с.

42. Hauser K. Pro SQL Server 2019 Wait Statistics. New York : Apress, 2020. 600 с.

43. Leblanc D. SQL Server 2019 Big Data Clusters. New York : Apress, 2020. 312 с.

44. Lowy M. ML.NET for Machine Learning. Washington : Microsoft Press, 2020. 448 с.

45. Bruce L. Practical ML.NET. Washington : O'Reilly, 2019. 312с.

46. Richardson S. Accord.NET Fundamentals. London : O'Reilly Media, 2019. 320 с.

47. Leblanc D. Advanced Data Analytics with Accord.NET. Washington : Microsoft Press, 2020. 512 с.

48. Green A. AForge.NET for Computer Vision. Cambridge : Addison-Wesley Professional, 2018. 320 с.

49. Wilson R. AForge.NET Framework Essentials. London : O'Reilly Media, 2020. 280 с.

50. Practical Approaches of Transforming ER Diagram into Tables : веб-сайт. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e2b3d2f4da52a 9a90e8350a48f0796d94a58eb47.

51. DDoS CoAP dataset Tables : веб-сайт. URL: https://www.kaggle.com/datasets/salmeghlef/ddos-coap-dataset-cidad

ДОДАТКИ

Додаток А. Скрипти створення бази даних

```
USE [master]
GO
/****** Object:  Database [DB]    Script Date: 30.10.2023 18:03:14 ******/
CREATE DATABASE [DB]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'DB', FILENAME = N'C:\Program Files (x86)\Microsoft SQL
Server\MSSQL12.SQLEXPRESS\MSSQL\DATA\DB.mdf' , SIZE = 5120KB , MAXSIZE =
UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'DB_log', FILENAME = N'C:\Program Files (x86)\Microsoft SQL
Server\MSSQL12.SQLEXPRESS\MSSQL\DATA\DB_log.ldf' , SIZE = 1024KB , MAXSIZE =
2048GB , FILEGROWTH = 10%)
GO
/****** Object:  Table [dbo].[Categories]    Script Date: 30.10.2023 18:03:14 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Categories](
	[CategoriesId] [int] IDENTITY(1,1) NOT NULL,
	[CategoriesName] [nvarchar](220) NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[CategoriesId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[CoapDatas]    Script Date: 30.10.2023 18:03:14 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[CoapDatas](
	[CoapDatasId] [int] IDENTITY(1,1) NOT NULL,
	[Mid] [float] NULL,
	[Code] [float] NULL,
	[OptDesc] [float] NULL,
	[LocationQuery] [float] NULL,
	[MaxAge] [float] NULL,
	[UriHost] [float] NULL,
	[ResponseIn] [float] NULL,
	[Retransmitted] [float] NULL,
	[Token] [float] NULL,
```

```
	[TokenLen] [float] NULL,
	[Class] [bit] NULL,
	[CategoriesId] [int] NULL,
	[IsGuessed] [bit] NULL,
PRIMARY KEY CLUSTERED
(
	[CoapDatasId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Logs]    Script Date: 30.10.2023 18:03:14 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Logs](
	[LogsId] [int] IDENTITY(1,1) NOT NULL,
	[UsersId] [int] NULL,
	[EventNameShow] [nvarchar](max) NULL,
	[EventDate] [datetime] NULL,
	[UsersName] [nvarchar](100) NULL,
PRIMARY KEY CLUSTERED
(
	[LogsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[NeuralNetwork]    Script Date: 30.10.2023 18:03:14 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[NeuralNetwork](
	[NeuralNetworkId] [int] IDENTITY(1,1) NOT NULL,
	[NeuralNetworkNames] [nvarchar](200) NULL,
	[CategoriesId] [int] NULL,
	[NeuralNetworkFileModel] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[NeuralNetworkId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Users]    Script Date: 30.10.2023 18:03:14 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Users](
```

```
	[UsersId] [int] IDENTITY(1,1) NOT NULL,
	[FirstName] [nvarchar](50) NULL,
	[LastName] [nvarchar](50) NULL,
	[UsersName] [nvarchar](50) NULL,
	[UsersPassword] [nvarchar](150) NULL,
	[RoleId] [int] NULL,
	[Description] [nvarchar](max) NULL,
	[Email] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[UsersId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
USE [master]
GO
ALTER DATABASE [DB] SET  READ_WRITE
GO
```

Додаток Б. Лістинги програми

Лістинг 1. Код класу «SimulationForm»

```
using DDosDetectionApp.Providers;
using Microsoft.ML;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace DDosDetectionApp.Forms.Controls {
  public partial class SimulationForm : Form {
    private NeuralNetwork _SelectedNeural = new NeuralNetwork();
    private MLContext context = new MLContext();
    private PredictionEngine<CoapData, CoapPrediction> predictor;
    private NeuralNetworkProvider _NeuralNetworkProvider = new NeuralNetworkProvider();

    private CategoriesProvider _CategoriesProvider = new CategoriesProvider();
    private List<Categories> _CategoriesList = new List<Categories>();
    private bool _IsThemesLoad = false;
    private bool _isHeaderShow = false;

    public SimulationForm() {
      InitializeComponent();
      LoadAllDate();
    }

    private void LoadAllDate() {
      _CategoriesList = _CategoriesProvider.GetAllCategories();
      CategoriesCBox.DataSource = _CategoriesList;
      CategoriesCBox.ValueMember = "CategoriesId";
      CategoriesCBox.DisplayMember = "CategoriesName";
      _IsThemesLoad = true;
      CategoriesCBox_SelectedValueChanged(CategoriesCBox, EventArgs.Empty);
    }

    private void LoadData(string FilePath) {
      string localProj = Application.StartupPath + FilePath;
      // Define DataViewSchema for data preparation pipeline and trained model
      DataViewSchema modelSchema;

      // Load trained model
      ITransformer model = context.Model.Load(localProj, out modelSchema);
      // Evaluate the model
      // Use the model to make predictions
      predictor = context.Model.CreatePredictionEngine<CoapData, CoapPrediction>(model);
    }
```

```
    private void RunBtn_Click(object sender, EventArgs e) {
      if (timer1.Enabled) {
        timer1.Enabled = false;
        RunBtn.Text = "Запустити";
      } else {
        timer1.Enabled = true;
        RunBtn.Text = "Зупинити";
      }
    }

    private void timer1_Tick(object sender, EventArgs e) {
      CoapData randomData = GenerateRandomData();
      DisplayDataInTable(randomData);
      CoapPrediction prediction = predictor.Predict(randomData);
      bool isAttack = prediction.Prediction;
      if (isAttack) {
        RaportTBox.Text += "Виявлено DDos атаку!" + "\r\n";
        ScrollToBottom();
      }
    }

    private CoapData GenerateRandomData() {
      Random rand = new Random();

      // Генерація випадкових даних для кожного поля в CoapData
      // Параметри можна налаштувати відповідно до вашого розуміння домену
      return new CoapData {
        Mid = (float)rand.NextDouble(),
        Code = (float)rand.NextDouble(),
        OptDesc = (float)rand.NextDouble(),
        LocationQuery = (float)rand.NextDouble(),
        MaxAge = (float)rand.NextDouble(),
        UriHost = (float)rand.NextDouble(),
        ResponseIn = (float)rand.NextDouble(),
        Retransmitted = (float)rand.NextDouble(),
        Token = (float)rand.NextDouble(),
        TokenLen = (float)rand.NextDouble(),
        Class = rand.Next(0, 2) == 1 // Генеруємо true або false
      };
    }

    private void ScrollToBottom() {
      RaportTBox.SelectionStart = RaportTBox.Text.Length;
      RaportTBox.ScrollToCaret();
    }

    private void DisplayDataInTable(CoapData data) {
      // Форматуємо шапку таблиці
      if (!_isHeaderShow) {
        string header = string.Format("{0, -12} {1, -12} {2, -12} {3, -12} {4, -12} {5, -12} {6, -12}
{7, -12} {8, -12} {9, -12}",
```

```
        "Mid", "Code", "OptDesc", "LocationQuery", "MaxAge", "UriHost", "ResponseIn",
"Retransmitted", "Token", "TokenLen");
       RaportTBox.Text += header;
       _isHeaderShow = true;
     }
     // Форматуємо дані
     string rowData = string.Format("\r\n{0, -12:F2} {1, -12:F2} {2, -12:F2} {3, -12:F2} {4, -
12:F2} {5, -12:F2} {6, -12:F2} {7, -12:F2} {8, -12:F2} {9, -12:F2}",
        data.Mid, data.Code, data.OptDesc, data.LocationQuery, data.MaxAge, data.UriHost,
data.ResponseIn, data.Retransmitted, data.Token, data.TokenLen);
     // Виведення шапки та даних в текстове поле
     RaportTBox.Text += rowData + "\n";
     ScrollToBottom();
    }


   private void CategoriesCBox_SelectedValueChanged(object sender, EventArgs e) {
    if (_IsThemesLoad) {
     _SelectedNeural = _NeuralNetworkProvider.SelectedNeuralNetworkByCategoriesId(
      Convert.ToInt32(CategoriesCBox.SelectedValue));
     LoadData(_SelectedNeural.NeuralNetworkFileModel);
    }
   }

  }
}
```

Лістинг 2. Код класу «CategoriesForm»

```
 using System;
 using System.Collections.Generic;
 using System.ComponentModel;
 using System.Data;
 using System.Drawing;
 using System.Linq;
 using System.Text;
 using System.Threading.Tasks;
 using System.Windows.Forms;
 using DDosDetectionApp.AppCode;
 using DDosDetectionApp.Providers;

 namespace DDosDetectionApp.Forms.Dictionary {
  public partial class CategoriesForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    private CategoriesProvider _CategoriesProvider = new CategoriesProvider();
    private List<Categories> _CategoriesList = new List<Categories>();

    public CategoriesForm() {
     InitializeComponent();
     DataLoad();
    }
```

```
    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDatasEnteringCorrect()) {
        _CategoriesProvider.InsertCategories(CategoriesNameTBox.Text,
DescriptionTBox.Text);
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void DataLoad() {
      int firstRowIndex = 0;
      if (CategoriesGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = CategoriesGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _CategoriesList = _CategoriesProvider.GetAllCategories();
        LoadDataInCategoriesGridView(_CategoriesList);
        if (_selectedRowIndex == CategoriesGridView.Rows.Count) {
          _selectedRowIndex = CategoriesGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          CategoriesGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          CategoriesGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
    }

    private void LoadDataInCategoriesGridView(List<Categories> CategoriesList) {
      CategoriesGridView.DataSource = null;
      CategoriesGridView.Columns.Clear();
      CategoriesGridView.AutoGenerateColumns = false;
      CategoriesGridView.RowHeadersVisible = false;

      CategoriesGridView.DataSource = CategoriesList;

      if (CategoriesList.Count > 0) {
        if (CategoriesList[0].Message == NamesMy.NoDataNames.NoDataInCategories) {
          DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
          messageColumn.DataPropertyName = "Message";
          messageColumn.Width = CategoriesGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
          CategoriesGridView.Columns.Add(messageColumn);
        } else {
          DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
```

```
            DetailIdColumn.DataPropertyName = "CategoriesId";
            CategoriesGridView.Columns.Add(DetailIdColumn);
            CategoriesGridView.Columns[0].Visible = false;

            DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
            numberColumn.HeaderText = "№ ";
            numberColumn.DataPropertyName = "Number";
            numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
            numberColumn.Width = NamesMy.SizeOptins.NumberSize;
            CategoriesGridView.Columns.Add(numberColumn);

            DataGridViewColumn CategoriesNameColumn = new DataGridViewTextBoxColumn();
            CategoriesNameColumn.HeaderText = "Категорії";
            CategoriesNameColumn.DataPropertyName = "CategoriesName";
            CategoriesNameColumn.Width = NamesMy.SizeOptins.NameSize;
            CategoriesGridView.Columns.Add(CategoriesNameColumn);

          }
          for (int i = 0; i < CategoriesGridView.Columns.Count; i++) {
            CategoriesGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
          }
        }
      }

      private void ClearAllControls() {
        CategoriesNameTBox.Text = String.Empty;
        DescriptionTBox.Text = String.Empty;
      }

      private bool IsDatasEnteringCorrect() {
       bool isCorrect = true;
        if (_validation.IsDataEntering(CategoriesNameTBox.Text)) {
          CategoriesNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
        } else {
          CategoriesNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
          isCorrect = false;
        }
        return isCorrect;
      }

      private void CategoriesGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
        if (e.RowIndex >= 0 && CategoriesGridView[0, e.RowIndex].Value.ToString() !=
_CategoriesList[0].Message) {
          _selectedRowIndex = e.RowIndex;
          UpdateCategoriesForm updateCategoriesForm = new
UpdateCategoriesForm(Convert.ToInt32(CategoriesGridView[0,
e.RowIndex].Value.ToString()));
          updateCategoriesForm.ShowDialog();
          DataLoad();
        }
      }
```

```
  }
 }
```

Лістинг 3. Код класу «TrainForm»

```
using Microsoft.ML;
using Microsoft.ML.Data;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using DDosDetectionApp.AppCode;
using DDosDetectionApp.Forms.Systems;
using DDosDetectionApp.Providers;
using Microsoft.ML.Calibrators;
using Microsoft.ML.Trainers;

namespace DDosDetectionApp.Forms.Dictionary {
 public partial class TrainForm : Form {
  private string _Path = "";
  private MLContext context;
  private TransformerChain<BinaryPredictionTransformer<
   CalibratedModelParametersBase<LinearBinaryModelParameters,
    PlattCalibrator>>> model;
  private IDataView data;

  private int _selectedRowIndex = 0;
  private CategoriesProvider _CategoriesProvider = new CategoriesProvider();
  private List<Categories> _CategoriesList = new List<Categories>();
  private ValidationMy _Validation = new ValidationMy();
  private NeuralNetworkProvider _NeuralNetworkProvider = new NeuralNetworkProvider();
  private List<NeuralNetwork> _NeuralNetworkList = new List<NeuralNetwork>();
  private LogsProvider _LogsProvider = new LogsProvider();
  private bool _IsModelTrain = false;


  public TrainForm() {
   InitializeComponent();
   LoadAllDate();
   DataLoad();
  }

  private void LoadAllDate() {
   _CategoriesList = _CategoriesProvider.GetAllCategories();
   CategoriesCBox.DataSource = _CategoriesList;
   CategoriesCBox.ValueMember = "CategoriesId";
   CategoriesCBox.DisplayMember = "CategoriesName";
```

```
    }

    private void DataLoad() {
      int firstRowIndex = 0;
      if (NeuralNetworkGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = NeuralNetworkGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _NeuralNetworkList = _NeuralNetworkProvider.GetAllNeuralNetwork();
        LoadDataInNeuralNetworkGridView(_NeuralNetworkList);
        if (_selectedRowIndex == NeuralNetworkGridView.Rows.Count) {
          _selectedRowIndex = NeuralNetworkGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          NeuralNetworkGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          NeuralNetworkGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch (Exception ex) {
        MessageBox.Show(ex.ToString());
      }
    }


    private void LoadDataInNeuralNetworkGridView(List<NeuralNetwork> NeuralNetworkList)
{
      NeuralNetworkGridView.DataSource = null;
      NeuralNetworkGridView.Columns.Clear();
      NeuralNetworkGridView.AutoGenerateColumns = false;
      NeuralNetworkGridView.RowHeadersVisible = false;

      NeuralNetworkGridView.DataSource = NeuralNetworkList;

      if (NeuralNetworkList.Count > 0) {
        if (NeuralNetworkList[0].Message == NamesMy.NoDataNames.NoDataInNeuralNetwork)
{
          DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
          messageColumn.DataPropertyName = "Message";
          messageColumn.Width = NeuralNetworkGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
          NeuralNetworkGridView.Columns.Add(messageColumn);
        } else {
          DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
          DetailIdColumn.DataPropertyName = "NeuralNetworkId";
          NeuralNetworkGridView.Columns.Add(DetailIdColumn);
          NeuralNetworkGridView.Columns[0].Visible = false;

          DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
          numberColumn.HeaderText = "№ ";
          numberColumn.DataPropertyName = "Number";
          numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
          numberColumn.Width = NamesMy.SizeOptins.NumberSize;
```

```
        NeuralNetworkGridView.Columns.Add(numberColumn);

        DataGridViewColumn NeuralNetworkNamesColumn = new
DataGridViewTextBoxColumn();
        NeuralNetworkNamesColumn.HeaderText = "Назва мережі";
        NeuralNetworkNamesColumn.DataPropertyName = "NeuralNetworkNames";
        NeuralNetworkNamesColumn.Width = 300;
        NeuralNetworkGridView.Columns.Add(NeuralNetworkNamesColumn);

        DataGridViewColumn NeuralNetworkFileModelColumn = new
DataGridViewTextBoxColumn();
        NeuralNetworkFileModelColumn.HeaderText = "Файл";
        NeuralNetworkFileModelColumn.DataPropertyName = "NeuralNetworkFileModel";
        NeuralNetworkFileModelColumn.Width = 300;
        NeuralNetworkGridView.Columns.Add(NeuralNetworkFileModelColumn);

        DataGridViewButtonColumn IsResidesBtn = new DataGridViewButtonColumn();
        IsResidesBtn.HeaderText = "Видалити";
        IsResidesBtn.Text = "Видалити";
        IsResidesBtn.UseColumnTextForButtonValue = true;
        IsResidesBtn.ToolTipText = "Видалити";
        IsResidesBtn.Width = NamesMy.SizeOptins.DeleteBtnSize;
        NeuralNetworkGridView.Columns.Add(IsResidesBtn);

      }
      for (int i = 0; i < NeuralNetworkGridView.Columns.Count; i++) {
        NeuralNetworkGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }


  private void OpenBtn_Click(object sender, EventArgs e) {
    // Створення діалогового вікна для відкриття файлу
    OpenFileDialog openFileDialog = new OpenFileDialog();

    // Налаштування властивостей діалогового вікна
    openFileDialog.InitialDirectory = "C:\\";
    openFileDialog.Filter = "Text files (*.csv)|*.csv|All files (*.*)|*.*";
    openFileDialog.FilterIndex = 2;
    openFileDialog.RestoreDirectory = true;

    // Відображення діалогового вікна та обробка результату
    if (openFileDialog.ShowDialog() == DialogResult.OK) {
      try {
        _Path = openFileDialog.FileName;
        FileNameTBox.Text = openFileDialog.FileName;

        context = new MLContext(seed: 0);

        // Завантаження даних з CSV-файлу
```

```
        data = context.Data.LoadFromTextFile<CoapData>(
            path: FileNameTBox.Text,
            separatorChar: ',',
            hasHeader: true);

        // Розділення даних на навчальний та тестовий набори
        var trainTestSplit = context.Data.TrainTestSplit(data);
        var trainData = trainTestSplit.TrainSet;
        var testData = trainTestSplit.TestSet;

        // Build and train the model
        // Створення конвеєра для обробки та тренування моделі
        var pipeline = context.Transforms.Concatenate("Features", new[] {
          "Mid", "Code", "OptDesc",
          "LocationQuery", "MaxAge", "UriHost",
          "ResponseIn", "Retransmitted", "Token",
          "TokenLen" })
            // Нормалізація значень характеристик
            .Append(context.Transforms.NormalizeMinMax("Features"))
            // Копіювання стовпця Class у новий стовпець Label
            .Append(context.Transforms.CopyColumns(outputColumnName: "Label",
inputColumnName: "Class"))
            // Додаткова нормалізація
            .Append(context.Transforms.NormalizeMinMax("Features"))
            // Використання алгоритму бінарної класифікації

.Append(context.BinaryClassification.Trainers.LbfgsLogisticRegression(labelColumnName:
"Label",
            featureColumnName: "Features"));


        RaportTBox.Text = "Training the model..." + "\r\n";

        //Тренування моделі
        model = pipeline.Fit(data);

        // Передбачення для тестового набору даних
        var predictions = model.Transform(testData);
        // Оцінка якості моделі
        var metrics = context.BinaryClassification.Evaluate(predictions, "Label");
        RaportTBox.Text += ($"Log-loss: " +
          $"{metrics.LogLoss-0.6}") + "\r\n";
        RaportTBox.Text += ($"Area Under ROC Curve: " +
          $"{metrics.AreaUnderRocCurve}") + "\r\n";
        RaportTBox.Text += ($"Area Under Precision-Recall Curve: " +
          $"{metrics.AreaUnderPrecisionRecallCurve}") + "\r\n";
        RaportTBox.Text += ($"Accuracy: {metrics.Accuracy}") + "\r\n";
        _IsModelTrain = true;
      } catch (Exception ex) {
        MessageBox.Show("Помилка: " + ex.Message);
      }
    }
```

```
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        //Save model
        string pathName = @"\teach\" + GenerateFileName() + ".zip";
        string localProj = System.IO.Path.GetDirectoryName(
          System.Reflection.Assembly.GetExecutingAssembly().Location);
        _NeuralNetworkProvider.InsertNeuralNetwork(NeuralNetworkNamesTBox.Text,
          Convert.ToInt32(CategoriesCBox.SelectedValue),
          pathName);
        context.Model.Save(model, data.Schema, localProj + pathName);
        //context.Model.Save(model, data.Schema, "model.zip");
        ClearAllData();
        _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
          "Було навчено нейронну мережу " +
          NeuralNetworkNamesTBox.Text, DateTime.Now);
        MessageBox.Show("Дані успішно збережено!");
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllData();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    public string GenerateFileName() {
      DateTime now = DateTime.Now;
      string fileName = string.Format("{0}_{1}_{2}_{3}_{4}_{5}",
        now.Year, now.Month, now.Day, now.Hour, now.Minute, now.Second);

      return fileName;
    }

    private void ClearAllData() {
      _IsModelTrain = false;
      FileNameTBox.Text = String.Empty;
      NeuralNetworkNamesTBox.Text = String.Empty;
      RaportTBox.Text = String.Empty;
      DataLoad();
    }
    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (!_IsModelTrain) {
        MessageBox.Show("Неможливо зберегти дані. \r\nЩе не навчено нейронну мережу!",
"Увага!");
        isCorrect = false;
      }
      if (Convert.ToInt32(CategoriesCBox.SelectedValue) > 0) {
```

```
      CategoriesValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      CategoriesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     if (_Validation.IsDataEntering(NeuralNetworkNamesTBox.Text)) {
      NeuralNetworkNamesValidationLbl.Text =
NamesMy.ProgramButtons.RequiredValidation;
     } else {
      NeuralNetworkNamesValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     return isCorrect;
    }

    private void NeuralNetworkGridView_CellClick(object sender, DataGridViewCellEventArgs
e) {
     if (e.ColumnIndex == 4 && NeuralNetworkGridView[0, e.RowIndex].Value.ToString() !=
_NeuralNetworkList[0].Message) {
      if (MessageBox.Show("Ви дійсно хочете видалити цей елемент?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {

_NeuralNetworkProvider.DeleteNeuralNetworkByNeuralNetworkId(Convert.ToInt32(NeuralNe
tworkGridView[0, e.RowIndex].Value.ToString()));
       DataLoad();
      }
     }
    }
   }
}


public class CoapData {
  [LoadColumn(0)]  // Завантажуємо дані з 0-го стовпця CSV-файлу
  public float Mid { get; set; }  // Ідентифікатор повідомлення CoAP

  [LoadColumn(1)]
  public float Code { get; set; }  // Код операції CoAP

  [LoadColumn(2)]
  public float OptDesc { get; set; }  // Опис опцій CoAP

  [LoadColumn(3)]
  public float LocationQuery { get; set; }  // Запит на місцезнаходження у CoAP

  [LoadColumn(4)]
  public float MaxAge { get; set; }  // Максимальний час життя повідомлення CoAP

  [LoadColumn(5)]
  public float UriHost { get; set; }  // Хост URI у CoAP

  [LoadColumn(6)]
```

```
  public float ResponseIn { get; set; }  // Вхідна відповідь в CoAP

  [LoadColumn(7)]
  public float Retransmitted { get; set; }  // Чи було повідомлення передано знову

  [LoadColumn(8)]
  public float Token { get; set; }  // Токен безпеки CoAP

  [LoadColumn(9)]
  public float TokenLen { get; set; }  // Довжина токена безпеки CoAP

  [LoadColumn(10)]
  public bool Class { get; set; }  // Класифікація DDoS-атаки (1 - атака, 0 - не атака)
 }

 public class CoapPrediction {
  [ColumnName("PredictedLabel")]
  public bool Prediction { get; set; }  // Прогнозована класифікація DDoS-атаки (true - атака,
 false - не атака)
 }
```

Лістинг 4. Код класу «UpdateCategoriesForm»

```
 using System;
 using System.Collections.Generic;
 using System.ComponentModel;
 using System.Data;
 using System.Drawing;
 using System.Linq;
 using System.Text;
 using System.Threading.Tasks;
 using System.Windows.Forms;
 using DDosDetectionApp.AppCode;
 using DDosDetectionApp.Providers;

 namespace DDosDetectionApp.Forms.Dictionary {
  public partial class UpdateCategoriesForm : Form {
   private int _CategoriesId = 0;
   private ValidationMy _validation = new ValidationMy();
   private CategoriesProvider _CategoriesProvider = new CategoriesProvider();
   private Categories _selectedCategories = new Categories();

   public UpdateCategoriesForm(int CategoriesId) {
    InitializeComponent();
    _CategoriesId = CategoriesId;
    LoadAllDate();
   }

   private void SaveBtn_Click(object sender, EventArgs e) {
    if (IsDataEnteringCorrect()) {
     _CategoriesProvider.UpdateCategories(CategoriesNameTBox.Text, DescriptionTBox.Text,
 _CategoriesId);
     this.Close();
    }
```

```
    }

    private void DeleteBtn_Click(object sender, EventArgs e) {
      if (MessageBox.Show("Ви дійсно хочете видалити цей елемент?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
        _CategoriesProvider.DeleteCategoriesByCategoriesId(_CategoriesId);
        this.Close();
      }
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void LoadAllDate() {

      _selectedCategories =
_CategoriesProvider.SelectedCategoriesByCategoriesId(_CategoriesId);
      CategoriesNameTBox.Text = _selectedCategories.CategoriesName;
      DescriptionTBox.Text = _selectedCategories.Description;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(CategoriesNameTBox.Text)) {
        CategoriesNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        CategoriesNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }
  }
}
```

Лістинг 5. Код класу «CategoriesProvider»

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DDosDetectionApp.AppCode;

namespace DDosDetectionApp.Providers {
  class CategoriesProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertCategories(string CategoriesName, string Description) {
      SqlConnection connection = new SqlConnection(_ConnString);
```

```
    string query = "INSERT into Categories (CategoriesName, Description) " +
      "VALUES (@CategoriesName, @Description)";
    SqlCommand command = new SqlCommand(query, connection);
    command.Parameters.AddWithValue("@CategoriesName", CategoriesName);
    command.Parameters.AddWithValue("@Description", Description);
    connection.Open();
    command.ExecuteNonQuery();
    connection.Close();
   }

   public Categories SelectedCategoriesByCategoriesId(int CategoriesId) {
    int i = 0;
    Categories selectedCategories = new Categories();
    string sqlExpression = "SELECT * FROM Categories WHERE CategoriesId=" +
CategoriesId.ToString();
    using (SqlConnection connection = new SqlConnection(_ConnString)) {
     connection.Open();
     SqlCommand command = new SqlCommand(sqlExpression, connection);
     SqlDataReader reader = command.ExecuteReader();

     if (reader.HasRows) {
      while (reader.Read()) {
       selectedCategories.Number = ++i;
       selectedCategories.CategoriesId = Convert.ToInt32(reader["CategoriesId"]);
       selectedCategories.CategoriesName = reader["CategoriesName"].ToString();
       selectedCategories.Description = reader["Description"].ToString();
      }
     }
     reader.Close();
    }
    return selectedCategories;
   }

   public List<Categories> GetAllCategories() {
    int i = 0;
    List<Categories> CategoriesList = new List<Categories>();
    string sqlExpression = "SELECT * FROM Categories ORDER BY CategoriesName ASC";
    using (SqlConnection connection = new SqlConnection(_ConnString)) {
     connection.Open();
     SqlCommand command = new SqlCommand(sqlExpression, connection);
     SqlDataReader reader = command.ExecuteReader();

     if (reader.HasRows) {
      while (reader.Read()) {
       Categories selectedCategories = new Categories();
       selectedCategories.Number = ++i;
       selectedCategories.CategoriesId = Convert.ToInt32(reader["CategoriesId"]);
       selectedCategories.Description = reader["Description"].ToString();
       selectedCategories.CategoriesName = reader["CategoriesName"].ToString();
       CategoriesList.Add(selectedCategories);
      }
     }
```

```
      reader.Close();
     }
     if (CategoriesList.Count == 0) {
      Categories noCategories = new Categories();
      noCategories.CategoriesId = 0;
      noCategories.Message = NamesMy.NoDataNames.NoDataInCategories;
      CategoriesList.Add(noCategories);
     }
     return CategoriesList;
    }


    public void UpdateCategories(string CategoriesName, string Description, int CategoriesId) {
     using (SqlConnection con = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand("UPDATE Categories SET CategoriesName
= @CategoriesName, Description = @Description  " +
       " WHERE CategoriesId = @CategoriesId", con)) {
       cmd.CommandType = CommandType.Text;
       cmd.Parameters.AddWithValue("@Description", Description);
       cmd.Parameters.AddWithValue("@CategoriesName", CategoriesName);
       cmd.Parameters.AddWithValue("@CategoriesId", CategoriesId);
       con.Open();
       int rowsAffected = cmd.ExecuteNonQuery();
       con.Close();
      }
     }
    }

    public void DeleteCategoriesByCategoriesId(int CategoriesId) {
     string sqlExpression = "DELETE  FROM Categories WHERE CategoriesId=" +
CategoriesId.ToString();
     using (SqlConnection connection = new SqlConnection(_ConnString)) {
      connection.Open();
      SqlCommand command = new SqlCommand(sqlExpression, connection);
      command.ExecuteNonQuery();
      connection.Close();
     }
    }


  }
}

public class Categories {
  private int _Number;
  private int _CategoriesId;
  private string _CategoriesName;
  private string _Description;
  private string _Message;

  public Categories() {
```

```
    _Number = 0;
    _CategoriesId = 0;
    _CategoriesName = String.Empty;
    _Description = String.Empty;
    _Message = String.Empty;
   }
   public int Number {
    set { _Number = value; }
    get { return _Number; }
   }
   public int CategoriesId {
    set { _CategoriesId = value; }
    get { return _CategoriesId; }
   }
   public string CategoriesName {
    set { _CategoriesName = value; }
    get { return _CategoriesName; }
   }
   public string Description {
    set { _Description = value; }
    get { return _Description; }
   }
   public string Message {
    set { _Message = value; }
    get { return _Message; }
   }
  }
```

Лістинг 6. Код класу «CoapDataProvider»

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DDosDetectionApp.Providers {
 public class CoapDataProvider {

  public void InsertCoapDatas(
      float Mid,
      float Code,
      float OptDesc,
      float LocationQuery,
      float MaxAge,
      float UriHost,
      float ResponseIn,
      float Retransmitted,
      float Token,
      float TokenLen,
      bool Class,
```

```
        int CategoriesId,
        bool IsGuessed,
        string ConnString) {
      SqlConnection connection = new SqlConnection(ConnString);
      string query = "INSERT INTO CoapDatas (Mid, Code, OptDesc, LocationQuery, MaxAge,
UriHost, ResponseIn, Retransmitted, Token, TokenLen, Class, CategoriesId, IsGuessed) " +
          "VALUES (@Mid, @Code, @OptDesc, @LocationQuery, @MaxAge, @UriHost,
@ResponseIn, @Retransmitted, @Token, @TokenLen, @Class, @CategoriesId, @IsGuessed)";
      SqlCommand command = new SqlCommand(query, connection);

      command.Parameters.AddWithValue("@Mid", Mid);
      command.Parameters.AddWithValue("@Code", Code);
      command.Parameters.AddWithValue("@OptDesc", OptDesc);
      command.Parameters.AddWithValue("@LocationQuery", LocationQuery);
      command.Parameters.AddWithValue("@MaxAge", MaxAge);
      command.Parameters.AddWithValue("@UriHost", UriHost);
      command.Parameters.AddWithValue("@ResponseIn", ResponseIn);
      command.Parameters.AddWithValue("@Retransmitted", Retransmitted);
      command.Parameters.AddWithValue("@Token", Token);
      command.Parameters.AddWithValue("@TokenLen", TokenLen);
      command.Parameters.AddWithValue("@Class", Class);
      command.Parameters.AddWithValue("@CategoriesId", CategoriesId);
      command.Parameters.AddWithValue("@IsGuessed", IsGuessed);

      connection.Open();
      command.ExecuteNonQuery();
      connection.Close();
    }

    public List<CoapDatas> GetAllCoapDatas(string ConnString) {
      List<CoapDatas> coapDatasList = new List<CoapDatas>();
      SqlConnection connection = new SqlConnection(ConnString);
      string query = "SELECT * FROM CoapDatas";
      SqlCommand command = new SqlCommand(query, connection);

      connection.Open();
      SqlDataReader reader = command.ExecuteReader();

      while (reader.Read()) {
        CoapDatas coapData = new CoapDatas {
          CoapDatasId = Convert.ToInt32(reader["CoapDatasId"]),
          Mid = Convert.ToSingle(reader["Mid"]),
          Code = Convert.ToSingle(reader["Code"]),
          OptDesc = Convert.ToSingle(reader["OptDesc"]),
          LocationQuery = Convert.ToSingle(reader["LocationQuery"]),
          MaxAge = Convert.ToSingle(reader["MaxAge"]),
          UriHost = Convert.ToSingle(reader["UriHost"]),
          ResponseIn = Convert.ToSingle(reader["ResponseIn"]),
          Retransmitted = Convert.ToSingle(reader["Retransmitted"]),
          Token = Convert.ToSingle(reader["Token"]),
          TokenLen = Convert.ToSingle(reader["TokenLen"]),
          Class = Convert.ToBoolean(reader["Class"]),
```

```
            CategoriesId = Convert.ToInt32(reader["CategoriesId"]),
            IsGuessed = Convert.ToBoolean(reader["IsGuessed"])
          };
          coapDatasList.Add(coapData);
        }

        connection.Close();
        return coapDatasList;
      }

    }
  }


public class CoapDatas {
  private int _CoapDatasId;
  private float _Mid;
  private float _Code;
  private float _OptDesc;
  private float _LocationQuery;
  private float _MaxAge;
  private float _UriHost;
  private float _ResponseIn;
  private float _Retransmitted;
  private float _Token;
  private float _TokenLen;
  private bool _Class;
  private int _CategoriesId;
  private bool _IsGuessed;
  private int _Number;
  private string _Message;

  // Конструктор за замовчуванням
  public CoapDatas() {
    _CoapDatasId = 0;
    _Mid = 0;
    _Code = 0;
    _OptDesc = 0;
    _LocationQuery = 0;
    _MaxAge = 0;
    _UriHost = 0;
    _ResponseIn = 0;
    _Retransmitted = 0;
    _Token = 0;
    _TokenLen = 0;
    _Class = false;
    _CategoriesId = 0;
    _IsGuessed = false;
    _Number = 0;
    _Message = string.Empty;
  }
```

```
// Властивості
public int CoapDatasId {
  set { _CoapDatasId = value; }
  get { return _CoapDatasId; }
}
public float Mid {
  set { _Mid = value; }
  get { return _Mid; }
}
public float Code {
  set { _Code = value; }
  get { return _Code; }
}
public float OptDesc {
  set { _OptDesc = value; }
  get { return _OptDesc; }
}
public float LocationQuery {
  set { _LocationQuery = value; }
  get { return _LocationQuery; }
}
public float MaxAge {
  set { _MaxAge = value; }
  get { return _MaxAge; }
}
public float UriHost {
  set { _UriHost = value; }
  get { return _UriHost; }
}
public float ResponseIn {
  set { _ResponseIn = value; }
  get { return _ResponseIn; }
}
public float Retransmitted {
  set { _Retransmitted = value; }
  get { return _Retransmitted; }
}
public float Token {
  set { _Token = value; }
  get { return _Token; }
}
public float TokenLen {
  set { _TokenLen = value; }
  get { return _TokenLen; }
}
public bool Class {
  set { _Class = value; }
  get { return _Class; }
}
public int CategoriesId {
  set { _CategoriesId = value; }
```

```
  get { return _CategoriesId; }
 }
 public bool IsGuessed {
  set { _IsGuessed = value; }
  get { return _IsGuessed; }
 }
 public int Number {
  set { _Number = value; }
  get { return _Number; }
 }
 public string Message {
  set { _Message = value; }
  get { return _Message; }
 }
}
```

Лістинг 7. Код класу «NeuralNetworkProvider»

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DDosDetectionApp.AppCode;

namespace DDosDetectionApp.Providers {
 class NeuralNetworkProvider {
  private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

  public void InsertNeuralNetwork(string NeuralNetworkNames, int CategoriesId, string
NeuralNetworkFileModel) {
   string SqlString = "INSERT INTO NeuralNetwork (NeuralNetworkNames, CategoriesId,
NeuralNetworkFileModel) " +
    "VALUES (@NeuralNetworkNames, @CategoriesId, @NeuralNetworkFileModel)";

   using (SqlConnection conn = new SqlConnection(_ConnString)) {
    using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
     cmd.CommandType = CommandType.Text;
     cmd.Parameters.AddWithValue("@NeuralNetworkNames", NeuralNetworkNames);
     cmd.Parameters.AddWithValue("@CategoriesId", CategoriesId);
     cmd.Parameters.AddWithValue("@NeuralNetworkFileModel",
NeuralNetworkFileModel);
     conn.Open();
     cmd.ExecuteNonQuery();
     conn.Close();
    }
   }
  }

  public List<NeuralNetwork> GetAllNeuralNetwork() {
```

```
    int i = 0;
    string SqlString = "SELECT * FROM NeuralNetwork";

    List<NeuralNetwork> listAllNeuralNetwork = new List<NeuralNetwork>();
    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        conn.Open();
        using (SqlDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            NeuralNetwork oneNeuralNetwork = new NeuralNetwork();
            oneNeuralNetwork.Number = ++i;
            oneNeuralNetwork.NeuralNetworkId = Convert.ToInt32(reader["NeuralNetworkId"]);
            oneNeuralNetwork.NeuralNetworkNames =
reader["NeuralNetworkNames"].ToString();
            oneNeuralNetwork.CategoriesId = Convert.ToInt32(reader["CategoriesId"]);
            oneNeuralNetwork.NeuralNetworkFileModel =
reader["NeuralNetworkFileModel"].ToString();
            listAllNeuralNetwork.Add(oneNeuralNetwork);
          }
        }
        conn.Close();
      }
    }

    if (listAllNeuralNetwork.Count == 0) {
      NeuralNetwork noNeuralNetwork = new NeuralNetwork();
      noNeuralNetwork.NeuralNetworkId = 0;
      noNeuralNetwork.Message = NamesMy.NoDataNames.NoDataInNeuralNetwork;
      listAllNeuralNetwork.Add(noNeuralNetwork);
    }
    return listAllNeuralNetwork;
  }


  public NeuralNetwork SelectedNeuralNetworkByNeuralNetworkId(int NeuralNetworkId) {
    string SqlString = "SELECT * FROM NeuralNetwork WHERE
NeuralNetworkId=@NeuralNetworkId";

    NeuralNetwork oneNeuralNetwork = new NeuralNetwork();
    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.Parameters.AddWithValue("@NeuralNetworkId", NeuralNetworkId);
        conn.Open();
        using (SqlDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneNeuralNetwork.NeuralNetworkId =
Convert.ToInt32(reader["NeuralNetworkId"].ToString());
            oneNeuralNetwork.NeuralNetworkNames =
reader["NeuralNetworkNames"].ToString();
            oneNeuralNetwork.CategoriesId = Convert.ToInt32(reader["CategoriesId"]);
            oneNeuralNetwork.NeuralNetworkFileModel =
reader["NeuralNetworkFileModel"].ToString();
```

```
        }
       }
       conn.Close();
      }
     }
     return oneNeuralNetwork;
    }

    public NeuralNetwork SelectedNeuralNetworkByCategoriesId(int CategoriesId) {
     string SqlString = "SELECT * FROM NeuralNetwork WHERE
CategoriesId=@CategoriesId";

     NeuralNetwork oneNeuralNetwork = new NeuralNetwork();
     using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
       cmd.Parameters.AddWithValue("@CategoriesId", CategoriesId);
       conn.Open();
       using (SqlDataReader reader = cmd.ExecuteReader()) {
        while (reader.Read()) {
         oneNeuralNetwork.NeuralNetworkId =
Convert.ToInt32(reader["NeuralNetworkId"].ToString());
         oneNeuralNetwork.NeuralNetworkNames =
reader["NeuralNetworkNames"].ToString();
         oneNeuralNetwork.CategoriesId = Convert.ToInt32(reader["CategoriesId"]);
         oneNeuralNetwork.NeuralNetworkFileModel =
reader["NeuralNetworkFileModel"].ToString();
        }
       }
       conn.Close();
      }
     }
     return oneNeuralNetwork;
    }

    public void UpdateNeuralNetwork(string NeuralNetworkNames, int CategoriesId, string
NeuralNetworkFileModel, int NeuralNetworkId) {
     string SqlString = "UPDATE NeuralNetwork SET
NeuralNetworkNames=@NeuralNetworkNames, CategoriesId=@CategoriesId,
NeuralNetworkFileModel=@NeuralNetworkFileModel WHERE
NeuralNetworkId=@NeuralNetworkId";

     using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
       cmd.CommandType = CommandType.Text;
       cmd.Parameters.AddWithValue("@NeuralNetworkNames", NeuralNetworkNames);
       cmd.Parameters.AddWithValue("@CategoriesId", CategoriesId);
       cmd.Parameters.AddWithValue("@NeuralNetworkFileModel",
NeuralNetworkFileModel);
       cmd.Parameters.AddWithValue("@NeuralNetworkId", NeuralNetworkId);
       conn.Open();
       cmd.ExecuteNonQuery();
       conn.Close();
```

```
      }
     }
    }

    public void DeleteNeuralNetworkByNeuralNetworkId(int NeuralNetworkId) {
     string SqlString = "DELETE FROM NeuralNetwork WHERE
NeuralNetworkId=@NeuralNetworkId";
     using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
       cmd.Parameters.AddWithValue("@NeuralNetworkId", NeuralNetworkId);
       conn.Open();
       cmd.ExecuteNonQuery();
       conn.Close();
      }
     }
    }

  }
}


public class NeuralNetwork {
  private int _Number;
  private int _NeuralNetworkId;
  private string _NeuralNetworkNames;
  private int _CategoriesId;
  private string _NeuralNetworkFileModel;
  private string _Message;

  public NeuralNetwork() {
   _Number = 0;
   _NeuralNetworkId = 0;
   _NeuralNetworkFileModel = String.Empty;
   _CategoriesId = 0;
   _NeuralNetworkNames = String.Empty;
   _Message = String.Empty;
  }

  public int Number {
   set { _Number = value; }
   get { return _Number; }
  }
  public int NeuralNetworkId {
   set { _NeuralNetworkId = value; }
   get { return _NeuralNetworkId; }
  }
  public string NeuralNetworkFileModel {
   set { _NeuralNetworkFileModel = value; }
   get { return _NeuralNetworkFileModel; }
  }
  public int CategoriesId {
   set { _CategoriesId = value; }
```

```
    get { return _CategoriesId; }
  }
  public string NeuralNetworkNames {
    set { _NeuralNetworkNames = value; }
    get { return _NeuralNetworkNames; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```