

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Вычислительные машины, комплексы, системы и сети»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

ПРОГРАММНЫЙ АНАЛИЗАТОР

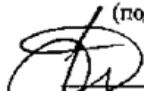
СЕТЕВОГО ТРАФИКА

Студент

 22.05.2026
(подпись, дата)

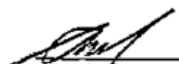
Тихонов Андрей Алексеевич
(ФИО полностью)

Руководитель

 26.05.2026
(подпись, дата)

Р.Н. Федюнин
(фамилия, инициалы)

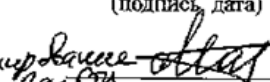
Нормоконтролёр

 27.05.2026
(подпись, дата)

А.С. Бычков
(фамилия, инициалы)

Рецензент

зв. кафе Программирования
(должность, место работы)

 27.05.2026
(подпись, дата)

А.И. Мартышкин
(фамилия, инициалы)

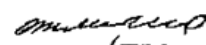
Работа допущена к защите (протокол зас. кафедры от 22.05.2026 № 14)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой



(протокол заседания ГЭК от 09.06.2026 № 21)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
« 20 » 09 2024 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Тихонов Андрей Алексеевич гр. 24ВВИм1 факультета ВТ

2. Тема работы Программный анализатор сетевого трафика

Тема утверждена приказом ПГУ № 920 от " 20 " Сентября 20 24 г.

3. Руководитель магистерской диссертации к.т.н., доцент кафедры ВТ ПГУ Федонин Р.Н.

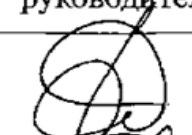
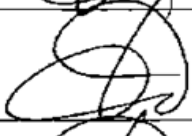
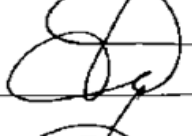
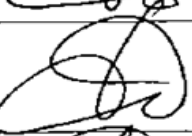
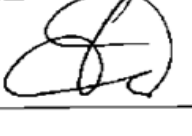
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Выполнить исследование и разработать оптимальный способ анализа потокового сетевого трафика при помощи нейронных сетей без потери в скорости передачи данных. Создать программный комплекс анализа сетевого трафика при помощи нейронной сети, получаемого напрямую из сетевого стека и разработать для него оптимальные алгоритмы передачи данных. Программный комплекс должен состоять из трёх компонентов: драйвера NDIS перехватывающего сетевой трафик, графического интерфейса и нейронной сети. Нейронная сеть должна анализировать приходящий к ней в потоковом режиме трафик для обнаружения сетевых атак.

5. Перечень подлежащих разработке вопросов

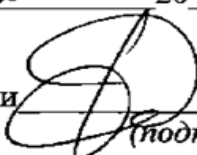
Требуется произвести анализ разработанности предметной области. Провести анализ и исследование эффективного способа передачи данных с уровня ядра в нейронную сеть. Провести анализ архитектур нейронных сетей для анализа сетевого трафика, провести поиск и подготовку подходящего набора данных для обучения нейронной сети, а также провести тестирование эффективности всей разработанной системы.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение теоретических основ	20	20.09.2024-15.11.2024	
2.	Анализ существующих способов передачи данных	20	16.11.2024-22.03.2025	
3.	Анализ существующих нейронных сетей	40	23.03.2025-14.06.2025	
4.	Выбор датасета и выявление значимых признаков	20	15.06.2025-10.09.2025	
5.	Реализация и тестирование нейронной сети	40	11.09.2025-05.01.2026	
6.	Реализация механизма перехвата и передачи данных	40	06.01.2026-17.04.2026	
7.	Реализация и тестирование компонентов программного комплекса	70	18.04.2026-07.05.2026	
8.	Оформление выпускной работы	20	08.05.2026-21.05.2026	

Дата выдачи задания " 20 " сентября 20 24 г.

Руководитель магистерской диссертации



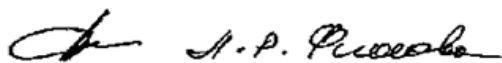
(подпись, дата)

Задание к исполнению принял студент



(подпись, дата)

Работу к защите допустить
Декан факультета



(подпись, дата)

от 06.2026

РЕФЕРАТ

Магистерская работа содержит 72 листа, 31 рисунок, 1 таблицу, 17 источников и 2 приложения.

РЕАЛИЗАЦИЯ ЗАЩИТЫ СЕТИ ПРИ ПОМОЩИ НЕЙРОННЫХ СЕТЕЙ, ПЕРЕХВАТ СЕТЕВОГО ТРАФИКА НА УРОВНЕ ЯДРА, МНОГОСЛОЙНЫЙ ПЕРЦЕПТРОН, СЕТЕВЫЕ АТАКИ, АЛГОРИТМЫ ПЕРЕДАЧИ ДАННЫХ

Цель работы – повышение защищённости конечных ЭВМ от сетевых атак при помощи использования технологий нейронных сетей и перехвата сетевого трафика с уровня ядра.

Результатом работы будет является программный комплекс, состоящий из трёх компонентов: драйвера NDIS выполняющий задачу перехвата сетевого трафика, который передаётся на пользовательский уровень для анализа, графический интерфейс, выполняющий функцию пользовательского интерфейса, а также связующим звеном передачи данных от драйвера и нейронной сети. Компонент нейронной сети является многоклассовым для определения типа сетевой атаки, а также должен выполнять анализ сетевого трафика в формате сетевых потоков.

Помимо этого, важной частью работы является оптимизация и поиск оптимальных алгоритмов передачи и работы с данными в разрабатываемой системы, для минимизации потерь скорости работы сети.

					ПГУ 1.090401.24.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Программный анализатор сетевого трафика Пояснительная записка	Лит.	Лист	Листов
Разраб.		Тихонов А.А.		14.05.26				
Провер.		Федюнин Р.Н.		14.05.26			5	109
Н. Контр.		Бычков А. С.		14.05.26				
Утверд.		Митрохин М.А.		14.05.26				
						ФВТ, каф. ВТ гр. 24ВВИМ1		

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 Анализ темы	7
1.1 Анализ степени разработанности предметной области.....	7
1.2 Постановка задачи	9
2 Построение архитектуры и поиск алгоритма передачи данных	11
2.1 Архитектура разрабатываемой системы.....	11
2.2 Поиск эффективного способа обработки сетевого трафика.....	16
2.3 Формализованное описание модели обработки трафика	20
3 Разработка нейронной сети	25
3.1 Описание разработанной нейронной сети	39
3.2 Выбор датасета.....	40
3.3 Подготовка обучающих данных	42
3.4 Архитектура и обучение нейронной сети	44
3.5 Результаты обучения нейронной сети	48
4 Установка и тестирование программного комплекса	53
4.1 Установка драйвера	53
4.2 Запуск компонентов 3 кольца.....	58
4.3 Тестирование программного комплекса	60
4.4 Тестирование пропускной способности.....	62
5 Возможности для улучшения и расширения функционала	65
ЗАКЛЮЧЕНИЕ	68
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	71
ПРИЛОЖЕНИЕ А – Листинг программы	73
ПРИЛОЖЕНИЕ Б – Презентация.....	102

ВВЕДЕНИЕ

Современные компьютерные сети являются критически важной инфраструктурой для бизнеса, государственных организаций и частных пользователей. С развитием облачных технологий, интернета вещей (IoT) и повсеместного внедрения цифровых сервисов объем передаваемых данных растет экспоненциально. В таких условиях анализ сетевого трафика становится не просто полезным инструментом, а необходимостью для обеспечения безопасности, отказоустойчивости и эффективности работы сетевых систем.

Механизмы обработки сетевого трафика могут включать в себя выполнение обширного количества всевозможных операций и решения различных задач над приходящим и исходящим сетевым трафиком таких как: модификация, шифрование, сортировка для последующей обработки, выборка, сбор статистики, формирование правил для защиты информации и т.д. На основе данных операций возможно создать огромное количество программного обеспечения, которое будет выполнять задачи по защите и шифрованию информации, журналирование сетевого трафика для последующего его анализа и обработки, построения собственных новых сетевых протоколов, выявлении и блокировании подозрительного сетевого трафика или его фильтрации на основе установленных правил.

1 Анализ темы

В условиях постоянного роста использования цифровых технологий, перехода на повсеместное использование электронно-вычислительных устройств (ЭВУ) для управления, сохранения, передачи и использования информации, растёт и необходимость защиты данных устройств от различных атак извне, особенно из сети. Учитывая также что количество конфиденциальной информации в различных цифровых хранилищах растёт, а также и растёт и использование устройств автоматизированного управления в транспортных, банковских и корпоративных и других областях проблема защиты сетей, в которых расположены эти устройства становится всё более и более острой. Если 10-15 лет назад тяжело было представить, что при помощи сетевой атаки можно остановить транспортную систему или коммунальную систему города или района практически невозможно, то сейчас из-за развития и интеграции ЭВУ во все сферы жизни такая опасность становится вполне реальной. Например, в 2021 году хакеры в округе Пинеллас, штат Флорида пытались взломать очистные сооружения чтобы отравить воду, повысив уровень гидроксида натрия в воде, опасность была устранена путём ручного вмешательства. Данный инцидент лишь один из многих который показывает от безопасности цифровой инфраструктуры, может повесить не только сохранность конфиденциальной информации, но и даже человеческие жизни [1].

Учитывая всё выше сказанное можно сказать, что при неостанавливающемся росте цифровизации проблема безопасности ЭВУ будет актуальной ещё очень долго время. Также для анализа сетевого трафика можно использовать быстро развивающуюся технологию нейронных сетей, для увеличения безопасности и актуализации данных о возможных направлениях и способах сетевых атак.

1.1 Анализ степени разработанности предметной области

Анализируя предметную область, аналоги уже созданного программного обеспечения и уже написанные работы на схожую тему, так же

важно учитывать одну отличительную особенность разрабатываемого программного обеспечения в выбранной теме, а именно интеграция в работу анализатора сетевого трафика нейронной сети. Аналогов подобных анализаторов сетевого трафика на рынке, как минимум в открытом доступе не существует. Одними из ближайших аналогов программ, которые перехватывают сетевой трафик и предоставляют его в более понятном для пользователя виде можно назвать Wireshark или Network Monitor, однако они никак не используют в своей работе нейронные сети, а также не предоставляют функционала защиты от сетевых атак. Научные работы на тему использования нейронных сетей для анализа сетевого трафика уже не настолько редки, однако все найденные исследования не предполагают использования нейронных сетей для анализа сетевого трафика в режиме реального времени, а используют в качестве входных данных файлы или статичные структуры и базы данных, что ещё раз подчёркивает уникальность и актуальность данной работы. Далее представлены несколько примеров научных работ по схожей тематике:

- статья в журнале «Международный научно-исследовательский журнал» за авторством Денисенко А.А. и Губарь А.Ю. на тему «Анализ сетевого трафика с использованием нейронных сетей» [2],

- статья в журнале «Труды СПИИРАН» за авторством Ахмедова А.Э. и Кузнецова А.А. на тему «Применение глубокого обучения для классификации сетевого» [3],

- статья в журнале «Кибербезопасность и защита информации» за авторством Кузнецова С.Д. и Иванова И.А. на тему «Анализ эффективности сверточных нейронных сетей для детектирования атак в сетевом трафике» [4].

Также, так как сетевые атаки никогда не стоят на месте, и злоумышленники разрабатывают новые способы взлома, что предполагает постоянное обновление защитных баз, с информацией о новых способах сетевых атак. Обычные программные средства защиты, такие как антивирусы, для поддержания актуальности своих защитных средств обычно могут

обновлять базы, с которыми ПО сравнивает вредоносные приложения или обновлять сами методы и алгоритмы обнаружения вредоносного ПО. Однако в случае, когда для анализа обнаружения использует нейронная сеть вопрос уже стоит не только в обновлении набора данных, но и постоянном дообучении готовой модели. На тему эффективного метода дообучения нейронной сети также была найдена научная работа Мисаилиди А. А. на тему «Автоматизация процесса дообучения нейронной сети и его перенос на сторону клиента» [5], которая помогает решить вопрос актуализации защитных механизмов при использовании нейронной сети.

1.2 Постановка задачи

Целью данной научной работы является повышение устойчивости конечных ЭВМ к различным сетевым атакам, путём перехвата сетевого трафика из ядра системы в режиме реального времени и его анализа при помощи нейронных сетей.

Объектом исследования данной научной работы является система перехвата и анализа сетевого трафика в режиме реального времени.

Предметом исследований данной научной работы является непосредственно нейронная сеть способная обнаруживать вредоносные сетевые соединения.

Задачами проводимого исследования является:

- разработка эффективного способа/алгоритма перехвата сетевого трафика в режиме реального времени для последующей его обработки нейронными сетями без потери в производительности и скорости соединения,
- разработка компонента NDIS-драйвера фильтра, осуществляющего перехват сетевого трафика,
- разработка компонента графического интерфейса сетевого анализатора, выполняющего также роль связующего звена между драйвером и нейронной сетью,
- поиск и подготовка данных сетевого трафика для обучения нейронной сети,

- разработка компонента нейронной сети, которая занимается обнаружением вредоносного сетевого трафика,
- тестирование разработанной системы и проверка эффективности скорости обработки сетевого трафика, погрешности обнаружения вредоносного трафика.

2 Построение архитектуры и поиск алгоритма передачи данных

В данной магистерской диссертации проводятся исследования эффективного способа передачи данных и разработка нейронной сети, способной анализировать сетевой трафик и обнаруживать в них сетевые атаки. При разработке такой сложной системы обойтись одним компонентом, при этом сохранив все функциональные возможности, предоставляемые на разных уровнях операционной системы и обеспечить высокую скорость обработки данных практически невозможно.

2.1 Архитектура разрабатываемой системы

Вследствие особенностей требований к скорости передачи данных при написании диссертации была разработана система, состоящая из 3 компонентов, каждый из которых выполняет свои отдельные функции и участвует в задачах по перехвату, обработке и анализу сетевого трафика. Далее на рисунке 1 представленная базовая модель связи программного обеспечения, которое будет использоваться для поставленной задачи.

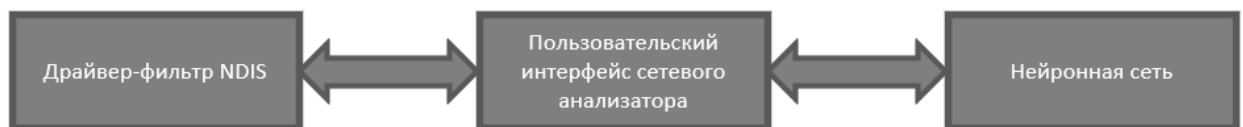


Рисунок 1 – Модель взаимодействия программного обеспечения построенной системы

Как можно увидеть, построенная система состоит из трёх взаимосвязанных программных компонентов:

- драйвер-фильтр NDIS: данная часть системы выполняет перехват сетевого трафика и передачу его с уровня ядра на пользовательский уровень для дальнейшей обработки,

- пользовательский интерфейс сетевого анализатора: программное обеспечение на пользовательском уровне системы, которое используется как графический интерфейс для отображения собранной статистики и результатов обработки сетевого трафика нейронной сетью. Так же используется как

связующее звено между драйвером-фильтром, перехватывающим пакеты на уровне ядра и нейронной сетью. Приложение графического интерфейса также выполняет задачу по разбору и конвертации полученного от драйвера сетевого трафика в формат, необходимый для работы нейронной сети, который будет подробнее разобран позднее,

- нейронная сеть: в данной системе нейронная сеть может использоваться для решения задач таких как: обнаружение сетевых атак на основе данных, полученных из компонента графического интерфейса, составление правил для последующего блокирования подозрительных сетевых соединений на уровне ядра в драйвере-фильтре.

Далее рассмотрим по отдельности каждый из компонентов данной системы. Драйвер-фильтр NDIS является основой, на которой базируется вся построенная система и используется для перехвата и первичной обработки пакетов.

Перехватывать и обрабатывать пакеты на уровне ядра системы позволяет библиотека NDIS, которая представляет собой драйвер операционных систем Windows. NDIS (Network Driver Interface Specification) – это спецификация сетевого интерфейса для драйверов, которая определяет и устанавливает правила того, как сетевые драйверы должны взаимодействовать с сетевым стеком операционной системы Windows [6]. Кроме того, NDIS так же предоставляет библиотеку для работы с сетевым стеком в драйверах. Из основных возможностей, которые предоставляет NDIS можно выделить:

- управлять буферами данных, проходящими через сетевое устройство в связи с чем появляется возможность фильтровать, изменять, добавлять или удалять сетевые данные (например пакеты или сетевые фреймы),

- позволяет получать уведомления и обрабатывать сетевые события устройства, такие как отправка или получение пакетов, изменения состояния устройства и другие, что позволяет создавать различные типы драйверов сетевых устройств,

- имплементация в стек драйверов. Благодаря этой особенности NDIS

может встраиваться в стек уже существующих драйверов на разных уровнях и управлять потоком данных на этих уровнях. К примеру, с его помощью драйвер фильтра NDIS может задержать, отменить, добавить пакеты и передать вверх или вниз по стеку драйверов, что бы например конечное веб или приложение пользовательского режима получила данные в необходимом системе виде или не получила его совсем [7].

Всё вышеописанное наглядно показывает, что NDIS-драйвер фильтр наиболее всего подходит для выполнения поставленной перед ним задачи по перехвату сетевого трафика для дальнейшей его обработки. Данный тип драйверов открывает возможность выполнять предварительную обработку пакетов в сетевом стеке, например фильтровать сетевые пакеты по заданным параметрам или правилам, которое в свою очередь могут быть сгенерированы нейросетью. Так же данный тип драйверов позволяет не только фильтровать, но и изменять сетевые пакеты, которые движутся по стеку, к операциям изменения можно отнести редактирование, удаление или добавление. Вышеописанные операции могут быть использованы в разных сценариях, от передачи данных сетевому-сниферу до создания защищённых систем и новых сетевых протоколов.

В данной работе драйвер-фильтр NDIS используется для передачи сетевого трафика сниферу, который используется для сбора статистики и отображения анализа сетевых пакетов нейронной сетью, а также для получения правил для блокировки потенциально нежелательных сетевых пакетов, основанных на решении нейронной сети.

Пользовательский интерфейс сетевого анализатора или снифер используется в связке с NDIS драйвер-фильтром и нейронной сетью для визуального отображения полученных сетевых пакетов, собранной на их основе статистике и связующим звеном передачи данных между нейронной сетью и драйвером. В качестве связующего звена анализатор приходится использовать по причине того, что на данный момент не представляется возможным взаимодействие с передачей данных нейронной сети из ядра

системы, где и находится драйвер-фильтр без сильных временных задержек, обусловленных не только системными ограничениями, но и скоростью работы с данными в языке python, на котором базируется нейронная сеть. Подробнее производимые операции показаны на диаграмме взаимодействий, изображённой на рисунке 2.

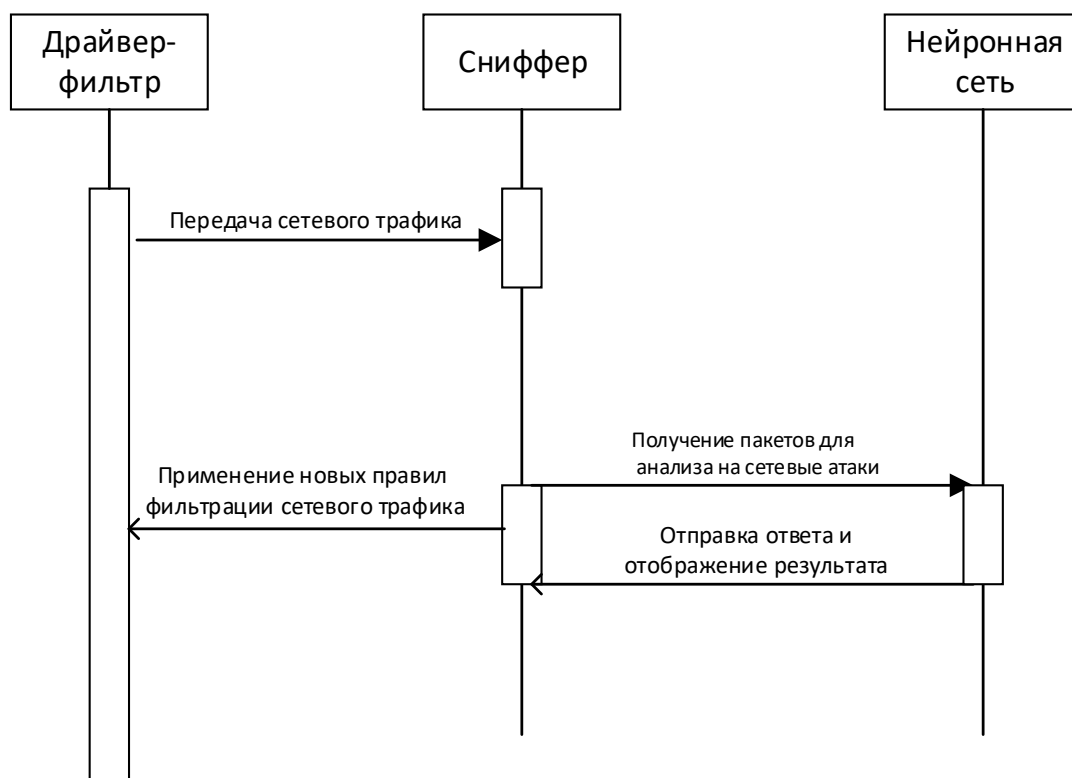


Рисунок 2 – Диаграмма взаимодействия системы

Как можно заметить, основными операциями между компонентами системы является передача данных в формате сетевого трафика или в преобразованном в требуемый для нейронной сети формат, результатов их анализа. Подробнее процесс передачи данных, их формат, процесс изменения во время обработки, формирования и отображение результатов во время работы всей системы можно увидеть на рисунке 3.

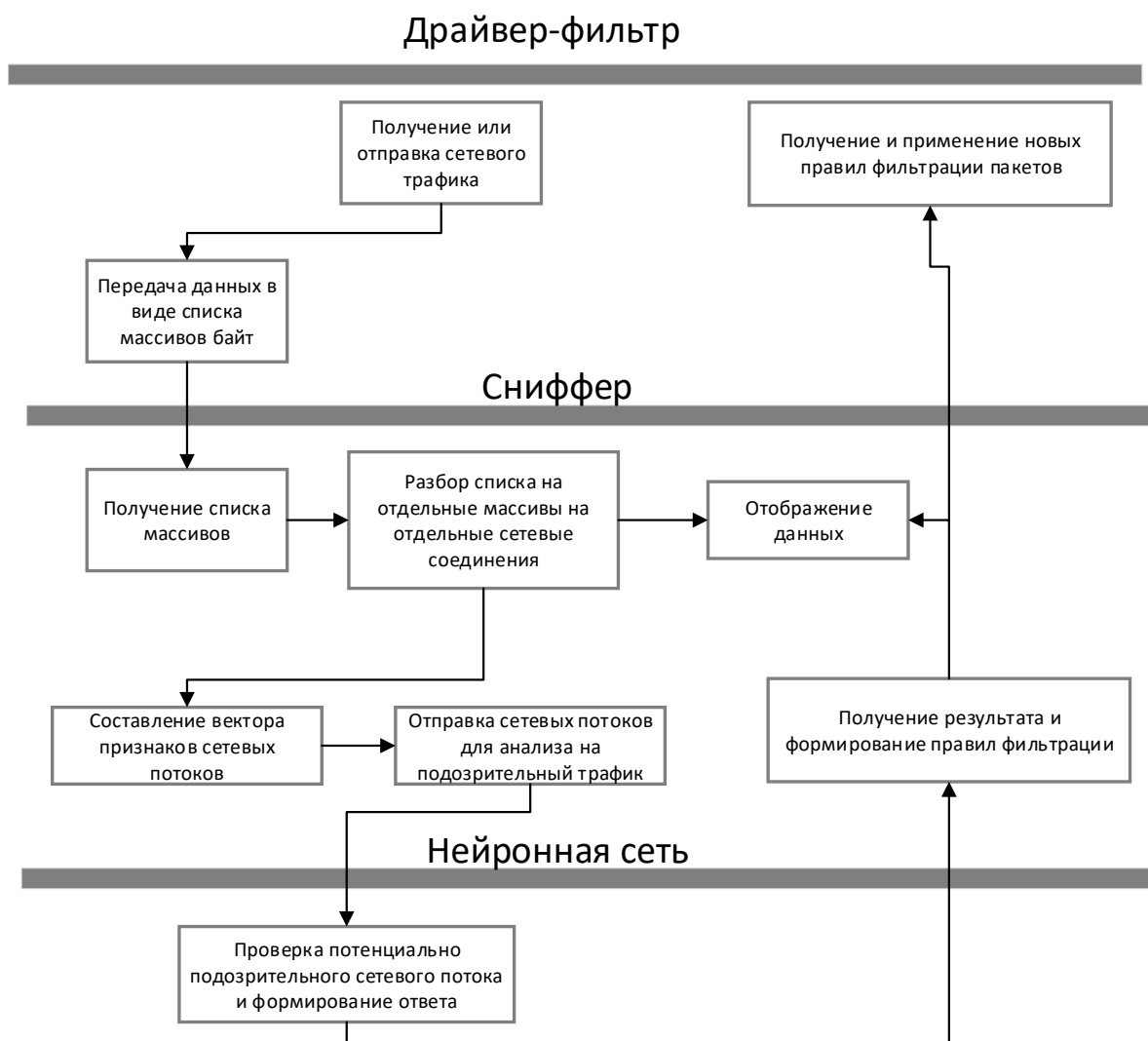


Рисунок 3 – График обработки данных

Из представленной выше схемы можно увидеть, что каждый компонент в системы выполняет свою определённую роль, а вместе работают как единый механизм. Итоговое описание основной выполняемой роли каждого компонента можно описать следующим образом:

- драйвер-фильтр: выполняется роль исполняющего модуля системы, который перехватывает и отправляет сетевой трафик для анализа и фильтрует его на основе полученных правил,

- сниффер: выполняет роль управляющего центра, который получает и преобразовывает данные от драйвера, передаёт обработанные данные нейронной сети и передаёт результаты её анализа обратно в драйвер. Также сниффер является графическим инструментом, с которым непосредственно взаимодействует пользователь,

– нейронная сеть: выполняет роль мозга системы, который отвечает за обнаружение подозрительных соединений, на основе данных полученных и обработанных драйвером и сниффером.

2.2 Поиск эффективного способа обработки сетевого трафика

При работе с сетевым потоком на уровне ядра при помощи сетевых фильтров важно помнить, что любая задержка или неоптимизированный процесс обработки сетевого трафика может привести к сильному замедлению передачи данных или вовсе обрушить сетевое соединение. Поэтому при работе над системой, которая не только перехватывает, но и анализирует сетевой трафик важно решить не только проблему непосредственного анализа, а также оптимизировать алгоритм работы с перехватываемыми данными.

При исследовании и анализе эффективных способов перехвата и передачи сетевого трафика в первую очередь акцент производился на минимизации задержек непосредственно оригинального сетевого трафика, вызываемые ожиданием получающей стороны или время затратными операциями, такими как копирование и перемещение памяти. Устойчивость к непредвиденным ошибкам также является очень важным параметром т.к. на уровне ядра любая ошибка в работе с памятью может привести к критическим ошибкам и падению системы.

При поиске и разработке эффективного алгоритма перехвата и передачи сетевого трафика также необходимо учитывать и формат сетевых пакетов в драйверах фильтрах NDIS, а именно структур, в которых содержатся данные. Во время работы сетевого адаптера при помощи библиотеки NDIS весь сетевой трафик передаётся по стеку сетевых драйверов в виде сетевых листов которая имеет вид структуры NET_BUFFER_LISTS, которая состоит из цепочки сетевых листов (NET_BUFFER_LIST). В каждом отдельном листе цепочки содержится список сетевых буферов (структура NET_BUFFER), содержащих конечные отправляемые или получаемые сетевые пакеты с данными [8]. Стоит отметить, что NET_BUFFER состоит из одного непрерывного буфера с

данными, а имеет более сложную структуру, которую можно увидеть на рисунке 4.

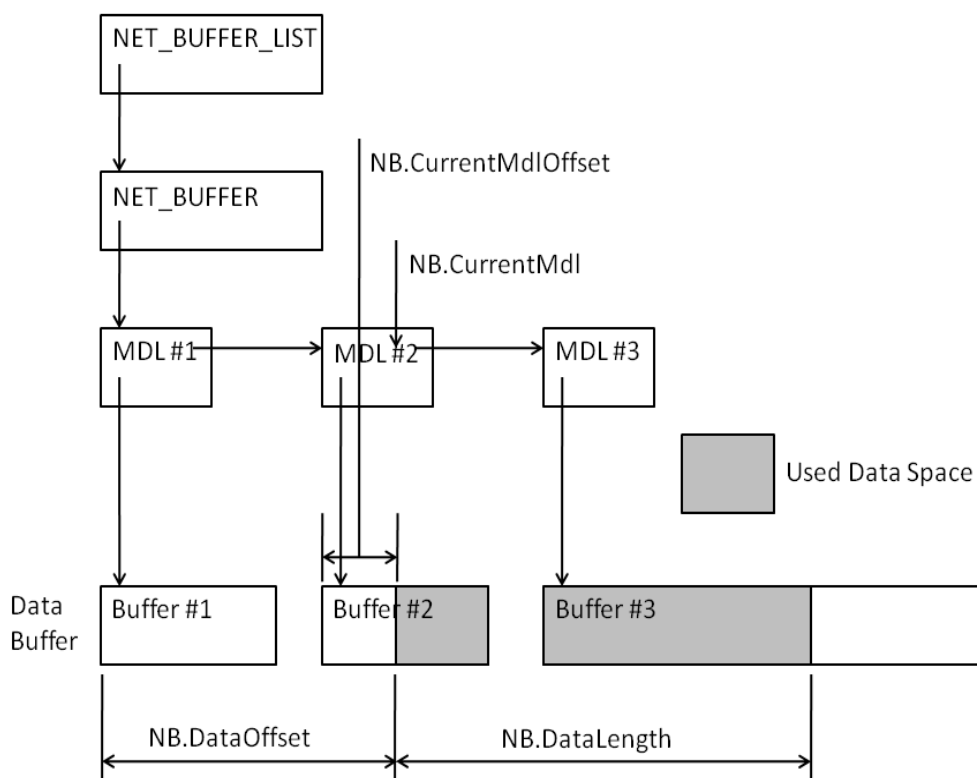


Рисунок 4 – Подробный вид структуры `NET_BUFFER`

Представленная выше структура показывает, что данные одного пакета представляются не в виде одного непрерывного буфера данных, вместо этого данные в пакете находятся в списках структур, называемых MDL (Memory Descriptor List) которая используется системой для описания страниц физической памяти [9]. Так же можно заметить, что данные одного сетевого пакета содержатся в цепочке структур MDL, в которой они распределены неравномерно. Даже внутри пакета от начала самой MDL до начала описываемых данных могут присутствовать (и в большинстве случаев присутствуют) дополнительные смещения. При увеличении количества данных в одном получаемом или передаваемом сетевом пакете увеличивается количество MDL в которых эти данные хранятся. Например, в небольших пакетах, данные которых укладываются в одной MDL могут быть представлены в виде одного непрерывного буфера, но такая ситуация на практике происходит очень редко. Таким образом, если все данные пакета не укладываются в одну

MDL, то представить данные сетевого пакета в виде единого непрерывного буфера становится невозможным без дополнительных манипуляций с памятью. Далее на рисунке 5 представлен пример общей структуры того, как данные располагаются внутри сетевых листов, буферов и MDL.

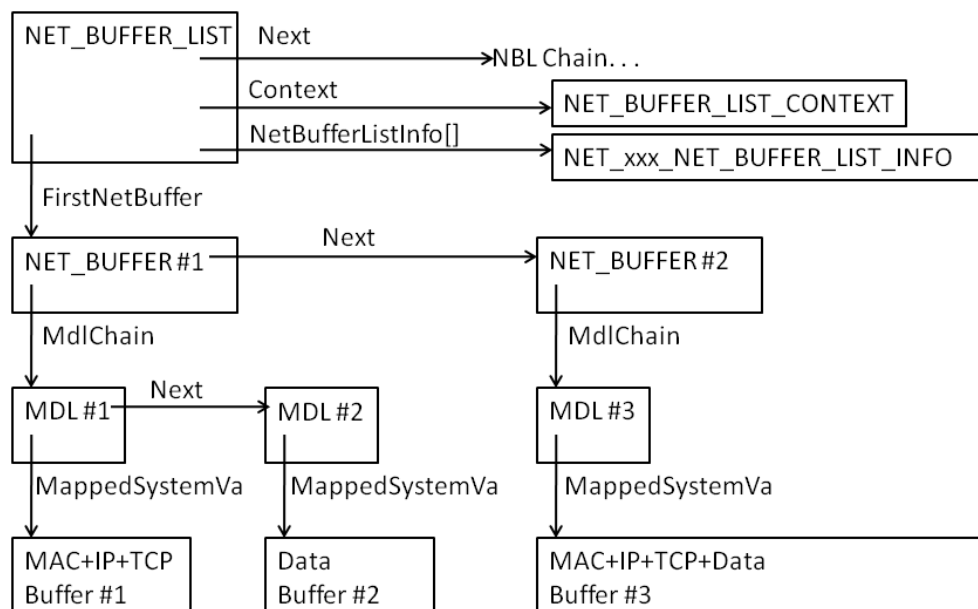


Рисунок 5 – Общая структура данных сетевых листов NDIS

Решением проблемы непоследовательного хранения данных сетевого трафика в памяти и скорости их передачи были решены при помощи разделяемой памяти. Так как копирование данных из пакета и их последующая передача снифферу довольно долгая и трудоёмкая задача, вставить операцию копирования и передачи приложению прямо в потоке получения и отправки сетевых данных не представляется возможным из-за возможной сильной потери в скорости отправки и получения сетевых данных. Решением данной проблемы становится алгоритм постепенного получения данных по мере необходимости из цепочки MDL, принадлежащей одному сетевому пакету и последующая запись полученной области памяти пакета разделяемую память в асинхронном режиме. Таким образом получится избавиться от промежуточной операции копирования данных пакета, а также ожидания

окончания записи, что решит проблему сильного замедления сетевого потока данных при работе системы.

Сам алгоритм работы с разделяемой памятью работает на основе LockFree буфера использующим механизм CompletionPort и Pooling механизм. Основная суть механизма передачи данных при помощи разделяемой памяти при помощи LockFree состоит в том, что, для доступа к памяти не требуется ожидания со стороны драйвера, а сниффер на третьем кольце операционной системы получает данные при помощи Pooling механизма – периодического просмотра памяти на изменение. Передача структура памяти похожа на ту что используют сетевые карты при передачи данных из непосредственно устройства в систему и строится на том что в разделяемой памяти есть указатели на голову записанных данных и хвост, которые периодически проверяет сниффер, если данные сетевых пакетов были получены сниффером, он меняет указатель на голову списка. В драйвере же происходит циклическая запись сетевых пакетов, при этом, указатель на хвост списка меняется только после окончания непосредственной записи в него. В данном алгоритме единственное место где может происходить ожидание для получения или записи данных это момент изменения указателей на голову и хвост списка, однако т.к. время на их изменение не значительное, поэтому для синхронизации используется быстрая spin блокировка, которая работает за счёт атомарной переменной.

Данный алгоритм хорошо себя показывает в ситуациях, когда трафик идёт непрерывно, однако если по какой-то причине передача сетевого трафика остановится для оптимизации работы самой системы была использован механизм CompletionPort позволяющий отправлять уведомления о том, что были получены новые данные. Если сниффер после нескольких проверок не обнаружил изменений в разделяемой памяти он прекращает периодический просмотр изменений, снижая нагрузку на систему и начинает проверять память только после получения уведомления от драйвера.

Остальные взаимодействия между частями системы уже не влияют непосредственно на скорость передачи сетевого трафика, поэтому дополнительные меры по решению проблем по их ускорению принимать не нужно. Взаимодействие между сниффером и нейросетью происходит при помощи сокетов скорость работы, которых никак не замедляют работу разрабатываемой системы, а получение новых правил фильтрации сетевых пакетов может быть отправлена драйверу при помощи IOCTL запроса.

Так как данные из сниффера уже не привязаны к непрерывному сетевому потоку данных, не слишком быстрая работа нейронной сети не повлияет на скорость работы всей системы, а лишь отобразится на скорости визуального отображения полученной информации в окне приложения сниффера и получения драйвером новых правил блокировки. Однако одним из вариантов ускорения работы нейронной сети при необходимости может быть использование ресурсов видеокарты для работы нейронной сети [10].

2.3 Формализованное описание модели обработки трафика

На основе вышеописанного алгоритма передачи трафика и модели взаимодействия разрабатываемой системы было составлено формализованное описание модели. Предлагаемая модель служит основой для реализации и экспериментальной оценки разработанной системы, состоящей из драйвера, обрабатывающего сетевые пакеты и нейронной сети. Далее будет представлено формализованное описание процесса обработки сетевых пакетов [11].

Пусть входной поток сетевых пакетов описывается как последовательность:

$$P = \{p_1, p_2, \dots, p_n\}, \quad (1)$$

$$p_i = (h_i, d_i). \quad (2)$$

где p_i – пакет, содержащий заголовки и данные,

h_i – метаданные (IP-адреса, порты, флаги протоколов),

d_i – полезная нагрузка.

Фильтрация и обработка пакетов в NDIS-драйвере могут быть описаны предикатом F , определяющим, должен ли пакет быть передан дальше, модифицирован или отброшен:

$$F(p_i) \rightarrow \{ACCEPT, DROP, MODIFY\}. \quad (3)$$

Правила фильтрации формируются на основе анализа статистики или решений нейронной сети.

При описании нейронной сети анализа сетевого трафика необходимо учесть её архитектуру. Для классификации и обнаружения аномалий используется многослойный перцептрон (MLP) — полносвязная нейронная сеть с прямым распространением сигнала. Входными данными нейронной сети считается вектор признаков $x \in R^{78}$, извлечённый из набора пакетов p_i , относящихся к одному сетевому соединению, в потоке P который включает в себя:

- объёмные характеристики, например размер и дисперсию размера пакетов,
- временные характеристики потока,
- статистические характеристики потока, например, энтропию портов,
- флаговые характеристики, для TCP соединений,
- характеристики полезной нагрузки.

Архитектура MLP с формальным описанием скрытых слоёв выглядит следующим образом:

Для скрытого слоя $\ell = 1, \dots, 4$:

$$z^\ell = W^\ell a^{\ell-1} + b^\ell, \quad (4)$$

$$h^\ell = \max(0, z^\ell), \quad (5)$$

$$a^\ell = \text{BatchNorm}(h^\ell), \quad (6)$$

$$a^\ell \leftarrow \text{Dropout}_{p^\ell}(a^\ell). \quad (7)$$

где a^0 – стандартизированный выходной вектор,

p – вероятности обнуления нейронов для слоя равное: $p^1 = 0.4, p^2 = 0.3, p^3 = 0.3, p^4 = 0.2$.

Выходной слой (softmax) можно представить в виде следующего выражения:

$$y_k = \frac{\exp(z_k^5)}{\sum_{j=1}^K \exp(z_j^5)}, k = 1, \dots, K \quad (8)$$

где y_k – вероятность принадлежности пакета к классу k .

Выходом нейронной сети является вектор вероятностей и предсказанный класс пакета:

$$y = (y_1, y_2, \dots, y_K), \quad (9)$$

$$\hat{c} = \arg \max (y_k). \quad (10)$$

где y – вектор выходных вероятностей нейронной сети,

$\hat{c}$ – предсказанный класс пакета, выбираемый как элемент с большей вероятностью.

Оптимизация передачи данных между ядром и пользовательским пространством производится за счёт минимизации различных задержек в логике и передаче данных на различных этапах. Для минимизации задержек при передаче пакетов из NDIS-драйвера в пользовательское пространство используется алгоритм асинхронной буферизации. При подготовке данных для

обработки в данном алгоритме драйвер формирует кольцевой буфер в shared memory:

$$B = \{b_1, b_2, \dots, b_k\}. \quad (11)$$

где b – фрагмент данных пакета.

Для выполнения операций записи и чтения фрагментов сетевого пакета применяется следующая формула:

$$b_{write} = (b_{write} + 1) \bmod. \quad (12)$$

Стоит также отметить, что для избегания конфликтов чтения и записи используется lock-free на основе атомарных операций. Далее рассмотрим оценку производительности системы, которая зависит от задержки обработки одного сетевого пакета и точности классификации нейронной сети, которые могут быть описаны следующими формулами:

$$\Delta t = t_{recv} - t_{proc} t_{send}, \quad (13)$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (14)$$

где t_{recv} – время получения, t_{proc} – обработки, t_{send} – передачи,

TP, TN, FP, FN – true/false positives/negatives.

Таким образом описанная выше модель позволяет формализовать процесс обработки трафика, включая такие процессы как:

- фильтрацию на уровне NDIS,
- классификацию с помощью MLP,
- оптимизацию передачи данных.

Описанная выше модель программного анализатора обеспечивает высокую производительность и точность анализа, что подтверждается экспериментальными результатами.

3 Разработка нейронной сети

При выборе архитектуры нейронной сети для выполнения, поставленной в данной работы внимание уделялось скорости обучения и обработки запросов, а также формат входных данных. Так как входными данными нейронной сети является последовательность признаков сетевого потока в качестве архитектуры нейронной сети была выбрана модель многослойного перцептрона (MLP) т.к. она выделяется за счёт следующих важных особенностей по сравнению с другими архитектурами нейронных сетей [12]:

- производит относительно низкую вычислительную нагрузку что важно в контексте разрабатываемой системы,
- отлично справляются с обработкой нелинейных зависимостей, которые часто присутствуют между признаками сетевого трафика, в отличие от, например сверточных сетей, которые больше подходят для пространственной иерархии,
- эффективно работает с высокоразмерными данными, а именно с большим количеством признаков сетевого потока,
- высокая точность классификации при меньших затратах на обучение по сравнению, например с ResNet архитектурами,
- устойчива к зашумлённым данным, по сравнению с LSTM моделями, которые могут накапливать ошибку из-за различных сбоев в сетевом потоке которые встречаются достаточно часто.

Многослойный перцептрон (MLP) представляет собой архитектуру, состоящую из входного слоя, одного или нескольких скрытых слоев и выходного слоя. Данная архитектура относится к классу нейронных сетей прямого распространения (feedforward neural networks), в которых данные движутся строго в одном направлении: от входного слоя к выходному, без образования циклов или обратных связей. Далее на рисунке 6 представлена архитектура многослойного перцептрона с двумя слоями.

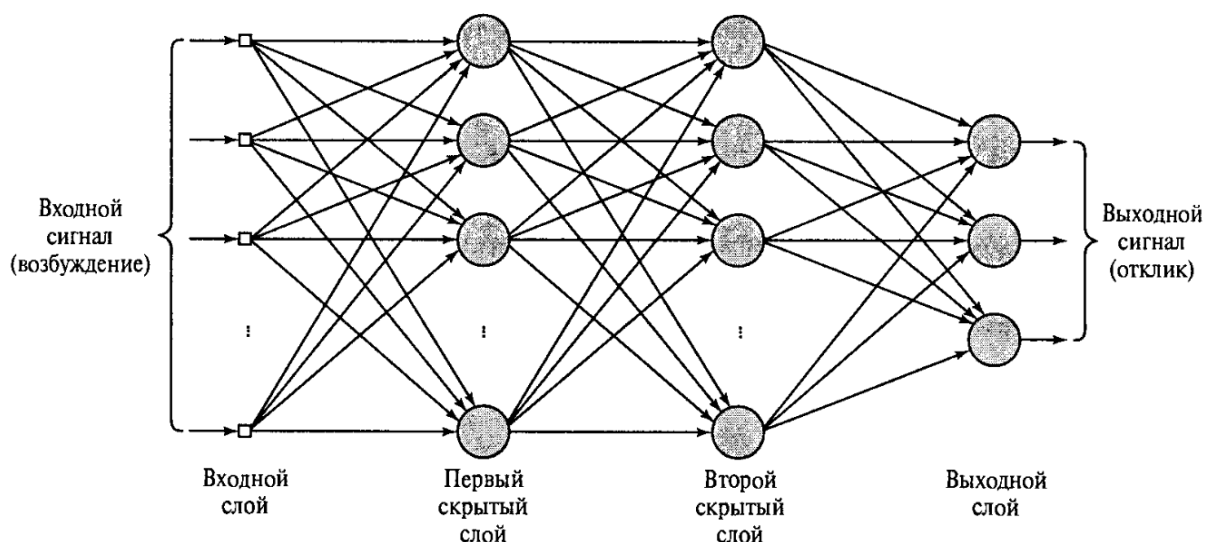


Рисунок 6 – Общая архитектура многослойного перцептрона

Из выше представленного рисунка видно, что архитектура MLP состоит из трёх основных компонентов:

- входной слой, состоящий из признаков обрабатываемого объекта,
- скрытые слои, создающие промежуточное представление объекта,
- выходной слой, количество нейронов, на котором зависит от решаемой задачи.

Действие представленной сети на вектор признаков x может быть записано аналитически:

$$y = f(x) = g(W_3 h(W_2 h(W_1 x + b_1) + b_2) + b_3). \quad (15)$$

где $W_1 \in \mathbb{R}^{K \times D}$, $b_1 \in \mathbb{R}^K$ – матрица весов и вектор смещений 1-го слоя,
 $W_2 \in \mathbb{R}^{M \times K}$, $b_2 \in \mathbb{R}^M$ – матрица весов и вектор смещений 2-го слоя,
 $W_3 \in \mathbb{R}^{P \times M}$, $b_3 \in \mathbb{R}^P$ – матрица весов и вектор смещений выходного слоя,
 g – функция нелинейности для выходного слоя (зависит от решаемой задачи),

h – функция нелинейности скрытых слоёв (для разных слоёв может различаться).

Стоит также отметить, что каждом слое используется нейрон, выдающий константную единицу, чтобы моделировать смещение при расчёте взвешенной

суммы активаций входных нейронов. В многослойном персептроне связи строятся только между нейронами соседних слоёв, причём каждый нейрон предыдущего слоя соединён с каждым нейроном последующего слоя, при этом у каждой такой связи есть собственный вес. По сути, нейроны скрытых слоёв строят промежуточное признаковое представление обрабатываемого объекта, а итоговый прогноз строится по извлечённым признакам последнего скрытого слоя, т.е. работает принцип глубокого обучения, когда модель сама подбирает наиболее информативные признаки, составляя из них входные [13].

При этом промежуточные признаки строятся автоматически, поскольку внутри сети настраиваются все веса: и для извлечения промежуточных признаков на скрытых слоях, и для построения итогового прогноза на выходном слое. Математически преобразование данных в пределах одного слоя можно описать следующим выражением:

$$H = \sigma(XW^{(i)} + b^{(i)}). \quad (16)$$

где σ - нелинейная функция активации,

X – матрица входных признаков размерностью $[\text{batch_size}, \text{n_features}]$,

$W^{(i)}$ – матрица весовых коэффициентов i -го слоя,

$b^{(i)}$ – вектор смещений (bias).

Стоит также отметить, что реальные модели, использующие архитектуру многослойного персептрона на практике, часто используют построения слоёв, при котором количество нейронов в последовательных скрытых слоях постепенно уменьшается, что способствует выделению наиболее информативных латентных признаков и снижению размерности представления данных перед финальным классификатором.

Ключевым элементом, обеспечивающим способность многослойного персептрона моделировать сложные нелинейные зависимости, являются функции активации. Если бы в сети использовались только линейные

преобразования, то вся сеть, независимо от количества слоев, была бы эквивалентна однослойному перцептрону и не могла бы решать задачи, в которых данные не являются линейно разделимыми. Функции активации вносят нелинейность, позволяя сети аппроксимировать практически любые непрерывные функции.

В разрабатываемой системе использованы две функций активации, выбор которых обусловлен поставленной задачей по обеспечению высокой скорости обработки данных, а также необходимостью соблюдения баланса между вычислительной эффективностью и конечным качеством обучения. После выполнения анализа, были выбраны две функции активации, совместное использование которых позволяет решить поставленные задачи оптимальным способом, а именно были выбраны: функция активации ReLU и Softmax [14].

Функция ReLU (Rectified Linear Unit) была выбрана для использования в скрытых слоях нейронной сети, по причине того, что ее математические свойства позволяют преодолеть фундаментальные ограничения классических сигмоидальных функций и обеспечить высокую скорость обучения. Математически её можно описать следующим образом:

$$ReLU(x) = \max(0, x). \quad (17)$$

На рисунке 7 представлен график функции, на котором видно, что функция возвращает нулевое значение для всех отрицательных аргументов и линейно возрастает для положительных.

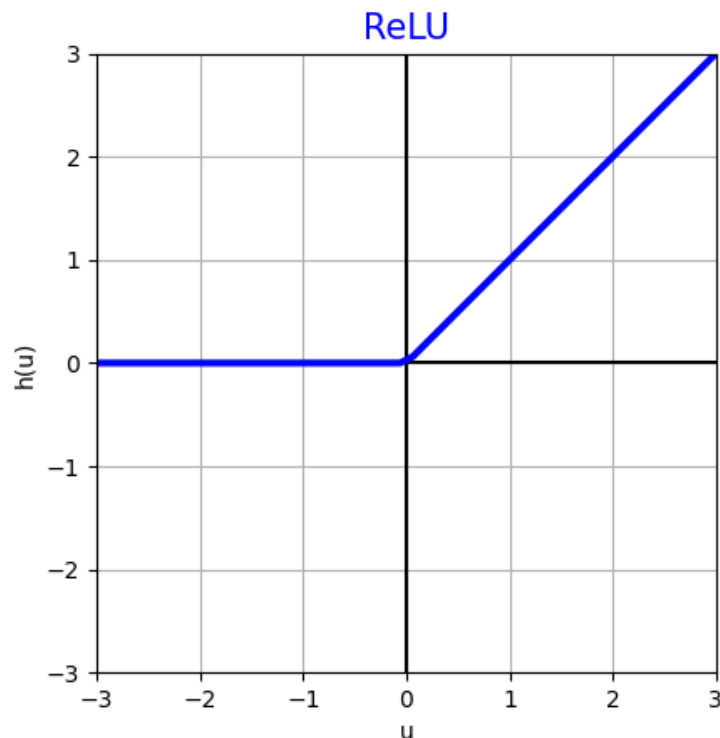


Рисунок 7 – График функции активации ReLU

Выбор функции активации ReLU для скрытых слоёв нейронной сети обусловлен следующими её свойствами:

- вычислительная эффективность, достигаемая за счёт сведения расчёта к сравнению входного значения с нулём,
- разрежённость активаций, за счёт обнуления отрицательных значений, что повышает эффективность использования модели,
- снижение проблемы исчезающего градиента, за счёт того, что производная положительного входного сигнала равна 1.

Стоит также отметить, что проблема умирающих нейронов при обучении, возникающая при систематическом поступлении отрицательных значений, из-за чего градиент нейрона становится нулевым была учтена и будет подробнее описана позднее.

В качестве функции выходного слоя была выбрана функция Softmax, которая позволяет выполнять преобразования вектора выходных нейронов выходного слоя в вектор вероятностей. Такое преобразование возможно благодаря ключевым особенностям функции softmax, а именно: сумма всех

выходных значений всегда равна 1, каждое значение находится на интервале от 0 до 1. За счёт этих особенностей выход нейронной сети может быть интерпретирован как распределение вероятностей по классам, что позволяет определять наиболее вероятный класс, а также уверенность модели. Математически её можно описать следующим образом:

$$Softmax(z_j) = \frac{e^{z_j}}{\sum_{k=1}^K z_k}. \quad (18)$$

где z_j – входное значение j -го нейрона,

k – общее количество классов.

На рисунке 8 представлен график функции, на котором видно, что функция является монотонно возрастающей и имеет S-образную форму, что позволяет эффективно разделять классы и оценивать уверенность модели в своем решении.

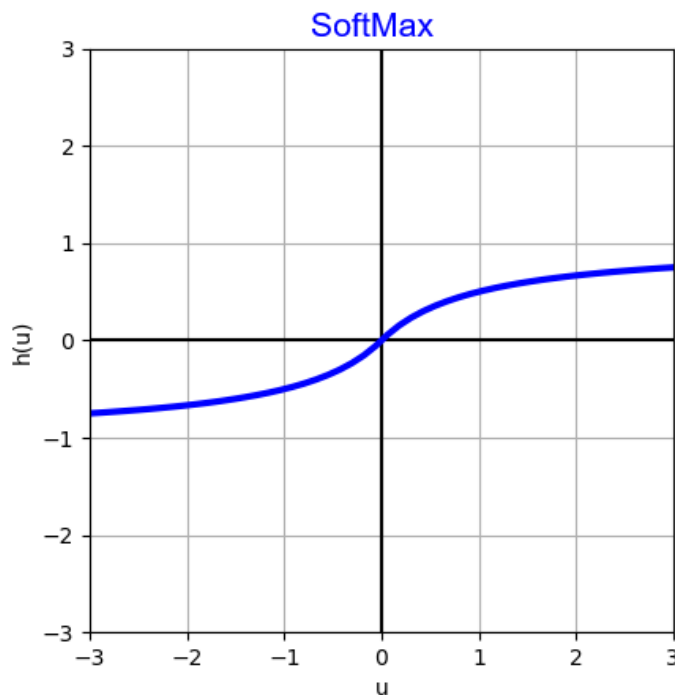


Рисунок 8 – График функции активации SoftMax

Обучение многослойного перцептрона, относящегося к классу сетей прямого распространения, выполняется при помощи алгоритма обратного распространения ошибки (backpropagation) [15]. Данный алгоритм состоит из

последовательного повторения двух этапов, прямого и обратного распространения, которые работают взаимосвязанно, так, например при прямом проходе последовательно выполняются следующие четыре действия:

- 1) входной вектор подаётся на входной слой сети,
- 2) выполняется взвешенное суммирование путём вычисления линейной комбинации входов с весами и добавляется смещение: $z = W \cdot h_{prev} + b$,
- 3) к полученному результату взвешенного суммирования применяется функция активации,
- 4) на выходном слое при помощи функции активации формируется вероятностное распределение по классам для мультиклассовой классификации, или принадлежит ли объект к конкретному классу при бинарной классификации.

Стоит отметить, что пункты 2 и 3 выполняются на всех скрытых слоях сети, вплоть до выходного слоя. При обратном же подходе алгоритм состоит из трёх следующих действий:

- 1) вычисление ошибки на основе предсказания сети и истинной метки рассчитывается значение функции потерь,
- 2) обратное распространение градиентов: начиная с выходного слоя, с помощью правила цепочки последовательно вычисляются частные производные функции потерь по весам каждого слоя,
- 3) обновление весов: полученные градиенты используются оптимизатором для корректировки весов сети с целью минимизации функции потерь.

Таким образом прямое распространение генерирует выходные данные, а обратное распространение использует информацию об ошибке для корректировки весов, чтобы улучшить качество предсказаний. Метод обратного распространения ошибки позволяет эффективно вычислять градиенты функции потерь по всем обучающим параметрам, что необходимо для оптимизации весов. Математически функция потерь описывается как

Стоит отметить, что функция потерь, является важным этапом обучения точной модели нейронной сети. С её помощью оценивается расхождения между предсказанием модели в процессе обучения и реальными значениями, что используется на этапе обратного прохода во время обучения алгоритмом обратного распространения ошибки. Для решения задачи многоклассовой классификации в данной работе оптимальной функцией потерь является категориальная кросс-энтропия из-за своей совместимости с выходной функцией активации softmax, а также повышенным штрафом за неуверенные ошибки на выходе обучения, который даёт более сильный сигнал для исправления весов при уверенной ошибке нейронной сети.

Математически описать функции потерь можно следующим образом: пусть имеется K классов, и для одного обучающего образца сеть выдаёт вектор предсказанных вероятностей $\hat{y}$. Истинная метка образца представлена вектором y , где метка правильного класса равна единице, а для неправильного нулю. Тогда категориальная кросс-энтропия для одного образца определяется как:

$$L(y, \hat{y}) = -\sum_{j=1}^K y_j \log(\hat{y}_j). \quad (19)$$

Поскольку только один элемент y_j равен единице, выражение упрощается до $L = -\log(\hat{y}_c)$, где c — индекс истинного класса. Другими словами, функция потерь равна отрицательному логарифму вероятности, которую модель присвоила правильному классу. Тогда для всей обучающей выборки из N независимых образцов общая функция потерь вычисляется как:

$$L_{total} = \frac{1}{N} \sum_{n=1}^N L(y^n, \hat{y}^n). \quad (20)$$

Однако стоит помнить, что для улучшения качества предсказания и уменьшить ошибку необходимо вычислить градиент функции потерь, который указывает направление наискорейшего возрастания функции потерь.

Соответственно, чтобы уменьшить ошибку, необходимо двигаться в противоположном направлении — вдоль антиградиента, который можно рассчитать по формуле:

$$\nabla L(w) = -\left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \dots, \frac{\partial L}{\partial w_k}\right). \quad (21)$$

где w – веса нейронной сети.

Полученный градиент на этапе обратного прохода используется для корректировки весов и увеличения качества предсказаний нейронной сети.

Как можно заметить важной задачей при обучение нейронной сети является минимизация функции потерь, которая количественно оценивает расхождение между предсказаниями сети и истинными значениями. Для минимизации функции потерь применяются алгоритмы оптимизаторы, которые определяют, как именно должны обновляться веса сети на каждом шаге обучения. Самым базовым алгоритмом оптимизации, является метод градиентного спуска, при котором на каждом шаге обучения веса корректируются в направлении, противоположном градиенту функции потерь. В процессе обучения сеть оптимизирует свои веса путём минимизации средних потерь по обучающей выборке, которые определяются следующим образом:

$$L(w) = \frac{1}{N} \sum_{n=1}^N L_n(w) \rightarrow \min(w). \quad (22)$$

где w – вектор всех весовых коэффициентов и смещений сети,

$L_n(w)$ – функция потерь, оценивающая расхождением между предсказанием и истинным значением для одного объекта.

Поскольку нейросети представляют собой суперпозицию большого числа нелинейных преобразований, функция $L(w)$ является сложной, невыпуклой и не допускает аналитического решения. Минимизация такой

функции выполняется численно с использованием градиентных методов, которые основаны на линейной аппроксимации функции потерь в окрестности текущей точки. Математически описать это можно при помощи следующей формулы:

$$L(\hat{w}) \approx L(w) + w^T \nabla L(w). \quad (23)$$

где $\nabla L(w)$ – градиент функции потерь.

Из этой аппроксимации следует, что наибольшая локальная скорость убывания функции потерь достигается в направлении, противоположном градиенту, то есть вдоль антиградиента ($-\nabla L(w)$). На этом наблюдении основан метод градиентного спуска.

Однако проблемой стандартных градиентных методов оптимизации, является то, что они устанавливают одинаковый шаг обучения вдоль всех направлений в пространстве настраиваемых весов обучаемой модели. Это не оптимально, поскольку:

- вдоль одних направлений функция потерь изменяется резко и быстро, поэтому, во избежание расходимости, нужно выбирать малый шаг обучения,
- вдоль других направлений функция потерь меняется более плавно и постепенно, и шаг обучения нужно выбрать побольше, чтобы ускорить сходимость.

Вследствие этого для методов с фиксированным шагом приходится выбирать малый шаг, что приводит к медленной сходимости по слабо меняющимся направлениям. Из-за чего более эффективным методом оптимизации являются те, которые предполагают динамически подстраивать шаг обучения вдоль каждой оси весов. Примером такого алгоритма может послужить алгоритм Adam (Adaptive Moment Estimation), который является одним из наиболее популярных алгоритмов оптимизации в задачах глубокого обучения благодаря своей вычислительной эффективности. Данный алгоритм оптимизации сочетает в себе две ключевые идеи: во-первых, он вносит

инерцию, сглаживая обновления весов за счёт экспоненциально скользящего среднего градиентов, что позволяет ускорить сходимость в областях с пологим рельефом функции потерь и уменьшить колебания. Во-вторых, он адаптирует скорость обучения индивидуально для каждого веса, используя скользящее среднее квадратов градиентов. В результате чего веса, которые обновляются редко или имеют малые градиенты, получают больший шаг, а часто обновляемые веса — меньший. Обновление каждого веса w_i на шаге t производится по следующим формулам:

$$s_i^t = \beta_1 s_i^{t-1} + (1 - \beta_1) \left(\frac{\partial L_{n:n+B-1}(w)}{\partial w_i} \right), \quad (24)$$

$$g_i^t = \beta_2 g_i^{t-1} + (1 - \beta_2) \left(\frac{\partial L_{n:n+B-1}(w)}{\partial w_i} \right), \quad (25)$$

$$\hat{s}_i^t = \frac{s_i^t}{1 - \beta_1^t}, \quad (26)$$

$$\hat{g}_i^t = \frac{g_i^t}{1 - \beta_2^t}, \quad (27)$$

$$w_i^t = w_i^t - \frac{\eta}{\sqrt{\hat{g}_i^t + \varepsilon}} \hat{s}_i^t. \quad (28)$$

где s_i и g_i — экспоненциально скользящие средние первого и второго моментов градиента соответственно,

β_1 и β_2 — параметры, управляющие скоростью затухания этих средних, лежащие в диапазоне от 0 до 1,

η — базовая скорость обучения,

ε — малая константа, предотвращающая деление на 0.

Коррекция смещения $\hat{s}_i^t$ и $\hat{g}_i^t$ необходима, поскольку начальные значения s_i^0 и g_i^0 равные нулю приводят к заниженным оценкам моментов на

первых итерациях. Благодаря делению на $1 - \beta^t$ эти оценки быстро становятся несмещёнными.

Однако стоит помнить, что при обучении нейронных сетей всегда присутствует риск переобучения, при котором сеть запоминает обучающую выборку, вместо того чтобы учиться находить закономерности, в результате чего высокая точность ответов на такой сети будет только на обучающих данных, но не на новых примерах, которые нейросеть ещё не видела. Предотвратить переобучение можно разными методами при этом их зачастую можно и нужно комбинировать, в данной работе используются следующие механизмы предотвращения переобучения:

- dropout - метод обнуляющий часть нейронов в слое,
- batch normalization - пакетная нормализация,
- early stopping - ранняя остановка обучения,
- reduceLROnPlateau - изменение скорости обучения.

Dropout или же слои прореживания встраиваются в структуру нейронной сети и препятствуют переобучению за счёт того, что на этих слоях рассчитывается вероятность отключения p нейронов на предыдущем слое сети. Отключение нейронов при этом происходит при помощи умножения вероятности сохранения связи с нейроном на предыдущем слое, а сама вероятность равна $1 - p$. На рисунке 9 представлен пример использования слоя dropout к нейронной сети из 4 слоёв.

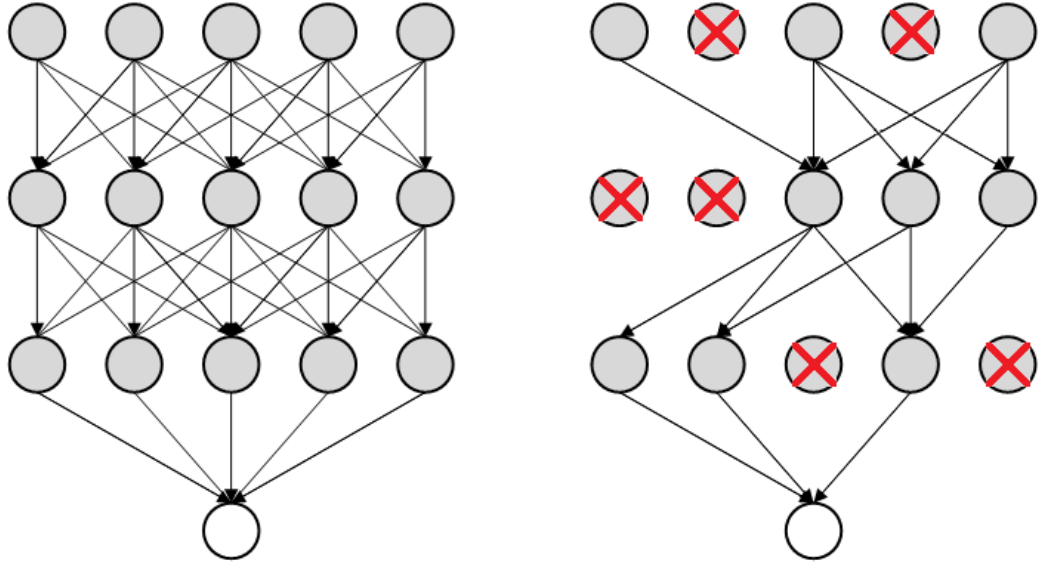


Рисунок 9 – Результат работы dropout слоя

Batch Normalization также является одним из слоёв нейронной сети который нормализует выходы слоя внутри каждого батча, что приводит их к нулевому среднему или единичной дисперсии. Этот слой позволяет ускорить сходимость и уменьшает зависимость от начальной инициализации весов позволяет, а также позволяет при помощи добавления шума в функцию активации дополнительно снизить переобучение. Для батча из m образцов с h слоями данную функцию нормализации можно представить следующим образом:

$$\mu_B = \frac{1}{m} \sum_{i=1}^m h^i, \quad (29)$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (h^i - \mu_B)^2, \quad (30)$$

$$\hat{h}^i = \frac{h^i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \quad (31)$$

$$h_{out}^i = \gamma \hat{h}^i + \beta. \quad (32)$$

где μ_B – среднее по батчу,

σ_B^2 – дисперсия по батчу,

$\hat{h}^i$ – нормализованный выход,

ε – малая константа численной устойчивости,

h_{out}^i – окончательный выход слоя,

γ и β – обучаемые параметры масштаба и сдвига, восстанавливающие исходное распределение.

Также существуют способы предотвращения переобучения, которые работают не как слои нейронной сети, а как callback функции отслеживающие параметры нейронной сети в процессе обучения, такой функцией, например является early stopping. В данном методе отслеживается значение функции потерь на вариационной выборке и, если в течении заданного количества эпох улучшений не наблюдается, обучение приостанавливается и восстанавливаются веса. На рисунке 10 приведён график обучения сети, на котором красными точками отмечены места срабатывания early stopping.



Рисунок 10 – График обучения с early stopping

Вторым примером функции, отслеживающей параметры обучения нейронной сети, является ReduceLROnPlateau. Данный метод динамически снижает скорость обучения, если значение функции потерь на валидационной выборке не улучшается в течении заданного количества эпох что позволяет делать более мелкие шаги приближения к минимумам, что способствует более

точной настройке весов и уменьшает расходимость. Данный метод отлично дополняет early stopping, что обеспечивает плавную адаптацию параметров в процессе обучения.

3.1 Описание разработанной нейронной сети

Нейронная сеть, разработанная в рамках данной работы, анализирует сетевой трафик на основе 78 признаков, среди которых можно выделить направление передачи, ip и mac адреса, порты, наличие флагов и параметров передачи данных по tcp протоколу и их количество, а также различные временные промежутки передачи данных. Как можно заметить большинство признаков указывают на то, что анализируются не одиночные сетевые пакеты, массивы из пакетов, на основе которых собираются признаки. В целом такой метод позволяет более точно определить является ли соединение вредоносным или же нет, так как в большинстве случаев по одному сетевому пакету невозможно понять является ли он потенциально опасным или же нет.

Такой анализ называется потоковым, т.е. нейронная сеть анализирует целый поток сетевого трафика, относящегося к конкретному соединению. Поток данных при этом является двунаправленным, таким образом при защите конечных точек и обнаружении сетевых атак при помощи разрабатываемого механизма анализа на основе нейронной сети потенциально опасно соединение можно обнаружить если с защищаемой машины отправляется большое количество сетевого трафика к определённому получателю, что может указывать на то, что сеть конечного хоста пытаются перегрузить или вытягивают из неё данные. Или напротив если на конечную машину отправляется большое количество сетевых пакетов с определённого хоста, что может указывать на port scan, flood и другие типы сетевых атак.

Таким образом при подготовке данных, отправляемых нейронной сети на анализ, появляется необходимость разбивать их на отдельные потоки, основываясь на адресах отправителя и получателя. При этом массива сырых сетевых пакетов необходимо выделить и собрать все необходимые признаки чтобы представить их нейронной сети в требуемом для неё формате.

3.2 Выбор датасета

При обучении нейронной сети, которая выполняет задачу обеспечения безопасности, критически важно подобрать правильный набор данных. Использование устаревших или неполных данных может привести к созданию модели, которая будет демонстрировать высокую эффективность только в лабораторных условиях, но окажется совершенно бесполезной при работе в реальной сети, или будет пропускать часть сетевых атак из-за т.к. для их обнаружения просто не обучалась. Таким образом для обеспечения актуальности и практической значимости результатов, в качестве основного набора данных для данной работы был выбран общедоступный набор данных CIC-IDS-2017, подготовленный Канадским институтом кибербезопасности.

Выбор данного набора данных обусловлен их актуальностью и репрезентативностью, которая обеспечивается датой создания набора, а также того факта что все данные были собраны вручную, а не создавались синтетически. Также стоит отметить, что данный набор данных является набором общего типа т.е. сетевой трафик в нём собран в условиях обычной сети, это важно учитывать по причине того, что существует большое количество наборов данных, которые, однако, имеют узкую специализацию. Такие наборы данных могут эффективно представлять сетевой трафик только, например в пределах их сбора и использования (сетевой трафик компании или организационной структуры) или типа сети (Wifi, IOT и другие). Общий список рассматриваемых наборов данных включал в себя:

- KDD Cup 99 – содержит большое количество записей который, однако, были созданы синтетически, и за счёт года (1999) сетевые атаки, представленные в нём, являются устаревшими,
- NSL-KDD – более новый (2009 год) однако все ещё синтетический набор данных, в котором представлено слишком мало записей,
- UNSW-NB15 – набор 2015 года, который содержит как синтетически, так и реальные данные, однако трафик в нём разбит лишь на 49 признаков,
- CIC-Modbus2023 – новый набор данных, который, однако,

специализируется на промышленных сетях, использующих протокол Modbus,

- AWID – набор 2025 года, который, однако, включает сетевой трафик Wifi сетей, который не подходит для поставленной в данной работе задаче.

Датасет CIC-IDS-2017 представляет собой набор сетевых пакетов, собранных в течение пяти дней и содержит около 2.8 миллионов записей сетевых потоков, которые представляют собой наборы двунаправленного сетевого трафика собираются из группы сетевых пакетов принадлежащих к одному конкретному сетевому соединению, каждый из которых размечен как нормальный или аномальный трафик с указанием конкретного типа атаки, включая такие как:

- DoS (Denial of Service) и DDoS (Distributed DoS – атаки, нацеленные на отказ в обслуживании,

- port scan – сканирование портов для выявления уязвимостей,

- brute force – атаки подбора паролей для FTP и SSH,

- web attack – веб-атаки, включая внедрение SQL-кода и межсайтовый скриптинг (XSS),

- infiltration – попытки проникновения в систему изнутри сети,

- botnet – трафик, генерируемый сетью зараженных ботов,

- heartbleed – эксплуатация уязвимости в OpenSSL.

В CIC-IDS-2017 данные представляются в виде векторов признаков сетевых потоков, которые получаются из сетевого трафика, относящегося к одному соединению, также стоит отметить, что этот поток является двунаправленным. Вектор признаков содержит 78 числовых признаков, которые можно разбить на следующие логические группы:

- базовые идентификаторы потока - состоят из уникального идентификатора потока ip и mac адресов, а также портов. Данные параметры используются не для обучения, а для идентификации и группировки сетевого трафика,

- временные характеристики - характеристики длительности потока, а также его средних, минимальных и максимальных параметров в разных

аспектах. Эти признаки позволяют выявлять атаки, генерирующие пакеты с регулярными интервалами или, наоборот, с хаотичными задержками,

- объёмные характеристики - характеристики размеров сетевого трафика потока, а также его средних, минимальных и максимальных параметров в разных аспектах, а также скорости и интенсивности потока. Эти признаки важны для обнаружения DoS/DDoS-атак, при которых резко возрастает количество пакетов или объём трафика,

- статические характеристики размеров пакетов - статистические характеристики минимальных, максимальных или средних размеров или стандартных отклонений для сетевых пакетов в потоке. Эти признаки помогают выявлять аномалии, например, очень маленькие пакеты, которые характерны для сканирования или необычно большие пакеты,

- признаки TCP соединения - учитывает флаги работы TCP соединения, а также другие динамические параметры TCP соединения, такие как размер окна или размеры сегментов. Эти признаки критически важны для обнаружения PortScan или DoS-атак с аномальными комбинациями флагов, а также для анализа легитимности установления соединений.

3.3 Подготовка обучающих данных

Как говорилось выше в датасете CIC-IDS-2017 содержится 2.8 миллионов записей сетевых потоков из 78 признаков, которые содержат большое количество типов сетевых атак, однако без должной обработки обученная нейросеть получится очень слабой. Поэтому перед обучением модели необходимо выполнить ряд операций по очистке, преобразованию и нормализации данных.

В первую очередь стоит отметить, что в наборе CIC-IDS-2017 присутствуют записи с нечисловыми параметрами, в местах где данные повреждены или отсутствуют что приводит к проблемам с обучением, поэтому важным шагом является избавиться от этих параметров путём замены их на 0. Также важно учесть что в датасете количество сетевых потоков по типам атак распределено неравномерно, существуют такие атаки которые

характеризуются недостаточным количеством примеров в выборке (менее 0.1%). Если брать набор данных без изменений количество примеров для обучения сети на обнаружение таких атак будет крайне мало, в следствии чего точность модели сильно снизиться. Данную проблему пришлось решать отсеканием редких классов, которые были нерепрезентативны в выборке. На рисунке 11 показано распределение классов атак до и после отсечения.

```
Фильтрация классов
Классы до фильтрации:
0: benign -> 1743179
1: bot -> 1966
2: ddos -> 128027
3: dos goldeneye -> 10293
4: dos hulk -> 231073
5: dos slowhttptest -> 5499
6: dos slowloris -> 5796
7: ftp-patator -> 7938
8: heartbleed -> 11
9: infiltration -> 36
10: portscan -> 158930
11: ssh-patator -> 5897
12: web attack ⚡ brute force -> 1507
13: web attack ⚡ sql injection -> 21
14: web attack ⚡ xss -> 652

Оставлено классов: 6
Маппинг классов:
0 -> benign
1 -> ddos
2 -> dos goldeneye
3 -> dos hulk
4 -> ftp-patator
5 -> portscan
```

Рисунок 11 – Результаты отсечение нерепрезентативных классов

Стоит помнить, что нейронные сети чувствительны к масштабу признаков, некоторые из которых могут иметь разные диапазоны значений (например, длительность потока – миллионы микросекунд, а количество флагов – единицы), что может приводить к нестабильному обучению. Для устранения этого эффекта применяется стандартизация, которая преобразует каждый признак к распределению с нулевым средним и единичным стандартным отклонением. Также учитывая дисбаланс классов в выборке, который также отображён на рисунке 11, необходимо добавить веса классов, чтобы модель при обучении не игнорировала минорные классы. Веса классов при этом вычисляются по следующей формуле:

$$w_i = \frac{N}{K \cdot n_i}. \quad (33)$$

где N – общее количество образцов в обучающей выборке,

K – количество классов,

n_i – количество образцов i -го класса.

Таким образом после всех преобразований, выборка разбивается на 3 подвыборки: обучающую – 70% от общего количества, тестовую – 15% и валидационную – 15%. Стоит отметить, что разделение выполняется в два этапа с сохранением пропорций классов и после выполнения подготовки обучающей выборки можно переходить непосредственно к обучению нейронной сети.

3.4 Архитектура и обучение нейронной сети

На основе подготовленного набора данных была разработана и обучена нейросеть для многоклассовой классификации сетевых атак, в качестве инструмента разработки была выбрана библиотека TensorFlow с высокоуровневым API Keras. Как уже говорилось выше разработанная модель представляет собой полносвязную нейронную сеть прямого распространения с архитектурой MLP, которая построена по принципу постепенного уменьшения размерности: каждый последующий скрытый слой содержит меньше нейронов, чем предыдущий, что способствует выделению наиболее информативных латентных признаков и снижает вычислительную сложность. При формировании архитектуры модели в коде обучения использовалась функция `keras.Sequential`, полный код формирования архитектуры имеет следующий вид:

```
model = keras.Sequential([
    layers.Input(shape=(X_train.shape[1],)),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.4),
```

```

layers.Dense(256, activation='relu'),
layers.BatchNormalization(),
layers.Dropout(0.3),
layers.Dense(128, activation='relu'),
layers.BatchNormalization(),
layers.Dropout(0.3),
layers.Dense(64, activation='relu'),
layers.BatchNormalization(),
layers.Dropout(0.2),
layers.Dense(num_classes, activation='softmax')
])

```

Как можно заметить нейронная сеть стоит из 4 слоёв, которые логически можно разбить на блоки, содержащие:

- слои нейронов размерностью 512, 256, 128 и 64 соответственно. Сужение размерности от 512 до 64 нейронов позволяет сети последовательно выделять абстракции: первый слой может фиксировать грубые корреляции между признаками, а последующие - формировать компактное латентное представление, пригодное для классификации, что уменьшает риск переобучения, так как количество параметров снижается к выходу,

- функция активации ReLU была выбрана из-за устойчивости к проблеме исчезающего градиента и вычислительной эффективности, более сложные функции (selu, elu) не дали значимого прироста точности на предварительных тестах,

- Batch Normalization позволяет использовать более высокую скорость обучения и ускоряет сходимость в 2–3 раза, кроме того, она стабилизирует распределение активаций, что особенно важно для разнородных признаков сетевого трафика,

- Dropout с убывающей вероятностью равной 0.4, 0.3, 0.3 и 0.2 соответственно. Такое распределение вероятностей объясняется тем, что первые слои содержат больше параметров и сильнее подвержены переобучению, поэтому для них используется более агрессивный dropout. Последний слой в свою очередь имеет мало нейронов, и для него достаточно

слабой регуляризации.

Настройка модели производится при помощи метода `compile`, при помощи который выбираются метрики, функция потерь и оптимизатор. В качестве оптимизатора используется Adam со скоростью обучения 0.001, так как он сочетает адаптивную скорость обучения для каждого параметра с инерцией, что обеспечивает быструю сходимость и устойчивость к зашумлённым градиентам сетевого трафика. В качестве функции потерь используется категориальная кросс энтропия который не требует преобразования меток в one-hot-векторы, а также позволяет работать с закодированными метками классов. Она штрафует модель за уверенные ошибочные предсказания и корректно работает с выходным слоем Softmax. Подробнее каждый из этих методов был описан в соответствующей главе разработки архитектуры нейронной сети. В качестве основной метрики в процессе обучения используется accuracy, а дополнительные метрики такие как precision, recall, F1-score вычисляются на этапе расширенного тестирования после обучения модели. Код компиляции модели представлен ниже:

```
model.compile(  
    optimizer=keras.optimizers.Adam(learning_rate=0.001),  
    loss='sparse_categorical_crossentropy',  
    metrics=['accuracy']  
)
```

Перед началом обучения модели также создаются callback функции, которые используются для автоматической настройки и управлением в процессе обучения:

```
early_stop = callbacks.EarlyStopping(  
    monitor='val_loss',  
    patience=15,  
    restore_best_weights=True,  
    verbose=1  
)
```

```

reduce_lr = callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    min_lr=0.00001,
    verbose=1
)

```

Первой callback функцией является EarlyStopping, которая отслеживает значения функции потерь на валидационной выборке и в случае, если в течении 15 эпох не происходит улучшений обучение автоматически восстанавливается, а также позволяет восстанавливать веса из эпохи с наименьшим выходным значением функции потерь. Второй callback функцией является ReduceLROnPlateau которая также отслеживает значение функции потерь и, если в течении 5 эпох улучшения отсутствуют умножается скорость обучения на 0.5, что позволяет точнее настраивать веса в процессе обучения, особенно на последних его стадиях.

Непосредственно обучение нейронной сети производится при помощи функции fit, код которой представлен ниже:

```

history = model.fit(
    X_train_scaled, y_train.astype(np.int32),
    validation_data=(X_val_scaled, y_val.astype(np.int32)),
    epochs=200,
    batch_size=256,
    class_weight=class_weights,
    callbacks=[early_stop, reduce_lr],
    verbose=1
)

```

Как можно заметить описанные ранее callback функции передаются в эту функцию, что позволяет их использовать во время обучения модели. Количество эпох имеет значение 200 и задано заведомо больше, чем нужно из-за присутствия EarlyStopping. Размер батча имеет значение 256 и был подобран

экспериментально, меньший батч давал более шумные градиенты и дольше сходимость, а больший требовал больше памяти и не улучшал точность. Также из-за дисбаланса классов в датасете при обучении используются веса классов, т.к. без них обученная модель склонна игнорировать редкие типы атак. Формула расчёта весов классов представлена в главе подготовки данных.

3.5 Результаты обучения нейронной сети

На основе описанных ранее этапов по выбору датасета, подготовки обучающих наборов данных, а также построении архитектуры и обучении нейронной сети была получена модель многослойного перцептрона, способная обнаруживать сетевые атаки нескольких классов. Обучение сети с исходными параметрами заняло 27 минут и выполнялось в течении 42 эпох, при этом точность на тесте после обучения составила 98.5%. На рисунке 12 изображён график изменения точности и функции потерь во время процесса обучения.

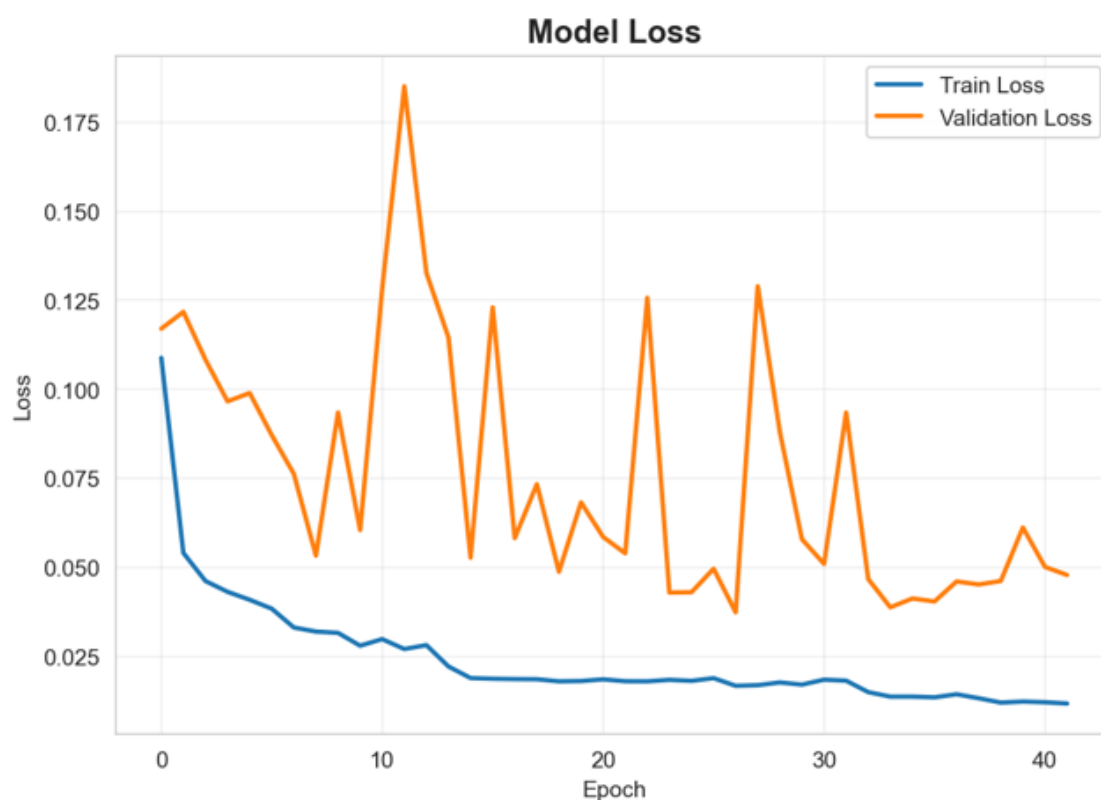


Рисунок 12 – График изменения точности и функции потерь

Из выше представленного графика можно заметить, что обучение было

остановлено механизмом EarlyStopping поскольку валидационная ошибка перестала улучшаться. Также из этого графика можно заметить что скорость обучения была снижена несколько раз благодаря механизму ReduceLROnPlateau на 14 и 33 эпохах, на графике при этом можно заметить уменьшение потерь на обучающей выборке, из чего можно сделать вывод что данный механизм эффективно позволяет улучшать метрики обучения даже на поздних его стадиях. Также разница между обучающей и валидационной потерями меньше 0.1 что является незначительным и позволяет указать на отсутствие переобучения модели.

Оценить общее качество обучения нейронной сети с многоклассовой классификацией можно использовать такие параметры как precision (точность) показывающий долю верно предсказанных объектов, recall (полнота) показывающая долю верно предсказанных объектов класса, F1-score (гармоническое среднее) которая позволяет получить единую метрику из precision и recall, при этом данные параметры относятся к каждому классу в нейронной сети. Далее в таблице 1 представлены данные метрики классификации для каждого выходного класса нейронной сети [16].

Таблица 1 – Метрики классификации после обучения

Класс	Precision	Recall	F1-score	Поддержка
benign	1.00	0.98	0.99	261 477
ddos	1.00	1.00	1.00	19 204
dos oldeneye	0.80	1.00	0.89	1 544
dos hulk	0.90	1.00	0.95	34 661
ftp-patator	0.80	1.00	0.89	1 191
portscan	0.98	1.00	0.99	23 839

Из данной таблицы можно заметить, что значения recall для всех классов атак равно 1 что означает что модель все тестовые образцы, помеченные как атака классифицировала как атаки, не пропустив ни одной. Однако precision для классов с малым количеством образцов составляет всего 0.8 то есть 20%

образцов, предсказанных как эти атаки, на самом деле относятся к другим типам атак, что является приемлемым компромиссом при работе с несбалансированными данными, модель стремится не пропустить атаку, но может ошибаться в её точном типе. При этом параметр F1-score держится на хорошем уровне для всех классов сетевого трафика.

Матрица ошибок, представленная на рисунке 13, также подтверждает таблицу метрик классификации, а также тенденцию обученной модели к обнаружению именно сетевых атак. Как можно увидеть из рисунка из всех тестовой выборки в 300 тысяч объектов, модель определила всего 35 атак за обычный трафик что является допустимой погрешностью.

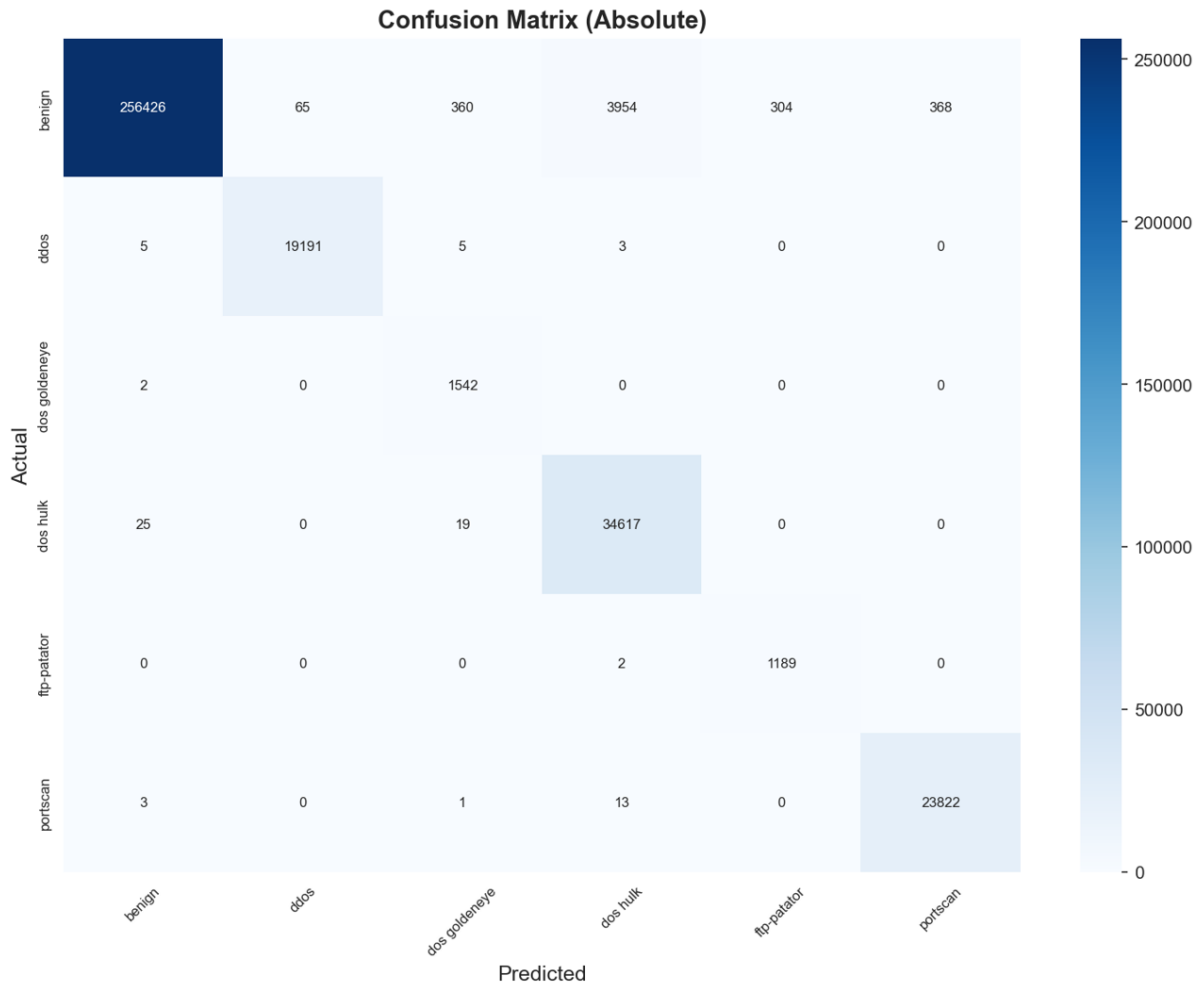


Рисунок 13 – Матрица ошибок

Для того чтобы точнее оценить, насколько предсказанные нейронной сетью вероятности соответствуют частотам правильных ответов использовалась метрика средняя ошибка калибровки (ECE), которая

показывает, насколько в среднем модель отклоняется от реальных частот, при этом приемлемой частотой являются значения до 0.15, а также график калибровочных кривых. Из графика, отображённого на рисунке 14 видно, что на всех классах модель демонстрирует отличную калибровку на всех классах >0.02 , кроме dos goldeneye и ftp-patator, у которых значение ECE находится в районе 0.1, что однако объясняется малым количеством обучающих примеров для данных классов атак в обучающей и валидационной выборке.

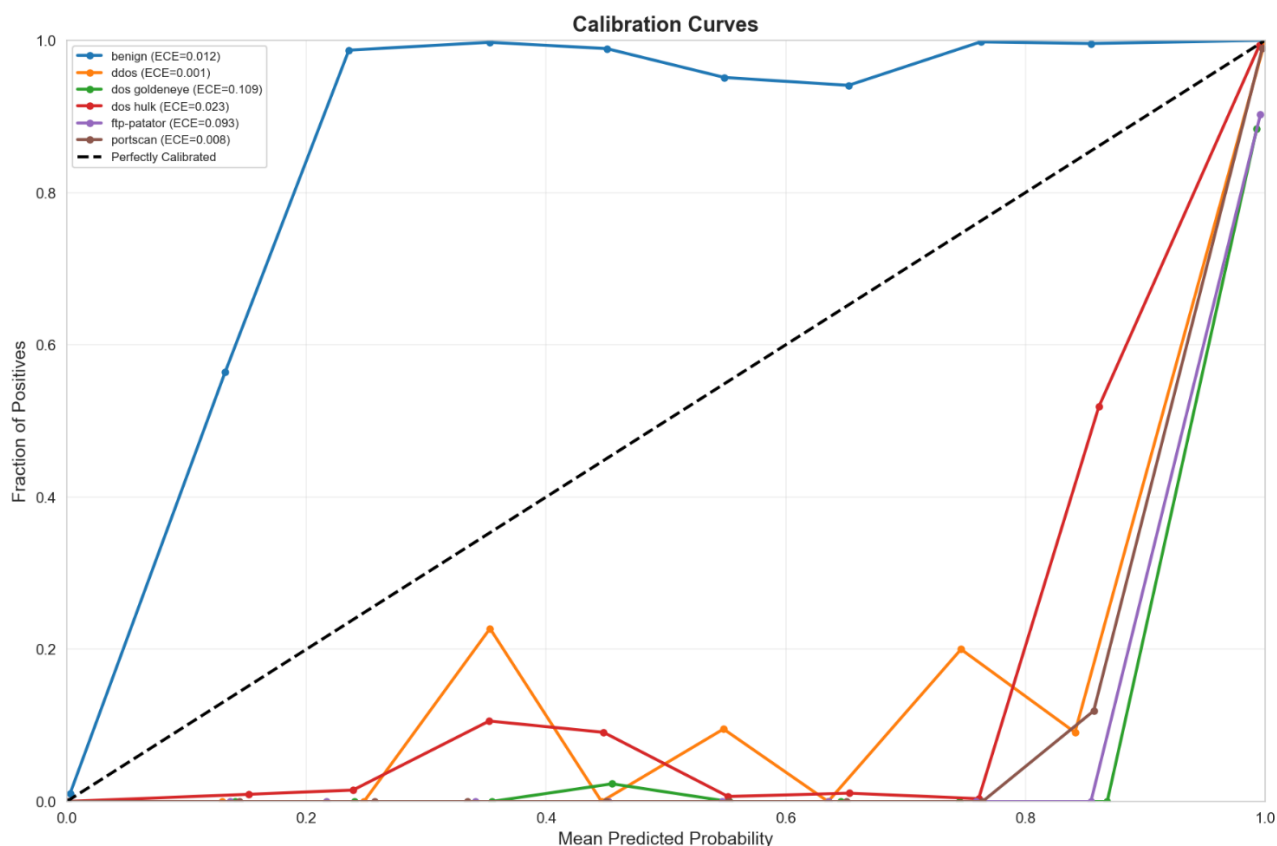


Рисунок 14 – График средней калибровки ошибок

Также хорошую калибровку модели подтверждает график уверенности модели в правильных и неправильных ответах, представленный на рисунке 15. На нём можно увидеть, что модель как правило менее уверена в своих ошибочных решениях. Также на графике отчётливо заметно что правильные предсказания концентрируются в одной области с высокой уверенностью от 0.95 до 1, а ошибочные предсказания распределены более равномерно и находятся в диапазоне от 0.5 до 0.9 и концентрируются в 0.8.

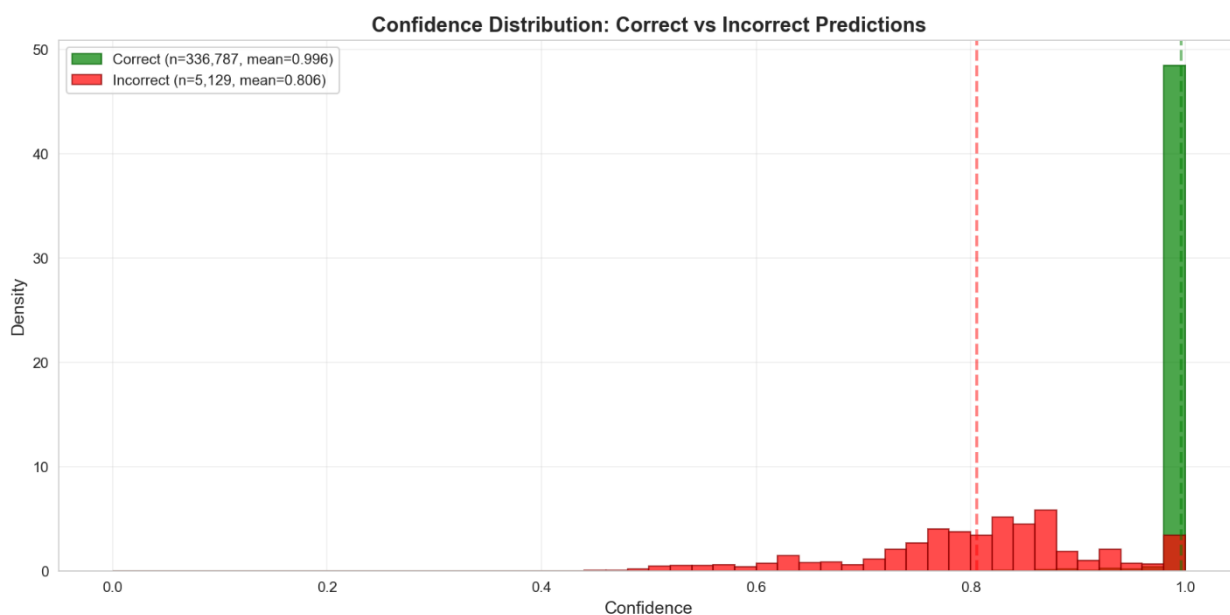


Рисунок 15 График уверенности модели

Также тестирование обученной модели включало в себя оценку применимости модели в системах реального времени для этого было измерено среднее время предсказания одного образца при работе модели на CPU. При размере батча в 100 потоков среднее время составило 46.4 мс, что соответствует пропускной способности 2155 образцов в секунду, что является отличным результатом скорости для большинства систем мониторинга сетевого трафика, так как речь в случае с данной моделью ведётся не про одиночные сетевые пакеты, а сетевые потоки соединений.

После тестирования разработанной модели многослойного перцептрона для обнаружения сетевых атак, обученная на датасете CIC-IDS-2017. можно с уверенностью сказать, что она продемонстрировала высокую эффективность многоклассовой классификации. Точность 98%, а также практически полное обнаружение атак подтверждают правильность выбранной архитектуры, набора данных, алгоритмов и функций для оптимизации. При этом выбранная модель также обладает высокой пропускной способностью что делает её отличным решением для использования в разрабатываемой системе анализа сетевого трафика в режиме реального времени.

4 Установка и тестирование программного комплекса

Перед началом использования разработанного в рамках данной работы программного комплекса необходимо выполнить следующие действия:

- отключить проверку подписи драйверов в операционной системе Windows или установить самоподписанный сертификат драйвера в хранилище доверенных сертификатов,
- выбрать сетевой адаптер и установить на него собранный драйвер перехвата сетевого трафика,
- запустить python приложение, которое загрузит обученную модель нейронной сети из файла и будет принимать вектора признаков сетевых потоков для анализа и отправлять его результаты,
- запустить приложение графического интерфейса и выбрать драйвер, для которого в данный момент будет производиться анализ сетевого трафика.

Важно отметить, что собранный драйвер компонента перехвата сетевого трафика имеет версию NDIS 6.20 и был проверен на операционной системе Windows 10, совместимость с версиями Windows 7 по Windows 11, а также Windows Server 2012 по Windows Server 2025 теоретически возможна, и разработанный компонент-драйвер должен работать на данных операционных системах.

4.1 Установка драйвера

После сборки компонента драйвера-перехватчика появляются следующие файлы с четырьмя расширениями:

- NeuralNetworkMonitorDriver.inf. Содержит информацию для установки драйвера, с помощью которой система определяет класс, тип, назначение и другие параметры устанавливаемого драйвера,
- NeuralNetworkMonitorDriver.sys. Системный файл, который содержит исходный код драйвера и будет использоваться во время и после его установки,
- NeuralNetworkMonitorDriver.cer. Содержит сертификат безопасности разработанного драйвера, который необходим для установки драйверов в

операционной системе Windows, стоит отметить, что сертификат является неподписанным, поэтому для установки необходимо отключить проверку подписи сертификатов или установить сертификат в систему в качестве доверенного вручную,

– NeuralNetworkMonitorDriver.pdb. Содержит символы и информацию для отладки драйвера и используется при отладке драйвера через приложение Microsoft WinDBG.

Так как сертификат безопасности драйвера является само подписанным необходимо выполнить установку сертификата в хранилище доверенных сертификатов. Для начала установки сертификата необходимо открыть файл сертификата и нажать кнопку “Установить”, после чего появится окно мастера импорта сертификатов, представленное на рисунке 16.

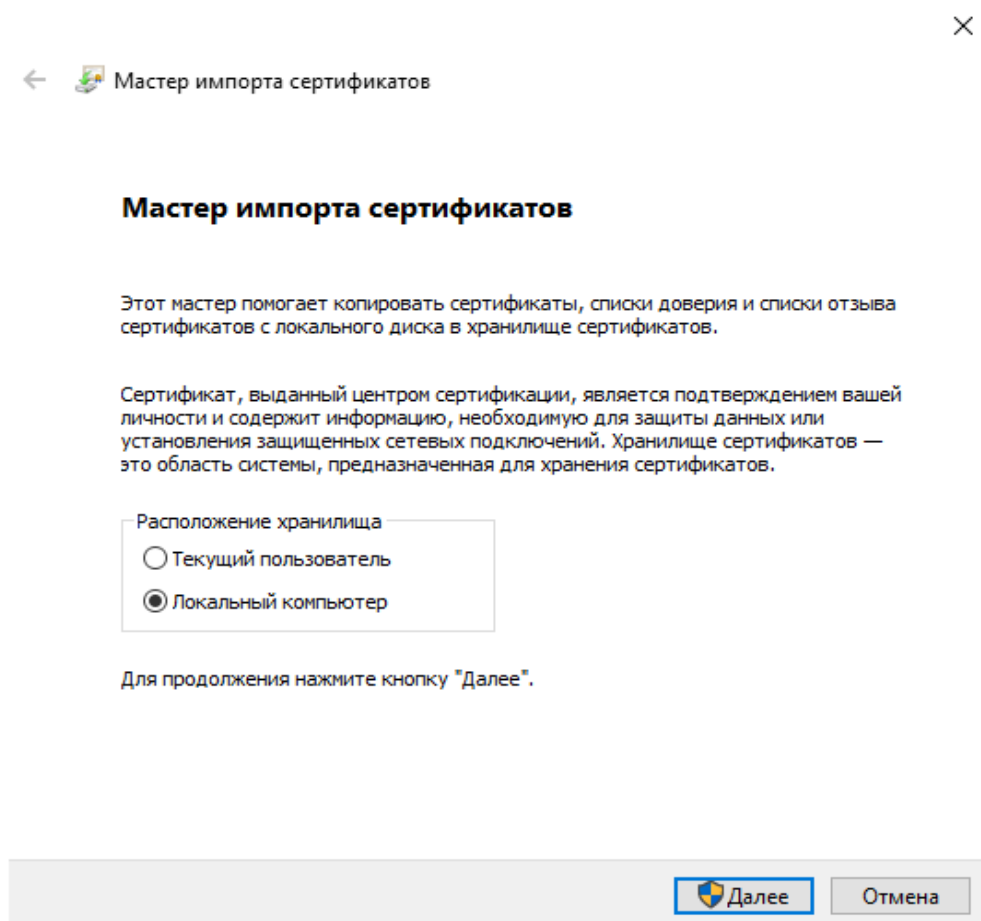


Рисунок 16 – Окно мастера импорта сертификатов

В окне импорта сертификатов необходимо выбрать пункт «Локальный компьютер» и нажать кнопку далее. В новом открывшемся окне необходимо

выбрать пункт «Поместить все сертификаты в следующее хранилище» и в окне поиска хранилищ сертификатов выбрать пункт «Доверенные корневые центры сертификации», после чего нажать кнопку далее и завершить установку сертификата в систему. Вышеописанный процесс отображён на рисунке 17.

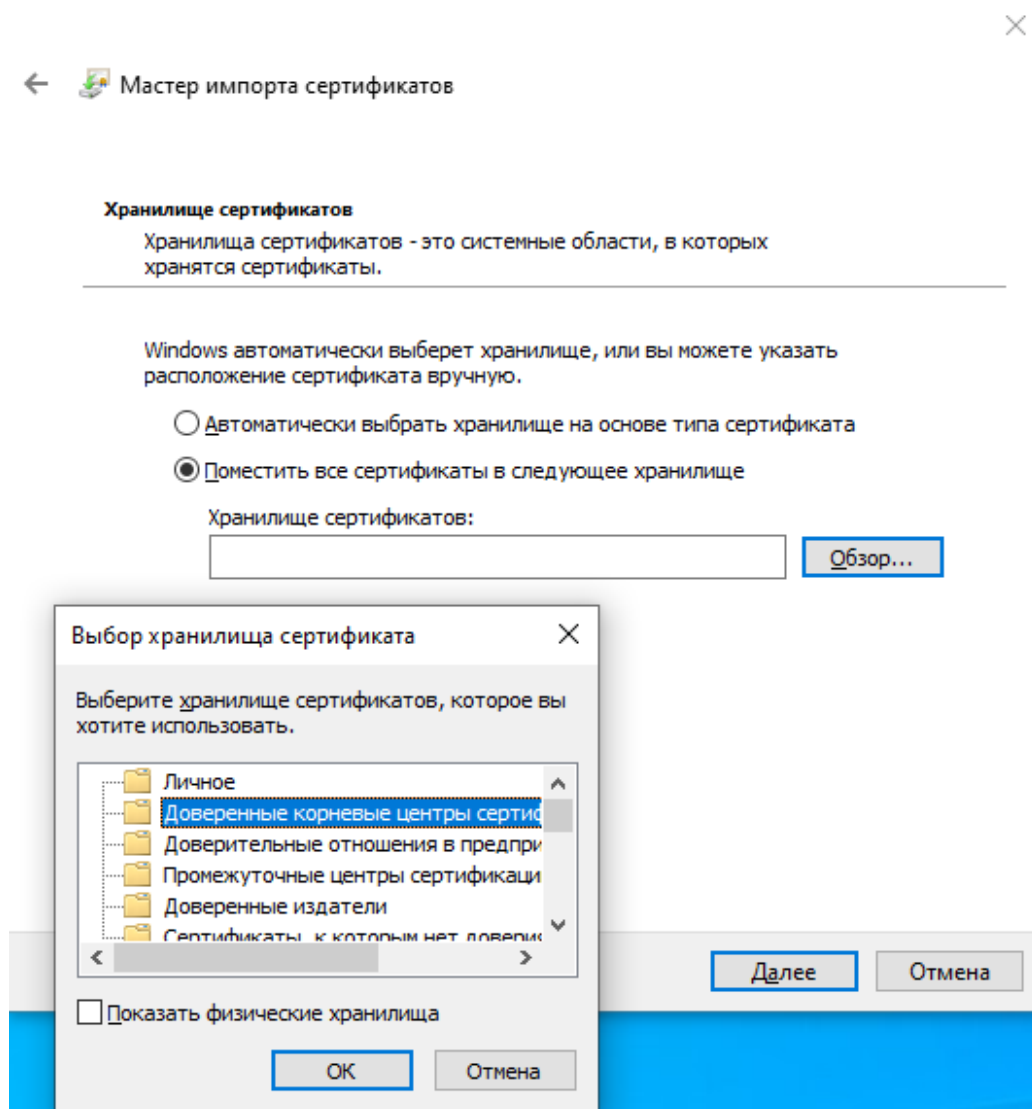


Рисунок 17 – Выбор места хранения сертификата

После выполнения установки сертификата безопасности драйвера можно переходить к установке непосредственно самого драйвера на сетевой адаптер. Для этого необходимо перейти в окно свойств сетевого адаптера, на который необходимо установить разработанный драйвер фильтр, сделать это можно через панель управления или через параметры сети и интернет в настройках Windows, где необходимо выбрать пункт «Настройка параметров адаптера». Окно свойств сетевого адаптера представлено на рисунке 18.

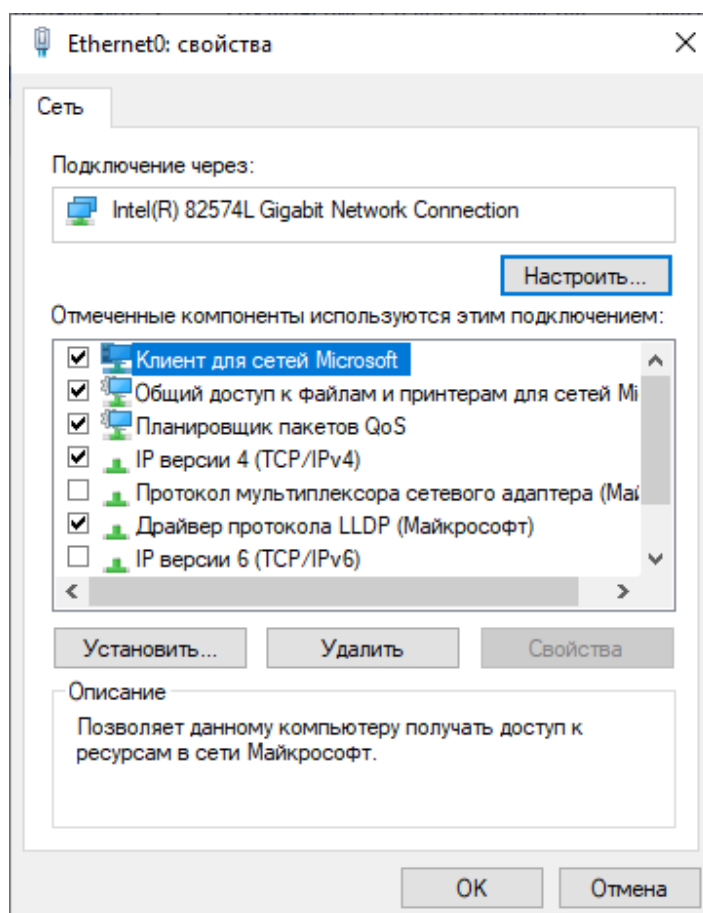


Рисунок 18 – Окно свойств сетевого адаптера

В данном окне необходимо нажать на кнопку установить, после чего в окне выбора сетевых компонентов выбрать пункт «Служба» и нажать кнопку добавить. После чего появится новое окно, в котором необходимо нажать кнопку «Установить с диска» и в окне поиска выбрать inf файл разработанного драйвера. Все вышеописанные действия показаны на рисунке 19.

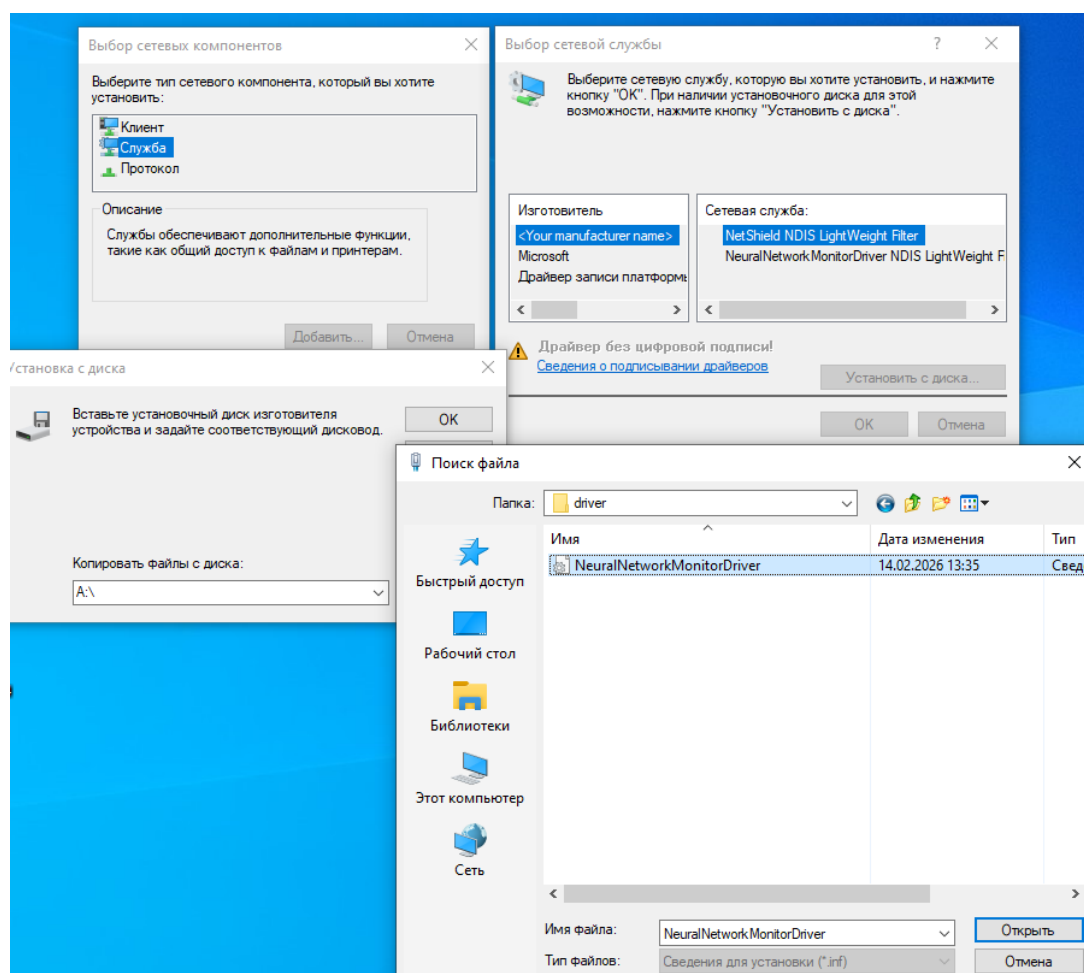


Рисунок 19 – Выбор установочного файла драйвера

После выполнения всех вышеописанных действий, установленный драйвер можно будет обнаружить в окне свойств сетевого адаптера (рисунок 20), что будет свидетельствовать о том, что драйвер был успешно встроен в стек драйверов выбранного сетевого адаптера и готов выполнять поставленные на него задачи.

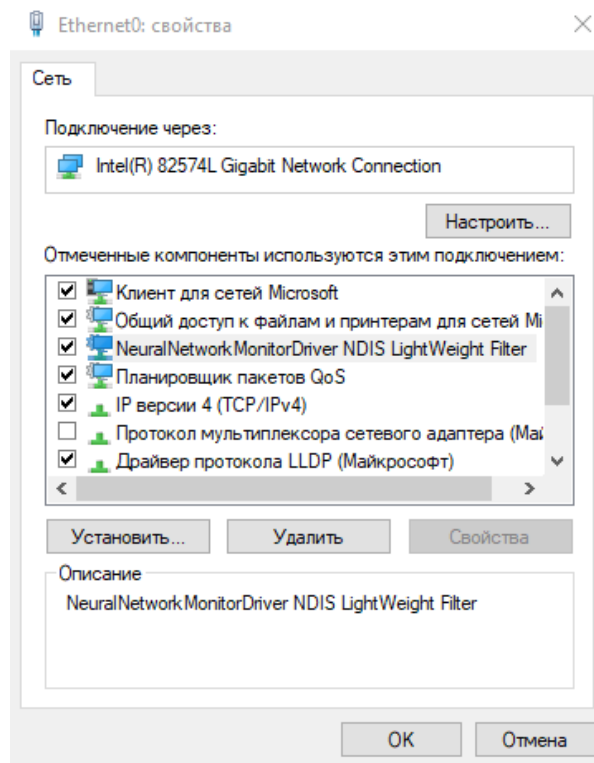


Рисунок 20 – Свойства сетевого адаптера после установки драйвера

4.2 Запуск компонентов 3 кольца

Для полного запуска программного комплекса на 3 кольце необходимо выполнить также запуск python приложения, выполняющего роль сервера, который загружает модель и может принимать запросы для анализа загруженной моделью. Приложение запускается из командной строки, а также имеет возможность выбрать ip адрес и порт, на котором будет запущен сервер с моделью, по умолчанию параметры порта равны 12.0.0.1 и 5555 соответственно. Пример запуска сервера с моделью с выбором ip адреса и порта продемонстрирован на рисунке 21.

```

C:\Windows\System32\cmd.exe - python3 model_server.py --host 192.168.145.139 --port 5556
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.
C:\Users\user\Desktop\NeuralComponent>python3 model_server.py --host 192.168.145.139 --port 5556
2026-05-06 06:49:38 - ModelServer - INFO - Загрузка модели из models\mlp_model.keras
2026-05-06 06:49:38 - ModelServer - INFO - Модель Keras успешно загружена
2026-05-06 06:49:38 - ModelServer - INFO - Загрузка scaler из models\mlp_scaler.pkl
2026-05-06 06:49:38 - ModelServer - INFO - Scaler загружен
2026-05-06 06:49:38 - ModelServer - INFO - Мappings классов загружен: {0: 'benign', 1: 'ddos', 2: 'dos goldeneye',
3: 'dos hulk', 4: 'ftp-patator', 5: 'portscan'}
2026-05-06 06:49:38 - ModelServer - INFO - Сервер запущен на 192.168.145.139:5556
2026-05-06 06:49:38 - ModelServer - INFO - Ожидание подключений...

```

Рисунок 21 – Запуск сервера с моделью нейронной сети

После запуска сервер готов получать векторы признаков сетевого потока анализировать его при помощи модели и возвращать полученный результат

анализа. Следующим и последним шагом запуска программного комплекса является запуск графического интерфейса, который будет получать перехваченный сетевой трафик от драйвера, заниматься подготовкой и передачей информации о сетевых потоках в нейронную сеть для анализа. Первым этапом запуска необходимо выбрать сетевой адаптер из списка доступных, с которого будет осуществляться перехват, подробнее процесс выбора отображён на рисунке 22.

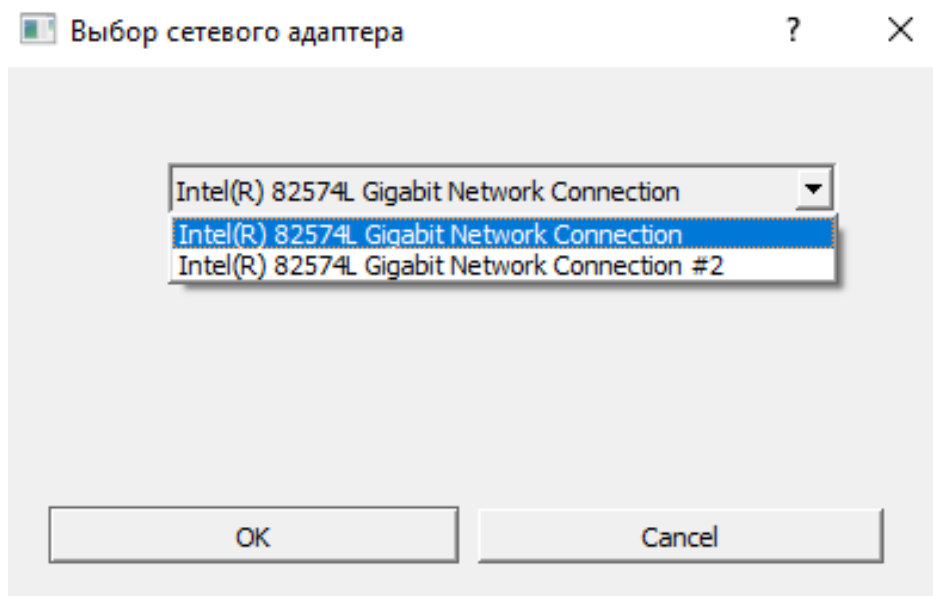


Рисунок 22 – Выбор сетевого адаптера для анализа сетевого трафика

Стоит отметить, что при необходимости можно анализировать сетевой трафик сразу с нескольких сетевых адаптеров, для этого необходимо запустить ещё один экземпляр программы. После выбора сетевого адаптера откроется основное окно программы, отображённое на рисунке 24, в котором в табличном виде будут представлены все сетевые потоки с ip и mac адресами сторон передачи сетевого трафика, а также переданное количество пакетов. В случае если нейронная сеть обнаружит что сетевой поток является вредоносным он будет окрашен в красный цвет. Также при запуске приложения происходит подключение к серверу модели что можно увидеть из её логов на рисунке 23

```

2026-05-06 06:49:38 - ModelServer - INFO - Scaler загружен
2026-05-06 06:49:38 - ModelServer - INFO - Мappings классов загружен: {0: 'benign', 1: 'ddos', 2: 'dos goldeneye',
3: 'dos hulk', 4: 'ftp-patator', 5: 'portscan'}
2026-05-06 06:49:38 - ModelServer - INFO - Сервер запущен на 192.168.145.139:5556
2026-05-06 06:49:38 - ModelServer - INFO - Ожидание подключений...
2026-05-06 07:10:11 - ModelServer - INFO - Клиент подключен: ('192.168.145.139', 53126)

```

Рисунок 23 – Сообщение о подключении к серверу модели

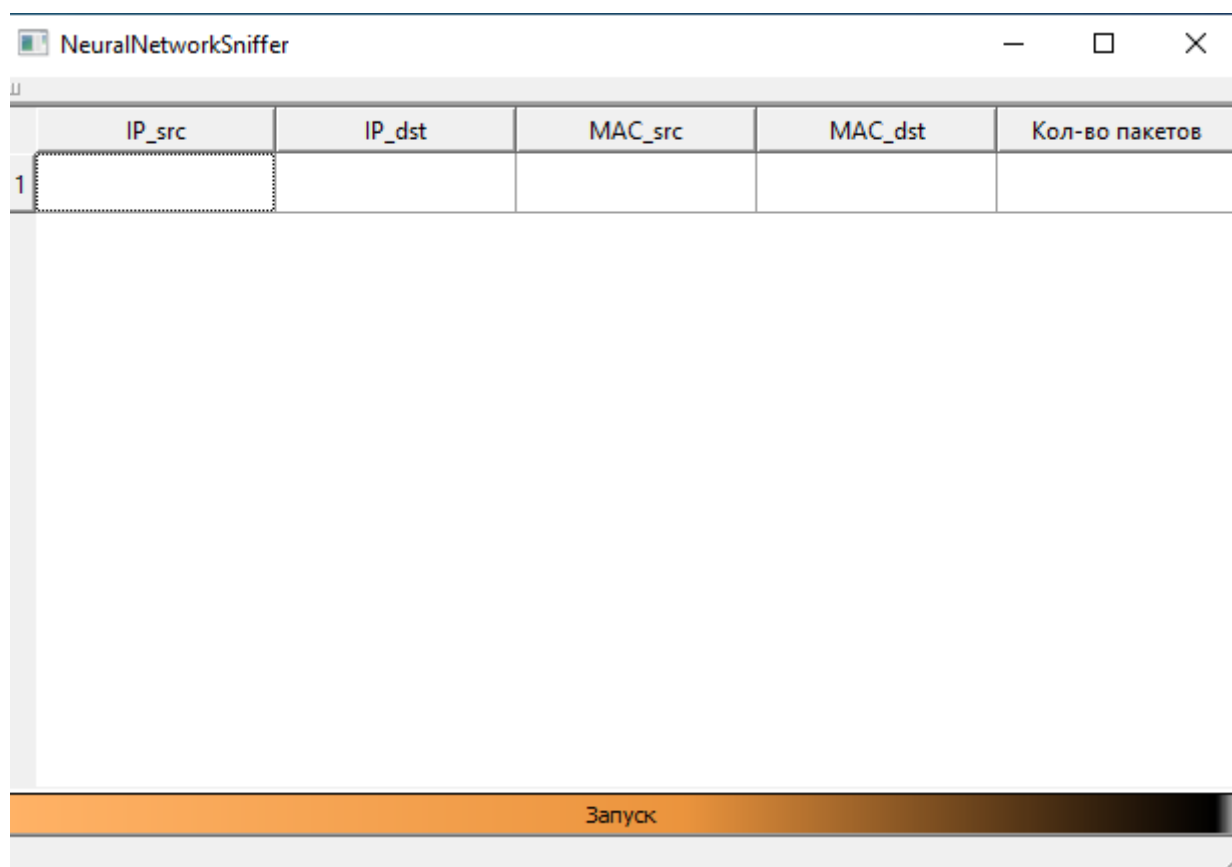


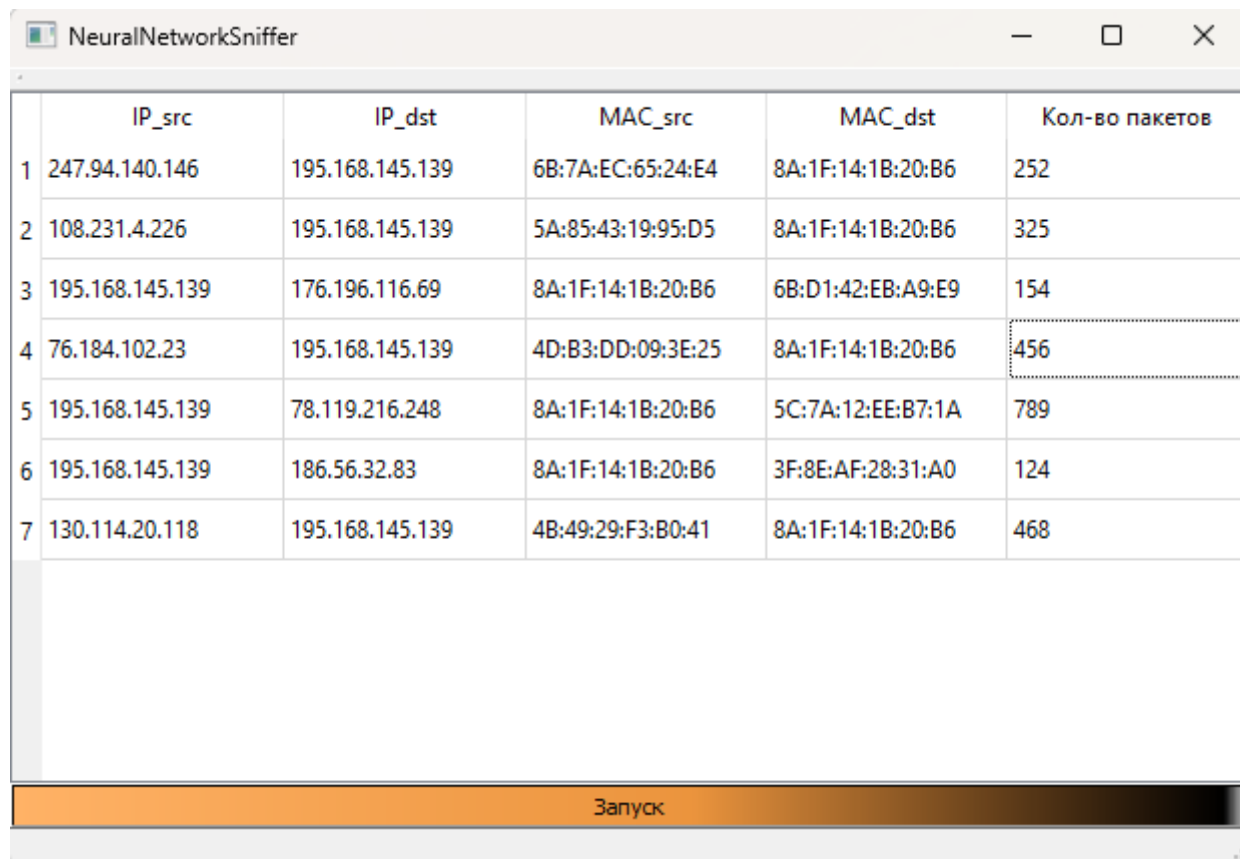
Рисунок 24 – Основное окно графического интерфейса

После запуска графического интерфейса можно считать, что программный комплекс полностью запущен и готов к анализу сетевого трафика.

4.3 Тестирование программного комплекса

Тестирование программного комплекса заключается в простой проверке его работы в открытой сети интернет, а также при контролируемой сетевой атаке. Основной целью данного теста является демонстрация работы и поведения программного комплекса в предполагаемых условиях его использования, а также проверка работы механизмов перехвата сетевого трафика и формирование из него сетевых потоков соединений. На рисунке 25 представлен результат работы программного комплекса при открытии доступа виртуальной машине, на которой он был установлен, в сеть интернет. Как

можно заметить с виртуальной машиной с адресом 195.168.145.139 было установлено несколько сетевых потоков, в котором инициатором иногда выступала наша виртуальная машина, а иногда внешние хосты, при этом сетевых атак не было обнаружено.



	IP_src	IP_dst	MAC_src	MAC_dst	Кол-во пакетов
1	247.94.140.146	195.168.145.139	6B:7A:EC:65:24:E4	8A:1F:14:1B:20:B6	252
2	108.231.4.226	195.168.145.139	5A:85:43:19:95:D5	8A:1F:14:1B:20:B6	325
3	195.168.145.139	176.196.116.69	8A:1F:14:1B:20:B6	6B:D1:42:EB:A9:E9	154
4	76.184.102.23	195.168.145.139	4D:B3:DD:09:3E:25	8A:1F:14:1B:20:B6	456
5	195.168.145.139	78.119.216.248	8A:1F:14:1B:20:B6	5C:7A:12:EE:B7:1A	789
6	195.168.145.139	186.56.32.83	8A:1F:14:1B:20:B6	3F:8E:AF:28:31:A0	124
7	130.114.20.118	195.168.145.139	4B:49:29:F3:B0:41	8A:1F:14:1B:20:B6	468

Запуск

Рисунок 25 – Результат работы программного комплекса

Далее отключим виртуальную машину от сети интернет и проведём контролируемую сетевую атаку со второй виртуальной машины с адресом 195.168.145.146 при помощи программы NCAP. На рисунке 26 представлен фрагмент логов работы сервера модели.

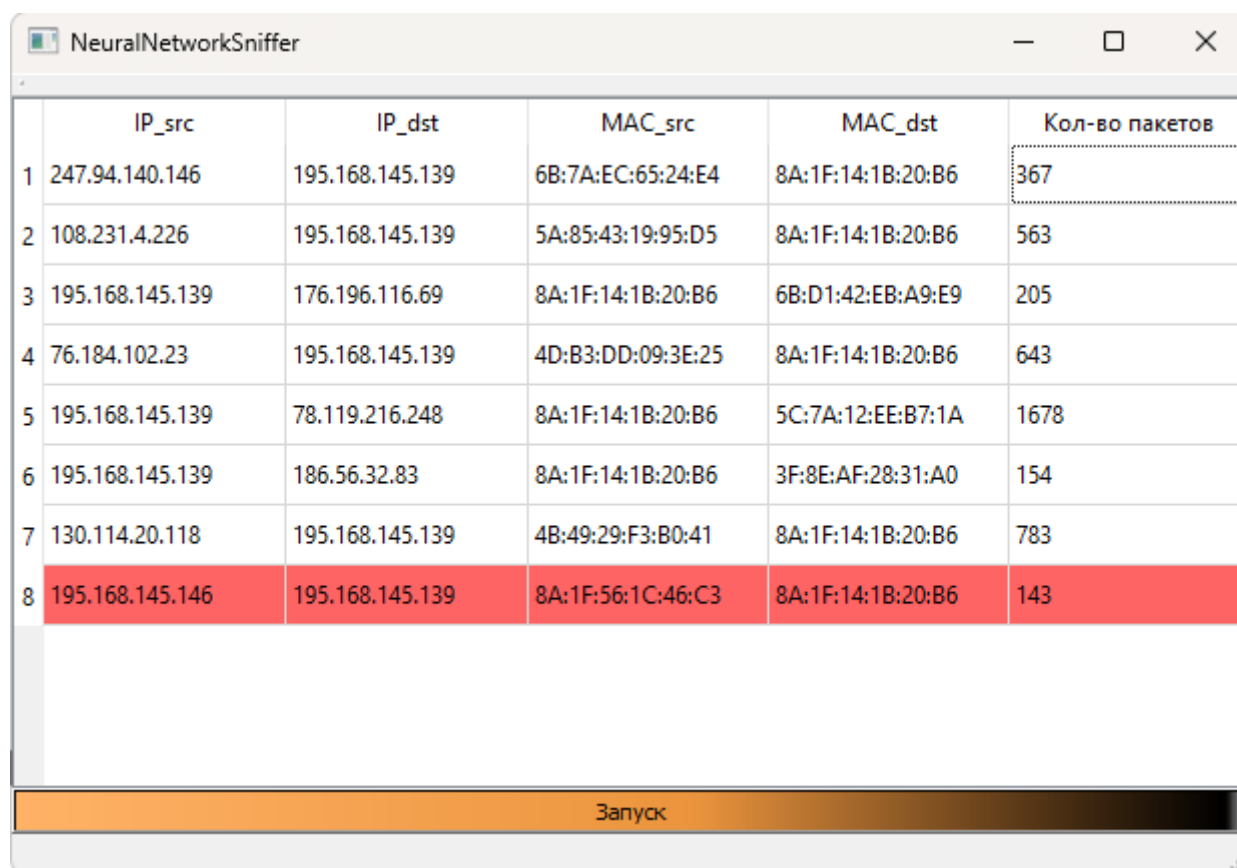
```

2026-05-06 07:41:49 - ModelServer - INFO - Запрос от ('192.168.145.139', 51669): класс=benign, уверенность=1.0000
2026-05-06 07:41:50 - ModelServer - INFO - Запрос от ('192.168.145.139', 51669): класс=benign, уверенность=0.9996
2026-05-06 07:41:50 - ModelServer - INFO - Запрос от ('192.168.145.139', 51669): класс=benign, уверенность=1.0000
2026-05-06 07:41:50 - ModelServer - INFO - Запрос от ('192.168.145.139', 51669): класс=benign, уверенность=0.9738
2026-05-06 07:41:51 - ModelServer - INFO - Запрос от ('192.168.145.139', 51669): класс=benign, уверенность=1.0000
2026-05-06 07:41:51 - ModelServer - INFO - Запрос от ('192.168.145.139', 51669): класс=portscan, уверенность=0.9996
2026-05-06 07:41:51 - ModelServer - INFO - Запрос от ('192.168.145.139', 51669): класс=benign, уверенность=1.0000

```

Рисунок 26 – Логи сервера модели в момент атаки

На рисунке 27 продемонстрирована реакция программного комплекса на сетевую атаку сканирования портов (portscan), как можно заметить атака была быстро обнаружена и отмечена как потенциально опасная, а значит программный комплекс справился с поставленной задачей.



	IP_src	IP_dst	MAC_src	MAC_dst	Кол-во пакетов
1	247.94.140.146	195.168.145.139	6B:7A:EC:65:24:E4	8A:1F:14:1B:20:B6	367
2	108.231.4.226	195.168.145.139	5A:85:43:19:95:D5	8A:1F:14:1B:20:B6	563
3	195.168.145.139	176.196.116.69	8A:1F:14:1B:20:B6	6B:D1:42:EB:A9:E9	205
4	76.184.102.23	195.168.145.139	4D:B3:DD:09:3E:25	8A:1F:14:1B:20:B6	643
5	195.168.145.139	78.119.216.248	8A:1F:14:1B:20:B6	5C:7A:12:EE:B7:1A	1678
6	195.168.145.139	186.56.32.83	8A:1F:14:1B:20:B6	3F:8E:AF:28:31:A0	154
7	130.114.20.118	195.168.145.139	4B:49:29:F3:B0:41	8A:1F:14:1B:20:B6	783
8	195.168.145.146	195.168.145.139	8A:1F:56:1C:46:C3	8A:1F:14:1B:20:B6	143

Запуск

Рисунок 27 – Тестирование обнаружения сетевой атаки

4.4 Тестирование пропускной способности

В данном пункте представлено тестирование пропускной способности сети в системе до и после включения программного комплекса. Тестирование пропускной способности проводится при помощи программы Iperf3 при помощи функции отправки TCP трафика в течении 10 секунд. Настройки отправки потока сетевого трафика выставлены так, чтобы они никак не ограничивали пропускную способность сети и не ограничивали её ресурсы. На рисунке 28 представлен результат отправки сетевого трафика в течении 20 секунд с выключенной подсистемой.

```

C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 0
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49683 connected to 192.168.0.26 port 5201
[ ID] Interval      Transfer    Bandwidth
[ 4]  0.00-1.00  sec    533 MBytes  4.47 Gbits/sec
[ 4]  1.00-2.00  sec    573 MBytes  4.81 Gbits/sec
[ 4]  2.00-3.00  sec    575 MBytes  4.82 Gbits/sec
[ 4]  3.00-4.00  sec    578 MBytes  4.85 Gbits/sec
[ 4]  4.00-5.00  sec    582 MBytes  4.88 Gbits/sec
[ 4]  5.00-6.00  sec    575 MBytes  4.82 Gbits/sec
[ 4]  6.00-7.00  sec    586 MBytes  4.92 Gbits/sec
[ 4]  7.00-8.00  sec    560 MBytes  4.69 Gbits/sec
[ 4]  8.00-9.00  sec    584 MBytes  4.90 Gbits/sec
[ 4]  9.00-10.00 sec    574 MBytes  4.82 Gbits/sec
-----
[ ID] Interval      Transfer    Bandwidth
[ 4]  0.00-10.00 sec   5.59 GBytes  4.80 Gbits/sec
[ 4]  0.00-10.00 sec   5.59 GBytes  4.80 Gbits/sec
sender
receiver

```

Рисунок 28 – Результат тестирования скорости с выключенным шифрованием

Как можно заметить средняя скорость отправки и получения сетевого трафика одинакова и составляет 4.8 Гигабит в секунду. Включим разработанный программный комплекс и проверим пропускную способность сети. Результат тестирования продемонстрирован на рисунке 29.

```

C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 0
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49687 connected to 192.168.0.26 port 5201
[ ID] Interval      Transfer    Bandwidth
[ 4]  0.00-1.00  sec    276 MBytes  2.30 Gbits/sec
[ 4]  1.00-2.00  sec    281 MBytes  2.36 Gbits/sec
[ 4]  2.00-3.00  sec    286 MBytes  2.40 Gbits/sec
[ 4]  3.00-4.00  sec    279 MBytes  2.34 Gbits/sec
[ 4]  4.00-5.00  sec    287 MBytes  2.41 Gbits/sec
[ 4]  5.00-6.00  sec    283 MBytes  2.38 Gbits/sec
[ 4]  6.00-7.00  sec    284 MBytes  2.38 Gbits/sec
[ 4]  7.00-8.00  sec    285 MBytes  2.39 Gbits/sec
[ 4]  8.00-9.00  sec    286 MBytes  2.40 Gbits/sec
[ 4]  9.00-10.00 sec    277 MBytes  2.32 Gbits/sec
-----
[ ID] Interval      Transfer    Bandwidth
[ 4]  0.00-10.00 sec   2.76 GBytes  2.37 Gbits/sec
[ 4]  0.00-10.00 sec   2.76 GBytes  2.37 Gbits/sec
sender
receiver
iperf Done.

```

Рисунок 29 – Результат тестирования пропускной способности со включенным шифрованием

Из данного теста можно увидеть, что, пропуская способность сети при включенном программном комплексе уменьшилась в два раза что, однако не означает что урезание скорости всегда будет двухкратным, оно заметно из-за изначально высокой скорости передачи данных и увеличивается вместе с ней. Программно уменьшив максимальную пропускную способность канала

проведём повторный тест. Далее на рисунках 30 и 31 представлены результаты тестирования с программно ограниченной пропускной способностью.

```
C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 200000000
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49707 connected to 192.168.0.26 port 5201
[ ID] Interval      Transfer    Bandwidth
[ 4]  0.00-1.00    sec  22.2 MBytes  187 Mb/s
[ 4]  1.00-2.00    sec  24.1 MBytes  202 Mb/s
[ 4]  2.00-3.02    sec  23.1 MBytes  191 Mb/s
[ 4]  3.02-4.02    sec  23.9 MBytes  200 Mb/s
[ 4]  4.02-5.00    sec  24.5 MBytes  209 Mb/s
[ 4]  5.00-6.02    sec  23.2 MBytes  192 Mb/s
[ 4]  6.02-7.00    sec  24.2 MBytes  207 Mb/s
[ 4]  7.00-8.02    sec  23.4 MBytes  193 Mb/s
[ 4]  8.02-9.02    sec  24.9 MBytes  209 Mb/s
[ 4]  9.02-10.00   sec  23.4 MBytes  199 Mb/s
-----
[ ID] Interval      Transfer    Bandwidth
[ 4]  0.00-10.00   sec  237 MBytes  199 Mb/s      sender
[ 4]  0.00-10.00   sec  237 MBytes  199 Mb/s      receiver
iperf Done.
```

Рисунок 30 – Результат тестирования скорости с выключенным шифрованием с ограничением пропускной способности

```
C:\Users\TixAn\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 200000000
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49739 connected to 192.168.0.26 port 5201
[ ID] Interval      Transfer    Bandwidth
[ 4]  0.00-1.00    sec  22.0 MBytes  184 Mb/s
[ 4]  1.00-2.02    sec  23.6 MBytes  195 Mb/s
[ 4]  2.02-3.02    sec  24.5 MBytes  206 Mb/s
[ 4]  3.02-4.00    sec  23.4 MBytes  199 Mb/s
[ 4]  4.00-5.00    sec  23.9 MBytes  200 Mb/s
[ 4]  5.00-6.02    sec  23.6 MBytes  195 Mb/s
[ 4]  6.02-7.00    sec  24.5 MBytes  209 Mb/s
[ 4]  7.00-8.00    sec  23.6 MBytes  198 Mb/s
[ 4]  8.00-9.00    sec  23.6 MBytes  198 Mb/s
[ 4]  9.02-10.00   sec  23.4 MBytes  199 Mb/s
-----
[ ID] Interval      Transfer    Bandwidth
[ 4]  0.00-10.00   sec  237 MBytes  199 Mb/s      sender
[ 4]  0.00-10.00   sec  237 MBytes  199 Mb/s      receiver
iperf Done.
```

Рисунок 31 – Результат тестирования скорости со включенным шифрованием с ограничением пропускной способности

Как можно заметить, в данной ситуации падение скорости передачи данных не было обнаружено, при всё ещё большом объёме передаваемых данных в секунду, что позволяет сделать вывод, что замедление скорости передачи данных при включенном программном комплексе будет более заметно при увеличении пропускной способности сети. Данное замедление приемлемый компромисс повышения безопасности сети.

5 Возможности для улучшения и расширения функционала

Разработанная в рамках данной работы программный комплекс по защите от сетевых атак с помощью использования технологий нейронных сетей также имеет большие возможности и потенциалы для развития и расширения. Развитие подсистемы может включать в себя, как и улучшение или добавление нового функционала или расширение доступности использования разработанного программного комплекса.

В качестве варианта возможного расширения доступности использования программного комплекса может послужить добавление возможности переноса части комплекса, связанной с нейронной сетью, на выделенный сервер, что позволит сильно снизить нагрузку на локальную защищаемую машину, а также позволит расширить доступные ресурсы для нейронной сети и откроет возможности для её расширения и увеличения эффективности. Однако в данном случае важно учитывать и возможные риски сетевых атак при передаче данных от сервера с нейронной сетью до конечной машины, что, однако можно решить несколькими способами:

- использованием защиты самого сервера,
- использованием кольцевой схемы сети с разными уровнями, для разграничения доступа к серверу,
- расположением и обеспечением доступа к серверу с нейронной сетью только из локальной сети.

Ещё одну возможность для развития и расширения функционала открывает модульная архитектура разрабатываемой системы, благодаря которой задача разработки версии системы защиты от сетевых атак на ОС Linux становятся более простой задачей. В последние годы эта операционная система всё больше и больше набирает популярность в секторе серверного и промышленного использования из-за своей легковесности, модульности и простоты в развёртке, вследствие чего перенос системы обнаружения сетевых атак на ОС Linux является отличной возможностью расширения возможного круга использования разработанной системы. Простота переноса при этом

заключается в модульности, т.к. полностью адаптировать необходимо только драйвер, работающий в ядре, из-за различий ОС, при этом нейронная сеть, написанная на python может работать на любой системе с установленной и настроенной средой запуска. Перенос графического интерфейса в свою очередь тоже упрощается, за счёт того, что он написан при помощи библиотек QT, которые являются кроссплатформенными, т.е. для переноса графического интерфейса на ОС Linux необходимо только избавиться от платформозависимых кусков кода, таких, как например работа с файловой системой [17].

Очень полезной и практически значимой функцией, которую можно добавить является возможность дообучения и добавление в нейронную сеть возможности обнаруживать новые атаки. При этом сам процесс дообучения должен проходить без необходимости полного переобучения модели, что позволит эффективно со временем увеличивать количество обнаруживаемых типов сетевых атак, а также позволит постоянно обновлять базу нейронной сети, добавляя самые новые образцы сетевых атак для её актуализации. Дообучение при этом можно выполнять при помощи метода transfer-learning, при котором последний слой нейронной сети срезается и на его место устанавливается новый или при помощи метода конкатенации при котором к последнему слою добавляется ещё дополнительный слой связи.

Также расширению функционала также может быть подвержен и графический интерфейс, т.к. на данном этапе разработанности системы он больше представляет информационное окно, нежели программу управления. Однако расширить возможно компонента графического интерфейса можно добавить различными способами, такими как:

- добавление возможность централизованно просматривать применённые правила фильтрации и включать или выключать, а также добавление возможности загрузки сохранённой конфигурации правил фильтрации,

- добавлением отображения более подробной статистике о сетевом потоке (количество сетевых пакетов, время существования, количество

переданных данных и т.д.),

- добавление возможности выбора режимы работы системы обнаружения и блокирования, в которых нейронная сеть производит анализ сетевого потока и возвращает результат, но при этом команда с добавлением новых правил блокировки может быть отправлена только при по решению пользователя, или автоматическое блокирование подозрительного сетевого потока соответственно,

- добавление возможности просматривать содержимое непосредственно сетевых пакетов, относящихся к сетевому потоку в разобранном в разобранном виде по аналогии с другими известными сниффер программами,

- добавление возможности сохранять файлы с информацией о перехваченных сетевых потоках, для обеспечения возможности их повторного открытия и анализа в дальнейшем.

Таким образом выше озвученные улучшение и дополнения функционала позволят превратить не только в информационное окно, но и в инструмент управления, а также анализа сетевого трафика на уровне с другими сниффер программами.

ЗАКЛЮЧЕНИЕ

В результате выполнения магистерской работы была выполнено исследование возможностей использования нейронных сетей для анализа сетевого трафика на наличие атак. Был произведён анализ существующих наборов данных и архитектур нейронных сетей, а также была разработана модульная система защиты от сетевых атак с эффективным алгоритмом передачи данных с уровня ядра на пользовательский уровень и нейронную сеть. Из полученных результатов можно выделить:

- был разработан эффективный алгоритм перехвата сетевого трафика в режиме реального времени для последующей его обработки нейронными сетями без сильной потери в производительности и скорости соединения, основывающийся на технология разделяемой памяти, кольцевых буферов и пуллинга,

- был разработан компонент NDIS-драйвера фильтра, работающего в ядре системы, осуществляющего перехват и передачу сетевого трафика в приложение графического интерфейса, при помощи разработанного алгоритма передачи данных. Также данный драйвер путём встраивания в стек сетевых драйверов каждого сетевого адаптера в системе позволяет перехватывать сетевой трафик с любого из доступных адаптеров,

- был разработан компонент графического интерфейса сетевого анализатора, который отображает статистику сетевых соединений, результаты анализа нейронной сетью, а также выполняющего роль связующего звена между драйвером и нейронной сетью,

- был выполнен поиск и подготовка данных сетевого трафика для обучения нейронной сети. В результате для обучения нейронной сети был выбран набор данных CIC-IDS-2017, содержащего данные о нескольких классах атак, а также признаки для их обнаружения в формате не отдельных сетевых пакетов, а сетевых потоков,

- был разработан компонент нейронной сети с архитектурой многослойного перцептрона, которая выполняет задачу многоклассовой

классификации сетевого трафика на наличие сетевых атак. Данный компонент получает в формате набора признаков сетевых потоков, которые были собраны напрямую из сетевого стека драйвером и переданы для анализа из компонента графического интерфейса.

Также было проведено комплексное тестирование разработанной системы и проверка эффективности скорости обработки сетевого трафика, погрешности обнаружения вредоносного трафика. В результате тестирования разработанной нейронной сети, обнаруживающей 6 классов сетевых атак, были получены следующие основные результаты: точность модели равняется 98.5%, при этом переобучения не обнаружено, высокий процент точности подтверждается параметрами precision, recall и F1-score, а также фактом того, что модель менее уверена в своих неправильных ответах. Также был проведён анализ скорость обработки, данный нейронной сетью в лабораторных условиях, который показал, что модель способна обрабатывать 2200 образцов сетевых потоков в секунду, что является очень хорошим результатом.

Полное тестирование программного комплекса также доказало свою работоспособность и доказало возможность определения сетевых атак при помощи обученной модели и сетевого трафика, полученного из драйвера перехватчика. Результаты проверки пропускной способности со включенным разработанным комплексом показали хороший результат, который показывает, что падение пропускной способности в большой степени зависит от изначальной и не является постоянной величиной. Так, например на изначальную очень высокой пропускной способности в 4.5 Гбит/сек, пропускная способность упала в два раза, однако при эксперименте с обычной пропускной способностью, использующейся в большинстве пользовательских систем 200-300 Мбит/сек падение совершенно не наблюдалось, что позволяет сделать вывод что программный комплекс позволяет выполнять поставленные на него задачи без сильного замедления системы.

В перспективе данная работа может быть полезна исследователям

систем сетевой безопасности или для разработчиков систем повышения сетевой безопасности с использованием нейронных сетей. При этом разработанный программный комплекс имеет большое количество векторов развития и улучшений, а также может при должной доработке использоваться в различных коммерческих или муниципальных учреждениях (например, в университетах) для повышения защиты сетевой инфраструктуры от различных сетевых атак. Однако для дальнейшей разработки для имплементации в различные учреждения и для коммерческого использования разработанной системы необходимо будет получить соответствующие сертификаты безопасности, а также пройти государственную сертификацию.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хакеры взломали очистные сооружения города в США и попытались отравить воду для 15 тысяч жителей [Электронный ресурс] – URL: <https://habr.com/ru/news/541514/> (дата обращения: 08.01.2026).
2. Денисенко А.А., Губарь А.Ю. Анализ сетевого трафика с использованием нейронных сетей // Международный научно-исследовательский журнал. – 2023. – № 3(129). – С. 6–10. – DOI: 10.23670/IRJ.2023.129.18.
3. Ахмедов А.Э., Кузнецов А.А. Применение глубокого обучения для классификации сетевого трафика // Труды СПИИРАН. – 2020. – Т. 19, № 4. – С. 812–836. – DOI: 10.15622/sp.2020.19.4.5.
4. Кузнецов С.Д., Иванов И.А. Анализ эффективности сверточных нейронных сетей для детектирования атак в сетевом трафике // Кибербезопасность и защита информации. – 2022. – Т. 10, № 2. – С. 112–125. – DOI: 10.26583/csi.2022.2.09.
5. Мисаилиди А. А. Автоматизация процесса дообучения нейронной сети и его перенос на сторону клиента [Электронный ресурс] – URL: <https://na-journal.ru/6-2023-informacionnye-tekhnologii/5922-avtomatizaciya-processa-doobucheniya-neironnoi-seti-i-ego-perenos-na-storonu-klienta> (дата обращения: 12.10.2025).
6. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 5-е изд. – СПб.: Питер, 2016. – 992 с.: ил. – (Серия «Учебник для вузов»).
7. Создание надежных драйверов Kernel-Mode. [Электронный ресурс] – URL: <https://learn.microsoft.com/ru-ru/windows-hardware/drivers/kernel/creating-reliable-kernel-mode-drivers>, свободный. (дата обращения 25.12.2024)
8. Sufyan T. Faraj Al-Janab, Muayad Kod On the Development of NDIS Intermediate Driver-Based Virtual Private Networks. - Conference: the Third International Scientific Conference of Salahaddin University-Erbil, 2011. – 5p.

9. Русинович Марк, Соломон Дэвид Внутреннее устройство Microsoft Windows. Основные подсистемы ОС. 6-е изд. — М.: ДМК Пресс, 2018. — 944 с.: ил. — ISBN 978-5-97060-628-1
10. Порошин, И. Е. Применение графических процессоров с технологией CUDA в высокопроизводительных системах с искусственными нейронными сетями / И. Е. Порошин, М. А. Титов. — Текст: непосредственный // Молодой ученый. — 2020. — № 25 (315). — С. 123-125
11. Тихонов А. А. Способ обработки интернет-трафика на базе спецификаций интерфейса сетевого драйвера / Проблемы информатики в образовании, управлении, экономике и технике: сборник статей XXIV Международной научно-технической конференции — Пензенский государственный университет, 2024. — С. 70–75.
12. Основы нейросетевых архитектур [Электронный ресурс] — URL: <https://deeplearning.ru/docs/Neural-networks/Multilayer-perceptron> (дата обращения: 08.4.2026).
13. Аггарвал, Ч. Нейронные сети и глубокое обучение: учебный курс / Ч. Аггарвал ; пер. с англ. — СПб. : ООО «Диалектика», 2020. — 752 с.
14. Жерон О. Прикладное машинное обучение с помощью Scikit-Learning и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем. — СПб.: ООО "Альфа-книга", 2018. — 688 с.
15. Waldo J. A comparative study of back propagation and its alternatives on multilayer perceptrons [Электронный ресурс] — URL: <https://arxiv.org/abs/2206.06098> (дата обращения: 14.04.2026).
16. Метрики в задачах машинного обучения. [Электронный ресурс] — URL: <https://habr.com/ru/company/ods/blog/328372/> (дата обращения: 23.12.2025).
17. Гордеев А.В. Операционные системы: Учебник для вузов. — 2-е изд. — СПб.: Питер, 2009. — 416 с.: ил. — (Серия «Учебник для вузов»).

ПРИЛОЖЕНИЕ А

(обязательное)

Программный анализатор сетевого трафика

Листинг программы

Файл PacketTransfer.c

```
#include "precomp.h"
#include "PacketTransfer.h"
#include "MdlDataOperations.h"
#include <ntstrsafe.h>

double _fltused;

double ln(double x) {
    if (x <= 0.0) return -1000000.0;

    union {
        double d;
        unsigned long long l;
    } u = { x };

    int exponent = ((u.l >> 52) & 0x7FF) - 1023;

    u.l = (u.l & 0xFFFFFFFFFFFFFFFFULL) | 0x3FF0000000000000ULL;
    double mantissa = u.d;

    double y = mantissa - 1.0;
    double p = y / (2.0 + y);
    double p2 = p * p;

    double ln_mantissa = y - p * (y - 0.5 * p2 *
(1.3333333333333333 - 0.8 * p2));

    return ln_mantissa + exponent * 0.6931471805599453;
}

PVOID CreateSharedSection(UNICODE_STRING SectionName,
LARGE_INTEGER MaxSize, _Out_ HANDLE* Handle) {

    HANDLE SectionHandle;
    OBJECT_ATTRIBUTES SectionAttributes;
    NTSTATUS status;

    InitializeObjectAttributes(&SectionAttributes,
&SectionName,
        OBJ_CASE_INSENSITIVE, NULL, NULL);

    status = ZwCreateSection(Handle, SECTION_ALL_ACCESS,
        &SectionAttributes, &MaxSize, PAGE_READWRITE,
SEC_COMMIT, NULL);

    PVOID Map = NULL;

    if (!NT_SUCCESS(status))
        return NULL;

    SIZE_T ViewSize = 0;

    status = ZwMapViewOfSection(*Handle, NtCurrentProcess(),
&Map, 0,
        MaxSize.QuadPart, NULL, &ViewSize, ViewShare, 0,
PAGE_READWRITE);
```

```

        SHARED_MEMORY_INFO SectionInfo;
        SectionInfo.MemorySize = MaxSize;
        SectionInfo.MemoryBuffer = (CHAR*)Map +
sizeof(SHARED_MEMORY_INFO);
        SectionInfo.Head_ptr = (CHAR*)Map +
sizeof(SHARED_MEMORY_INFO);
        SectionInfo.Tail_ptr = (CHAR*)Map +
sizeof(SHARED_MEMORY_INFO);
        SectionInfo.Head = 0;
        SectionInfo.Tail = 0;
        SectionInfo.WritePointersProtect = 0;

        ULONG sizeMask = ln(SectionInfo.MemorySize.QuadPart) /
ln(2);
        sizeMask = (sizeMask << 1) - 1;
        SectionInfo.SizeMask = sizeMask;

        NdisMoveToMappedMemory(Map, &SectionInfo,
sizeof(SHARED_MEMORY_INFO));

        if (!NT_SUCCESS(status)) {
            ZwClose(*Handle);
            return NULL;
        }

        return Map;
    }

GUID GenerateUniqueNameByGUID(UNICODE_STRING* String) {

    GUID uGuid;
    NTSTATUS status = STATUS_SUCCESS;

    status = ExUuidCreate(&uGuid);

    RtlUnicodeStringPrintf(String, L"%08lX-%04X-%04X-%02X%02X-
%02X%02X%02X%02X%02X%02X",
        uGuid.Data1, uGuid.Data2, uGuid.Data3,
uGuid.Data4[0], uGuid.Data4[1], uGuid.Data4[2],
        uGuid.Data4[3], uGuid.Data4[4], uGuid.Data4[5],
        uGuid.Data4[6], uGuid.Data4[7]);

    return uGuid;
}

//FALSE ãíîè ìàò, TRUE ãíîè àñòù
UINT CheckForRace(SHARED_MEMORY_INFO* SharedMemInfo, UINT
PacketSize) {

    while (InterlockedCompareExchange(&SharedMemInfo-
>WritePointersProtect, 1, 0) == 1) {
        NdisMSleep(20);
    }

    if (((CHAR*)SharedMemInfo->Head_ptr && SharedMemInfo-
>SizeMask) < ((CHAR*)SharedMemInfo->Tail_ptr && SharedMemInfo-
>SizeMask)) {

```

```

        InterlockedExchange(&SharedMemInfo->WritePointersProtect, 0);
        return FALSE;
    }
    else if (((CHAR*)SharedMemInfo->Head_ptr && SharedMemInfo->SizeMask) > ((CHAR*)SharedMemInfo->Tail_ptr && SharedMemInfo->SizeMask)) {

        ULONG TailHeadSpace = (ULONG)((CHAR*)SharedMemInfo->Head_ptr && SharedMemInfo->SizeMask)
            - (ULONG)((CHAR*)SharedMemInfo->Tail_ptr && SharedMemInfo->SizeMask);

        InterlockedExchange(&SharedMemInfo->WritePointersProtect, 0);
        return TailHeadSpace < PacketSize;
    }
}

UINT GetSharedMemoryFreeSpace(SHARED_MEMORY_INFO* SharedMemInfo) {

    ULONG currentSize = (CHAR*)SharedMemInfo->Tail_ptr && SharedMemInfo->SizeMask;

    return SharedMemInfo->MemorySize.QuadPart - currentSize;
}

void SharePacket(NET_BUFFER* packet, SHARED_MEMORY_INFO* SharedMemInfo) {

    UINT packetOffset = 0;
    UINT packetSize = GetNetBufferDataSize(packet);
    LARGE_INTEGER DataSize;

    // Ĩđĩăăđŷăĩ ĩàĩŷòü ĩă ĩăđăĩĩěĩăĩěă, âñěě ĩĩĩ âñòü
    ñăđăñûăăăĩ óêàçàòăëü ôăĩñòà ĩă ĩă÷ăëĩ ĩàĩŷòë
    if (packetSize > GetSharedMemoryFreeSpace(SharedMemInfo)) {

        while (InterlockedCompareExchange(&SharedMemInfo->WritePointersProtect, 1, 0) == 1) {
            NdisMSleep(20);
        }

        ULONG FreeSpaceAtStart =
            (ULONG)((CHAR*)SharedMemInfo->Head_ptr && SharedMemInfo->SizeMask)
            - (ULONG)((CHAR*)SharedMemInfo->MemoryBuffer && SharedMemInfo->SizeMask);
        if (FreeSpaceAtStart > packetSize) { // ĩă÷ăëĩ
            ĩàĩŷòë ñăĩăĩăĩĩ, ĩĩăĩĩ ĩđĩăĩěăòü çăĩěñü
            SharedMemInfo->Tail_ptr = SharedMemInfo->MemoryBuffer;
        }
        else { // Æă,ĩ ĩĩêă ĩñăĩăĩăŷòñŷ äăĩĩûă â ĩă÷ăëă
            ĩàĩŷòë ÷òĩăũ ĩđĩăĩěăèü ĩăđăăă÷ó

```

```

        InterlockedExchange(&SharedMemInfo-
>WritePointersProtect, 0);

        while (TRUE) {

            while
(InterlockedCompareExchange(&SharedMemInfo->WritePointersProtect,
1, 0) == 1) {

                NdisMSleep(20);

            }

            ULONG FreeSpaceAtStart =
(ULONG)((CHAR*)SharedMemInfo->Head_ptr && SharedMemInfo->SizeMask)
            - (ULONG)((CHAR*)SharedMemInfo-
>MemoryBuffer && SharedMemInfo->SizeMask);
            if (FreeSpaceAtStart > packetSize) {
                SharedMemInfo->Tail_ptr =
SharedMemInfo->MemoryBuffer;
                break;
            }

            InterlockedExchange(&SharedMemInfo-
>WritePointersProtect, 0);
        }

        InterlockedExchange(&SharedMemInfo-
>WritePointersProtect, 0);
    }

    // Ĩöîââöÿâî îà îàëè÷èâ âîîèè
    if (CheckForRace(SharedMemInfo, packetSize)) {
        while (TRUE)
            if (!CheckForRace(SharedMemInfo, packetSize))
                break;
    }

    //Çàîèñü îàêâòà â îàìÿòü
    DataSize.QuadPart = packetSize;
    NdisMoveToMappedMemory(SharedMemInfo->Tail_ptr, &DataSize,
sizeof(LARGE_INTEGER));

    while (packetOffset < packetSize) {

        UINT dataInPointer = 0;
        void* data = GetData(packet, packetOffset,
&dataInPointer);

        NdisMoveToMappedMemory(((CHAR* )SharedMemInfo-
>Tail_ptr + sizeof(LARGE_INTEGER)
            + packetOffset), data, dataInPointer);

        packetOffset += dataInPointer;
    }

    while (InterlockedCompareExchange(&SharedMemInfo-
>WritePointersProtect, 1, 0) == 1) {
        NdisMSleep(20);
    }

```

```

        SharedMemInfo->Tail_ptr = (CHAR*)SharedMemInfo->Tail_ptr +
sizeof(LARGE_INTEGER) + packetOffset;
        SharedMemInfo->Tail++;

        InterlockedExchange(&SharedMemInfo->WritePointersProtect,
0);
}

```

Файл IODevice.c

```

#include <ntddk.h>
#include "precomp.h"
#include "PacketTransfer.h"
#include "NNMDefines.h"
#include <ntstrsafe.h>
#define __FILENUMBER 'CNPF'

_IRQL_requires_max_(PASSIVE_LEVEL)
NDIS_STATUS
NeuralNetworkMonitorDriverRegisterDevice(PDRIVER_OBJECT
FilterDriverObject,
        NDIS_HANDLE FilterDriverHandle, NDIS_HANDLE*
FilterContextDevice, PMS_FILTER Context){

        NDIS_STATUS                Status = NDIS_STATUS_SUCCESS;
        PDRIVER_DISPATCH
DispatchTable[IRP_MJ_MAXIMUM_FUNCTION+1];
        NDIS_DEVICE_OBJECT_ATTRIBUTES DeviceAttribute;
        PFILTER_DEVICE_EXTENSION      FilterDeviceExtension;
        PDRIVER_OBJECT                DriverObject;
        PDEVICE_OBJECT NdisDeviceObject = NULL;

        DEBUGP(DL_TRACE,
"==>NeuralNetworkMonitorDriverRegisterDevice\n");

        NdisZeroMemory(DispatchTable, (IRP_MJ_MAXIMUM_FUNCTION+1) *
sizeof(PDRIVER_DISPATCH));

        DispatchTable[IRP_MJ_CREATE] = DriverControlCreate;
        DispatchTable[IRP_MJ_CLOSE] = DriverControlClose;
        DispatchTable[IRP_MJ_DEVICE_CONTROL] = DriverControl;

        DECLARE_UNICODE_STRING_SIZE(DeviceName, 500);
        DECLARE_UNICODE_STRING_SIZE(DeviceLinkUnicodeString, 500);
        RtlUnicodeStringPrintf(&DeviceName, NTDEVICE_STRING,
Context->MiniportFriendlyName);
        RtlUnicodeStringPrintf(&DeviceLinkUnicodeString,
LINKNAME_STRING, Context->MiniportFriendlyName);

        NdisZeroMemory(&DeviceAttribute,
sizeof(NDIS_DEVICE_OBJECT_ATTRIBUTES));

        DeviceAttribute.Header.Type =
NDIS_OBJECT_TYPE_DEVICE_OBJECT_ATTRIBUTES;
        DeviceAttribute.Header.Revision =
NDIS_DEVICE_OBJECT_ATTRIBUTES_REVISION_1;

```

```

    DeviceAttribute.Header.Size =
sizeof(NDIS_DEVICE_OBJECT_ATTRIBUTES);

    DeviceAttribute.DeviceName = &DeviceName;
    DeviceAttribute.SymbolicName = &DeviceLinkUnicodeString;
    DeviceAttribute.MajorFunctions = DispatchTable;
    DeviceAttribute.ExtensionSize =
sizeof(FILTER_DEVICE_EXTENSION);

    Status = NdisRegisterDeviceEx(FilterDriverHandle,
&DeviceAttribute,
    &NdisDeviceObject, FilterContextDevice);

    if (Status == NDIS_STATUS_SUCCESS){
        FilterDeviceExtension =
(FILTER_DEVICE_EXTENSION*)NdisGetDeviceReservedExtension(NdisDevic
eObject);

        FilterDeviceExtension->Signature = 'FTDR';
        FilterDeviceExtension->Handle = Context;

        Status = RegAddDevice(Context->FilterModuleName,
Context->MiniportFriendlyName);
    }

    DEBUGP(DL_TRACE, "<==NeuralNetworkMonitorDriverRegisterDevice:
%x\n", Status);

    return (Status);
}

_IRQL_requires_max_(PASSIVE_LEVEL)
VOID NeuralNetworkMonitorDriverDeregisterDevice(NDIS_HANDLE
NdisFilterDeviceHandle){
    if (NdisFilterDeviceHandle != NULL)
        NdisDeregisterDeviceEx(NdisFilterDeviceHandle);
}

NTSTATUS DriverControl(IN PDEVICE_OBJECT fdo, IN PIRP irp) {
    DbgBreakPoint();

    PFILTER_DEVICE_EXTENSION IoFilterextention =
(PFILTER_DEVICE_EXTENSION)NdisGetDeviceReservedExtension(fdo);
    PMS_FILTER FilterContext = (PMS_FILTER)IoFilterextention-
>Handle;

    PIO_STACK_LOCATION IrpSp =
IoGetCurrentIrpStackLocation(irp);
    NTSTATUS status = STATUS_SUCCESS;

    if (IrpSp->FileObject == NULL)
        return(STATUS_UNSUCCESSFUL);

    irp->IoStatus.Information = 0;

    switch (IrpSp->Parameters.DeviceIoControl.IoControlCode)
    {
    case IOCTL_CONNECT_TO_GUI: {

```

```

        WCHAR* wcsbuffer =
ExAllocatePoolWithTag(NonPagedPool, sizeof(WCHAR) * 200, 'SHMI');
        FilterContext->SharedMemoryName->Buffer = wcsbuffer;
        FilterContext->SharedMemoryName->MaximumLength =
200;

        FilterContext->SharedMemoryName->Length = 0;
        LARGE_INTEGER MemSize;
        MemSize.QuadPart = 32000;
        FilterContext->NameGUID =
GenerateUniqueNameByGUID(FilterContext->SharedMemoryName);

        DECLARE_UNICODE_STRING_SIZE(SectionName, 120);

        RtlUnicodeStringPrintf(&SectionName,
L"\\BaseNamedObjects\\%s", FilterContext->SharedMemoryName-
>Buffer);

        FilterContext->SharedMemoryInfo =
(PSHARED_MEMORY_INFO)CreateSharedSection(SectionName,
        MemSize, &FilterContext->SharedMemoryHandle);

        if (!FilterContext->SharedMemoryHandle)
            status = STATUS_UNSUCCESSFUL;

        PVOID outputBuffer =
MmGetSystemAddressForMdlSafe(irp->MdlAddress, NormalPagePriority);

        RtlCopyMemory(outputBuffer, FilterContext-
>SharedMemoryName->Buffer,
            IrpSp-
>Parameters.DeviceIoControl.OutputBufferLength);

        FilterContext->GuiConnected = TRUE;

        break;
    }

    case IOCTL_SEND_SHARED_MEMORY_SIZE: {

        PVOID outputBuffer =
MmGetSystemAddressForMdlSafe(irp->MdlAddress, NormalPagePriority);

        if (FilterContext->SharedMemoryInfo) {
            RtlCopyMemory(outputBuffer, &FilterContext-
>SharedMemoryInfo->MemorySize.QuadPart,
                IrpSp-
>Parameters.DeviceIoControl.OutputBufferLength);
        }
        else status = STATUS_UNSUCCESSFUL;
    }
}
irp->IoStatus.Status = status;
IoCompleteRequest(irp, IO_NO_INCREMENT);

return STATUS_SUCCESS;
}

```

```

NTSTATUS DriverControlCreate(IN PDEVICE_OBJECT fdo, IN PIRP irp)
{
    DbgBreakPoint();

    irp->IoStatus.Status = STATUS_SUCCESS;
    irp->IoStatus.Information = 0;

    PFILTER_DEVICE_EXTENSION IoFilterextention =
(PFILTER_DEVICE_EXTENSION)NdisGetDeviceReservedExtension(fdo);
    PMS_FILTER FilterContext = IoFilterextention->Handle;

    FilterContext->SharedMemoryName =
ExAllocatePoolWithTag(NonPagedPool,
    sizeof(NDIS_STRING), 'SHMI');

    FilterContext->SharedMemoryHandle = NULL;

    if (FilterContext->SharedMemoryName == NULL) {
        irp->IoStatus.Status = STATUS_UNSUCCESSFUL;
        return STATUS_UNSUCCESSFUL;
    }

    NdisZeroMemory(FilterContext->SharedMemoryName,
sizeof(NDIS_STRING));

    IoCompleteRequest(irp, IO_NO_INCREMENT);

    return STATUS_SUCCESS;
}

NTSTATUS DriverControlClose(IN PDEVICE_OBJECT fdo, IN PIRP irp)
{
    DbgBreakPoint();
    NdisGetDeviceReservedExtension(fdo);
    PFILTER_DEVICE_EXTENSION IoFilterextention =
(PFILTER_DEVICE_EXTENSION)NdisGetDeviceReservedExtension(fdo);
    PMS_FILTER FilterContext = (PMS_FILTER)IoFilterextention-
>Handle;

    irp->IoStatus.Status = STATUS_SUCCESS;
    irp->IoStatus.Information = 0;
    IoCompleteRequest(irp, IO_NO_INCREMENT);

    if (FilterContext->SharedMemoryHandle) {
        ZwUnmapViewOfSection(NtCurrentProcess(),
FilterContext->SharedMemoryInfo);
        ZwClose(FilterContext->SharedMemoryHandle);
    }

    if (FilterContext->SharedMemoryName->Buffer) {
        ExFreePoolWithTag(FilterContext->SharedMemoryName-
>Buffer, 'SHMI');
        ExFreePoolWithTag(FilterContext->SharedMemoryName,
'SHMI');
    }
}

```

```

        FilterContext->GuiConnected = FALSE;

        return STATUS_SUCCESS;
    }

NTSTATUS RegAddDevice(UNICODE_STRING ParamName, UNICODE_STRING
DeviceName) {

    NTSTATUS st;
    UNICODE_STRING strKeyName;
    OBJECT_ATTRIBUTES ObjectAttributes;
    HANDLE hParamKey;
    RtlInitUnicodeString(&strKeyName, NNMS_REG_DEVICES_PATH);

    InitializeObjectAttributes(&ObjectAttributes, &strKeyName,
        OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE, NULL,
NULL);

    st = ZwCreateKey(&hParamKey, KEY_ALL_ACCESS,
&ObjectAttributes, 0, NULL, REG_OPTION_NON_VOLATILE, NULL);

    st = ZwSetValueKey(hParamKey, &ParamName, 0, REG_MULTI_SZ,
(VOID*)DeviceName.Buffer,
        DeviceName.Length);

    st = ZwClose(hParamKey);

    return st;
}

NTSTATUS RegDeleteDevice(UNICODE_STRING DeviceName) {

    NTSTATUS st;
    UNICODE_STRING strKeyName;
    OBJECT_ATTRIBUTES ObjectAttributes;
    HANDLE hParamKey;
    RtlInitUnicodeString(&strKeyName, NNMS_REG_DEVICES_PATH);

    InitializeObjectAttributes(&ObjectAttributes, &strKeyName,
        OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE, NULL,
NULL);

    st = ZwOpenKey(&hParamKey, KEY_ALL_ACCESS,
&ObjectAttributes);

    st = ZwDeleteValueKey(hParamKey, &DeviceName);

    ZwClose(hParamKey);

    return st;
}

```

Файл MdlDataOperations.c

```

#include "precomp.h"
#include "MdlDataOperations.h"

```

```

MDL* Get_Mdl_AtOffset(MDL* currentMdl, unsigned int* offset) {

    BOOLEAN round_next = TRUE;

    MDL* mdl = currentMdl;

    if (offset == 0)
        return NULL;

    while (!!mdl && (round_next) ? (mdl->ByteCount <= *offset)
: (mdl->ByteCount < *offset)) {
        *offset -= mdl->ByteCount;
        mdl = mdl->Next;
    }

    return mdl;
}

MDL* Get_Mdl(NET_BUFFER* netbuffer, unsigned int* offset, unsigned
int* DataInPointer) {

    if (offset == 0) {
        if (DataInPointer != NULL)
            *DataInPointer = netbuffer->CurrentMdl-
>ByteCount - netbuffer->CurrentMdlOffset;
        return netbuffer->CurrentMdl;
    }

    *offset += netbuffer->CurrentMdlOffset;

    MDL* mdl = Get_Mdl_AtOffset(netbuffer->CurrentMdl, offset);
    if (DataInPointer != NULL)
        *DataInPointer = mdl->ByteCount - *offset;
    return mdl;
}

void* Get_Buffer(MDL* mdl, unsigned int offset) {

    if (offset >= mdl->ByteCount)
        return NULL;

    void* buffer = MmGetSystemAddressForMdlSafe(mdl,
NormalPagePriority);
    (unsigned char*)buffer += offset;
    return buffer;
}

void* GetData(NET_BUFFER* netbuffer, unsigned int offset, unsigned
int* DataInPointer) {
    if (DataInPointer != NULL)
        *DataInPointer = 0;
    MDL* mdl = Get_Mdl(netbuffer, &offset, DataInPointer);

    void* data = (mdl) ? Get_Buffer(mdl, offset) : NULL;

    return data;
}

```

```

unsigned int GetNetBufferDataSize(NET_BUFFER* nb) {
    MDL* mdl = nb->CurrentMdl;
    unsigned int offset = nb->CurrentMdlOffset;
    unsigned int length = 0;

    while (mdl)
    {
        length += mdl->ByteCount - offset;
        offset = 0;

        mdl = mdl->Next;
    }

    return length;
}

```

Файл Model_train.py

```

#!/usr/bin/env python3
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score, roc_curve, auc, roc_auc_score,
precision_recall_curve
from tensorflow import keras
from tensorflow.keras import layers, callbacks
import joblib
import os
import json
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
import time
# Дополнительные импорты для правильной визуализации
from sklearn.calibration import calibration_curve

warnings.filterwarnings('ignore')

warnings.filterwarnings('ignore')

# Настройка графиков
plt.rcParams['figure.figsize'] = (14, 10)
sns.set_style("whitegrid")

print("\n Загрузка данных")

csv_files = [
    'data/Tuesday-WorkingHours.pcap_ISCX.csv',
    'data/Wednesday-workingHours.pcap_ISCX.csv',
    'data/Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv',
    'data/Thursday-WorkingHours-Afternoon-
Infiltration.pcap_ISCX.csv',
    'data/Friday-WorkingHours-Morning.pcap_ISCX.csv',
    'data/Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv',

```

```

    'data/Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv'
]

df_list = []
for f in csv_files:
    if os.path.exists(f):
        df = pd.read_csv(f, low_memory=False)
        df_list.append(df)
        print(f"\n Загружен: {os.path.basename(f)} -> {len(df)}
        строк")

if not df_list:
    print("\n НЕТ ДАННЫХ!")
    exit(1)

df = pd.concat(df_list, ignore_index=True)
print(f"\n Всего строк: {len(df)}")

print("\n Подготовка данных")

label_col = None
for col in df.columns:
    if 'Label' in col or 'label' in col:
        label_col = col
        break

print(f" Колонка меток: '{label_col}'")

# Метки
y_raw = df[label_col].astype(str).str.strip().str.lower()
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y_raw)
y = pd.Series(y_encoded, index=df.index)

# Признаки
X = df.drop(columns=[label_col])
X.index = df.index
for col in X.columns:
    X[col] = pd.to_numeric(X[col], errors='coerce')
X = X.fillna(0).replace([np.inf, -np.inf], 0)

print(f" Признаков: {X.shape[1]}")

print("\n Фильтрация классов")

# СОХРАНЯЕМ ОРИГИНАЛЬНЫЕ НАЗВАНИЯ КЛАССОВ ДО ПЕРЕКОДИРОВАНИЯ
original_class_names = {}
for i, name in enumerate(label_encoder.classes_):
    original_class_names[i] = str(name)

class_counts = {}
for i in range(len(label_encoder.classes_)):
    class_counts[i] = int((y == i).sum())

print(" Классы до фильтрации:")
for i, name in enumerate(label_encoder.classes_):
    print(f" {i}: {name} -> {class_counts[i]}")

```

```

# Оставляем только классы с >= 6000
keep_classes = [i for i, count in class_counts.items() if count >=
6000]
mask = y.isin(keep_classes)
X = X[mask]
y = y[mask]

# Перекодируем
y_filtered = y.values
new_label_encoder = LabelEncoder()
y_new = new_label_encoder.fit_transform(y_filtered)
y = pd.Series(y_new, index=X.index)

num_classes = len(new_label_encoder.classes_)

# Сохраняем маппинг классов с оригинальными названиями
class_mapping = {}
for new_id, old_id in enumerate(new_label_encoder.classes_):
    original_name = original_class_names[int(old_id)]
    class_mapping[new_id] = original_name

os.makedirs('models', exist_ok=True)
with open('models/class_mapping.json', 'w') as f:
    json.dump(class_mapping, f, indent=2)

print(f"\n Оставлено классов: {num_classes}")
print(f"  Маппинг классов:")
for i in range(num_classes):
    print(f"    {i} -> {class_mapping[i]}")

print("\n Разделение данных")

X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42,
    stratify=y_temp
)

print(f"  Train: {len(X_train)}")
print(f"  Val: {len(X_val)}")
print(f"  Test: {len(X_test)}")

print("\n Масштабирование данных")

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

joblib.dump(scaler, 'models/mlp_scaler.pkl')
print("Scaler сохранен")

# 6. ВЫЧИСЛЕНИЕ CLASS WEIGHTS
print("\n Вычисление весов классов")

total_samples = len(y_train)

```

```

class_counts_train = y_train.value_counts().to_dict()

class_weights = {}
for i in range(num_classes):
    count = class_counts_train.get(i, 1)
    weight = total_samples / (num_classes * count)
    class_weights[i] = weight

print("  Class weights:")
for i in range(num_classes):
    print(f"      {class_mapping[i]}: {class_weights[i]:.4f}")

# 7. ПОСТРОЕНИЕ МОДЕЛИ
print("\n Построение и обучение модели")

model = keras.Sequential([
    layers.Input(shape=(X_train.shape[1],)),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.4),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.3),
    layers.Dense(64, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.2),
    layers.Dense(num_classes, activation='softmax')
])

model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()

# Обучение с class weights
early_stop = callbacks.EarlyStopping(
    monitor='val_loss',
    patience=15,
    restore_best_weights=True,
    verbose=1
)

reduce_lr = callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    min_lr=0.00001,
    verbose=1
)

# Замер времени обучения
start_time = time.time()

```

```

history = model.fit(
    X_train_scaled, y_train.astype(np.int32),
    validation_data=(X_val_scaled, y_val.astype(np.int32)),
    epochs=200,
    batch_size=256,
    class_weight=class_weights,
    callbacks=[early_stop, reduce_lr],
    verbose=1
)

training_time = time.time() - start_time

# Сохраняем историю обучения с преобразованием типов
history_dict = {}
for k, v in history.history.items():
    if isinstance(v, list):
        history_dict[k] = [float(x) for x in v]
    else:
        history_dict[k] = float(v)

with open('models/training history.json', 'w') as f:
    json.dump(history_dict, f, indent=2)
print("История обучения сохранена в JSON")

# Также сохраняем в numpy формате для резерва
np.save('models/training_history.npy', history.history)
print("История обучения сохранена в NPY")

print("\n Тестирование модели")

# Получаем предсказания и вероятности
y_pred = np.argmax(model.predict(X_test_scaled, verbose=0),
axis=1)
y_pred_proba = model.predict(X_test_scaled, verbose=0)

print("\n Графики истории обучения")

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Loss
axes[0, 0].plot(history.history['loss'], label='Train Loss',
linewidth=2)
axes[0, 0].plot(history.history['val_loss'], label='Validation
Loss', linewidth=2)
axes[0, 0].set_title('Model Loss', fontsize=14, fontweight='bold')
axes[0, 0].set_xlabel('Epoch')
axes[0, 0].set_ylabel('Loss')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)

# Accuracy
axes[0, 1].plot(history.history['accuracy'], label='Train
Accuracy', linewidth=2)
axes[0, 1].plot(history.history['val_accuracy'], label='Validation
Accuracy', linewidth=2)
axes[0, 1].set_title('Model Accuracy', fontsize=14,
fontweight='bold')

```

```

axes[0, 1].set_xlabel('Epoch')
axes[0, 1].set_ylabel('Accuracy')
axes[0, 1].legend()
axes[0, 1].grid(True, alpha=0.3)

# Learning Rate
if 'lr' in history.history:
    axes[1, 0].plot(history.history['lr'], color='green',
linewidth=2)
    axes[1, 0].set_title('Learning Rate', fontsize=14,
fontweight='bold')
    axes[1, 0].set_xlabel('Epoch')
    axes[1, 0].set_ylabel('Learning Rate')
    axes[1, 0].set_yscale('log')
    axes[1, 0].grid(True, alpha=0.3)

# Loss difference
train_val_loss_diff = np.array(history.history['loss']) -
np.array(history.history['val_loss'])
axes[1, 1].plot(train_val_loss_diff, color='purple', linewidth=2)
axes[1, 1].axhline(y=0, color='red', linestyle='--', alpha=0.7)
axes[1, 1].set_title('Train - Validation Loss Difference',
fontsize=14, fontweight='bold')
axes[1, 1].set_xlabel('Epoch')
axes[1, 1].set_ylabel('Loss Difference')
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('models/training_history.png', dpi=150,
bbox_inches='tight')
plt.close()
print(" График истории обучения сохранен:
models/training_history.png")

print("\n Базовые метрики классификации")

test_loss, test_acc = model.evaluate(X_test_scaled, y_test,
verbose=0)
print(f" Точность на тесте: {test_acc:.4f} ({test_acc*100:.2f}%)")
print(f" Test Loss: {test_loss:.4f}")

print("\n Classification Report:")
print(classification_report(y_test, y_pred,
target_names=list(class_mapping.values())))

print("\n Матрица ошибок")

y_test_int = y_test.astype(int)
y_pred_int = y_pred.astype(int)

cm = confusion matrix(y_test_int, y_pred_int)
cm = cm.astype(int)

cm_normalized = np.zeros_like(cm, dtype=float)
for i in range(len(cm)):
    row_sum = cm[i].sum()
    if row_sum > 0:
        cm_normalized[i] = cm[i] / row_sum

```

```

        else:
            cm_normalized[i] = cm[i]

class_names = list(class_mapping.values())

max_display_classes = 15
if num_classes > max_display_classes:
    print(f" Класов слишком много ({num_classes}), отображаем
    только {max_display_classes}")
    class_counts = cm.sum(axis=1)
    top_classes_idx = np.argsort(class_counts)[-
max_display_classes:]
    cm = cm[np.ix_(top_classes_idx, top_classes_idx)]
    cm_normalized = cm_normalized[np.ix_(top_classes_idx,
top_classes_idx)]
    class_names = [class_names[i] for i in top_classes_idx]

fig, axes = plt.subplots(1, 2, figsize=(20, 8))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[0],
            xticklabels=class_names, yticklabels=class_names,
            annot_kws={'size': 8})
axes[0].set_xlabel('Predicted', fontsize=12)
axes[0].set_ylabel('Actual', fontsize=12)
axes[0].set_title('Confusion Matrix (Absolute)', fontsize=14,
fontweight='bold')
axes[0].tick_params(axis='x', rotation=45, labelsizе=8)
axes[0].tick_params(axis='y', labelsizе=8)

sns.heatmap(cm_normalized, annot=True, fmt='.2%', cmap='Blues',
ax=axes[1],
            xticklabels=class_names, yticklabels=class_names,
            annot_kws={'size': 8})
axes[1].set_xlabel('Predicted', fontsize=12)
axes[1].set_ylabel('Actual', fontsize=12)
axes[1].set_title('Confusion Matrix (Normalized)', fontsize=14,
fontweight='bold')
axes[1].tick_params(axis='x', rotation=45, labelsizе=8)
axes[1].tick_params(axis='y', labelsizе=8)

plt.tight_layout()
plt.savefig('models/confusion_matrix.png', dpi=150,
bbox_inches='tight')
plt.close()
print(" Матрица ошибок сохранена: models/confusion_matrix.png")

print("\n Детальные метрики по классам")

from sklearn.metrics import precision_recall_fscore_support

precision, recall, fscore, support =
precision_recall_fscore_support(y_test, y_pred)

precision = [float(p) for p in precision]
recall = [float(r) for r in recall]
fscore = [float(f) for f in fscore]
support = [int(s) for s in support]

```

```

metrics_df = pd.DataFrame({
    'Class': list(class_mapping.values()),
    'Precision': precision,
    'Recall': recall,
    'F1-Score': fscore,
    'Support': support
})

metrics_df = metrics_df.sort_values('F1-Score')

print("\n", metrics_df.to_string(index=False))
metrics_df.to_csv('models/per_class_metrics.csv', index=False)

fig, ax = plt.subplots(figsize=(14, max(6, num_classes * 0.3)))

x = np.arange(len(metrics_df))
width = 0.25

ax.barh(x - width, metrics_df['Precision'], width,
label='Precision', alpha=0.8, color='steelblue')
ax.barh(x, metrics_df['Recall'], width, label='Recall', alpha=0.8,
color='darkorange')
ax.barh(x + width, metrics_df['F1-Score'], width, label='F1-
Score', alpha=0.8, color='forestgreen')

ax.set_yticks(x)
ax.set_yticklabels(metrics_df['Class'], fontsize=9)
ax.set_xlabel('Score', fontsize=12)
ax.set_title('Per-Class Metrics (sorted by F1-Score)',
fontsize=14, fontweight='bold')
ax.legend(loc='lower right')
ax.set_xlim([0, 1.05])
ax.axvline(x=0.7, color='red', linestyle='--', alpha=0.5,
label='Acceptable threshold (0.7)')
ax.axvline(x=0.8, color='orange', linestyle='--', alpha=0.5,
label='Good threshold (0.8)')
ax.axvline(x=0.9, color='green', linestyle='--', alpha=0.5,
label='Excellent threshold (0.9)')
ax.grid(True, alpha=0.3, axis='x')

plt.tight_layout()
plt.savefig('models/per_class_metrics.png', dpi=150,
bbox_inches='tight')
plt.close()
print(" График per-class метрик сохранен:
models/per_class_metrics.png")

problem_classes = metrics_df[metrics_df['F1-Score'] < 0.8]
if len(problem_classes) > 0:
    print("\n ПРОБЛЕМНЫЕ КЛАССЫ (F1-Score < 0.8):")
    for , row in problem_classes.iterrows():
        print(f"      {row['Class']}: F1={row['F1-Score']:.3f},
Precision={row['Precision']:.3f}, Recall={row['Recall']:.3f}")

print("\n Анализ уверенности модели")

confidences = np.max(y_pred_proba, axis=1)
correct_pred = (y_pred == y_test)

```

```

mean_conf_correct = np.mean(confidences[correct_pred]) if
np.sum(correct_pred) > 0 else 0
mean_conf_incorrect = np.mean(confidences[~correct_pred]) if
np.sum(~correct_pred) > 0 else 0

print(f" Средняя уверенность для правильных предсказаний:
{mean_conf_correct:.4f}")
print(f" Средняя уверенность для неправильных предсказаний:
{mean_conf_incorrect:.4f}")

fig, ax = plt.subplots(figsize=(12, 6))

bins = np.linspace(0, 1, 51)

if np.sum(correct_pred) > 0:
    ax.hist(confidences[correct_pred], bins=bins, alpha=0.7,
            label=f'Correct (n={np.sum(correct_pred):,},
mean={mean_conf_correct:.3f})',
            density=True, color='green', edgecolor='darkgreen')

if np.sum(~correct_pred) > 0:
    ax.hist(confidences[~correct_pred], bins=bins, alpha=0.7,
            label=f'Incorrect (n={np.sum(~correct_pred):,},
mean={mean_conf_incorrect:.3f})',
            density=True, color='red', edgecolor='darkred')

ax.set_xlabel('Confidence', fontsize=12)
ax.set_ylabel('Density', fontsize=12)
ax.set_title('Confidence Distribution: Correct vs Incorrect
Predictions', fontsize=14, fontweight='bold')
ax.legend()
ax.grid(True, alpha=0.3)

if np.sum(correct_pred) > 0:
    ax.axvline(mean_conf_correct, color='green', linestyle='--',
alpha=0.5, linewidth=2)
if np.sum(~correct_pred) > 0:
    ax.axvline(mean_conf_incorrect, color='red', linestyle='--',
alpha=0.5, linewidth=2)

plt.tight_layout()
plt.savefig('models/confidence_distribution.png', dpi=150,
bbox_inches='tight')
plt.close()
print(" График уверенности сохранен:
models/confidence_distribution.png")

print("\n Калибровка модели")

fig, ax = plt.subplots(figsize=(12, 8))

calibration_data = []
valid_classes = [i for i in range(num_classes) if np.sum(y_test ==
i) >= 100]

for i in valid_classes:
    class_mask = (y_test == i)

```

```

class_count = class_mask.sum()

try:
    prob_true, prob_pred = calibration_curve(
        class_mask.astype(int),
        y_pred_proba[:, i],
        n_bins=min(10, class_count // 10),
        strategy='uniform'
    )

    # Правильный расчет ECE
    n_bins_actual = len(prob_true)
    bin_boundaries = np.linspace(0, 1, n_bins_actual + 1)

    ece_weighted = 0
    total_samples_class = class_count

    for bin_idx in range(n_bins_actual):
        bin_start = bin_boundaries[bin_idx]
        bin_end = bin_boundaries[bin_idx + 1]

        in_bin = (y_pred_proba[:, i] > bin_start) &
(y_pred_proba[:, i] <= bin_end)
        in_bin = in_bin & class_mask
        bin_size = in_bin.sum()

        if bin_size > 0:
            error = np.abs(prob_true[bin_idx] -
prob_pred[bin_idx])
            ece_weighted += error * bin_size

    ece_weighted = ece_weighted / total_samples_class

    calibration_data.append({
        'class': class_mapping[i],
        'ece': ece_weighted
    })

    ax.plot(prob_pred, prob_true, marker='o', linewidth=2,
            label=f'{class_mapping[i]}
(ECE={ece_weighted:.3f})', markersize=4)

except Exception as e:
    print(f" Ошибка калибровки для {class_mapping[i]}: {e}")

ax.plot([0, 1], [0, 1], 'k--', lw=2, label='Perfectly Calibrated')

ax.set_xlim([0.0, 1.0])
ax.set_ylim([0.0, 1.0])
ax.set_xlabel('Mean Predicted Probability', fontsize=12)
ax.set_ylabel('Fraction of Positives', fontsize=12)
ax.set_title('Calibration Curves', fontsize=14, fontweight='bold')
ax.legend(loc='best', fontsize=8)
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('models/calibration_curves.png', dpi=150,
bbox_inches='tight')

```

```

plt.close()

print("\n Expected Calibration Error (ECE) по классам:")
for data in calibration_data:
    print(f"      {data['class']}: {data['ece']:.4f}")

print("\n Precision-Recall кривые")

fig, ax = plt.subplots(figsize=(14, 10))

pr_curves_data = []

for i in valid_classes:
    try:
        precision_curve, recall_curve, thresholds_pr =
precision_recall_curve(
            (y_test == i), y_pred_proba[:, i]
        )

        f1_scores = 2 * (precision_curve * recall_curve) /
(precision_curve + recall_curve + 1e-10)
        best_idx = np.argmax(f1_scores)
        best_f1 = f1_scores[best_idx]
        best_threshold = thresholds_pr[best_idx] if
len(thresholds_pr) > best_idx else 0.5

        pr_curves_data.append({
            'class': class_mapping[i],
            'best_f1': best_f1,
            'best_threshold': best_threshold
        })

        ax.plot(recall_curve, precision_curve, lw=2,
            label=f'{class_mapping[i]} (F1={best_f1:.3f},
thr={best_threshold:.2f})')
    except Exception as e:
        print(f" Ошибка при PR кривой для {class_mapping[i]}:
{e}")

ax.set_xlim([0.0, 1.0])
ax.set_ylim([0.0, 1.05])
ax.set_xlabel('Recall', fontsize=12)
ax.set_ylabel('Precision', fontsize=12)
ax.set_title('Precision-Recall Curves per Class', fontsize=14,
fontweight='bold')
ax.legend(loc='best', fontsize=8, ncol=2 if len(valid_classes) > 5
else 1)
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('models/precision_recall_curves.png', dpi=150,
bbox_inches='tight')
plt.close()
print(" Precision-Recall кривые сохранены:
models/precision_recall_curves.png")

print("\n Оптимальные пороги для каждого класса:")
for data in sorted(pr_curves_data, key=lambda x: x['best_f1']):

```

```

    print(f"      {data['class']}: F1={data['best_f1']:.3f},
    порог={data['best_threshold']:.2f}")

print("\n Анализ порогов классификации")

thresholds = np.arange(0.1, 1.0, 0.05)
threshold_metrics = []

for thr in thresholds:
    max_probs = np.max(y_pred_proba, axis=1)
    confident_mask = max_probs >= thr

    if confident_mask.sum() > 0:
        y_pred_confident = y_pred[confident_mask]
        y_test_confident = y_test[confident_mask]
        acc = accuracy_score(y_test_confident, y_pred_confident)
        coverage = confident_mask.sum() / len(y_test)

        threshold_metrics.append({
            'threshold': thr,
            'accuracy': acc,
            'coverage': coverage
        })

if threshold_metrics:
    thr_df = pd.DataFrame(threshold_metrics)

    fig, ax1 = plt.subplots(figsize=(12, 6))

    ax1.plot(thr_df['threshold'], thr_df['accuracy'], 'b-o',
linewidth=2, markersize=8, label='Accuracy')
    ax1.set_xlabel('Confidence Threshold', fontsize=12)
    ax1.set_ylabel('Accuracy', fontsize=12, color='b')
    ax1.tick_params(axis='y', labelcolor='b')
    ax1.set_ylim([0.85, 1.01])
    ax1.grid(True, alpha=0.3)

    ax2 = ax1.twinx()
    ax2.plot(thr_df['threshold'], thr_df['coverage'], 'r-s',
linewidth=2, markersize=8, label='Coverage')
    ax2.set_ylabel('Coverage (fraction of data)', fontsize=12,
color='r')
    ax2.tick_params(axis='y', labelcolor='r')
    ax2.set_ylim([0, 1.05])

    plt.title('Accuracy vs Coverage by Confidence Threshold',
fontsize=14, fontweight='bold')

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    ax1.legend(lines1 + lines2, labels1 + labels2, loc='lower
left')

    plt.tight_layout()
    plt.savefig('models/threshold_analysis.png', dpi=150,
bbox_inches='tight')
    plt.close()

```

```

    print(" Анализ порогов сохранен:
models/threshold_analysis.png")

    print("\n Рекомендации по порогам:")
    optimal_tradeoff = thr_df.iloc[(thr_df['accuracy'] *
thr_df['coverage']).argmax()]
    print(f"    Оптимальный баланс:
порог={optimal_tradeoff['threshold']:.2f}, "
          f"accuracy={optimal_tradeoff['accuracy']:.4f},
coverage={optimal_tradeoff['coverage']:.2%}")

print("\n Тест производительности")

inference_times = []
for _ in range(100):
    start = time.time()
    _ = model.predict(X_test_scaled[:100], verbose=0)
    inference_times.append(time.time() - start)

avg_inference_time = np.mean(inference_times)
std_inference_time = np.std(inference_times)

print(f" Среднее время инференса (100 образцов):
{avg_inference_time*1000:.2f} ± {std_inference_time*1000:.2f} ms")
print(f" Throughput: {100/avg_inference_time:.1f} образцов/сек")
print(f" Общее время обучения: {training_time:.2f} сек")

print("\n Детальный анализ ошибок")

errors_mask = (y_pred != y_test)
errors_df = pd.DataFrame({
    'true_class': [class_mapping[i] for i in y_test[errors_mask]],
    'pred_class': [class_mapping[i] for i in y_pred[errors_mask]],
    'confidence': confidences[errors_mask]
})

confusion_pairs = errors_df.groupby(['true_class',
'pred_class']).size().sort_values(ascending=False)

print("\n Топ-10 самых частых ошибок:")
for (true_cls, pred_cls), count in
confusion_pairs.head(10).items():
    print(f"    {true_cls} -> {pred_cls}: {count} ошибок")

print(f"\n Средняя уверенность для ошибок:
{errors_df['confidence'].mean():.4f}")
errors_df.to_csv('models/error_analysis.csv', index=False)

print("\n Анализ важности признаков")

from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')

# Получаем названия признаков
feature_names = X_train.columns.tolist()

```

```

n_features = len(feature_names)
print(f"  Анализ {n_features} признаков...")

perm_top_features = []

try:
    # Берем выборку - ВАЖНО: используем .copy() и правильную
индексацию
    sample_size = min(2000, X_test_scaled.shape[0])

    # Случайные индексы в пределах размера массива
    np.random.seed(42)
    indices = np.random.choice(X_test_scaled.shape[0],
sample_size, replace=False)

    # Извлекаем выборку
    X_sample = X_test_scaled[indices].copy()
    y_sample = y_test.iloc[indices] if hasattr(y_test, 'iloc')
else y_test[indices]

    print(f"    Используем {sample_size} образцов из
{X_test_scaled.shape[0]}")

    # Базовая точность
    base_pred = np.argmax(model.predict(X_sample, verbose=0),
axis=1)
    base_acc = accuracy_score(y_sample, base_pred)
    print(f"    Базовая точность: {base_acc:.4f}")

    # Вычисляем важность для каждого признака
    for i in range(n_features):
        # Копируем и перемешиваем признак
        X_permuted = X_sample.copy()
        np.random.shuffle(X_permuted[:, i])

        # Предсказываем
        perm_pred = np.argmax(model.predict(X_permuted,
verbose=0), axis=1)
        perm_acc = accuracy_score(y_sample, perm_pred)

        # Важность
        importance = max(0, base_acc - perm_acc)
        perm_top_features.append((feature_names[i], importance))

        if (i + 1) % 20 == 0:
            print(f"    Обработано {i + 1}/{n_features}
признаков")

    # Сортируем
    perm_top_features.sort(key=lambda x: -x[1])

    print("  20 Наиболее важных признаков")
    print("  (Показывает, насколько падает точность при удалении
признака)\n")
    for rank, (name, imp) in enumerate(perm_top_features[:20], 1):
        print(f"    {rank:2d}. {name}: {imp:.6f}")

except Exception as e:

```

```

print(f" Ошибка: {e}")
import traceback
traceback.print_exc()

# Вторая попытка - с еще меньшей выборкой
try:
    print("\n Пробуем с меньшей выборкой (1000 образцов)...")
    sample_size = min(1000, X_test_scaled.shape[0])
    indices = np.random.choice(X_test_scaled.shape[0],
sample_size, replace=False)
    X_sample = X_test_scaled[indices].copy()
    y_sample = y_test.iloc[indices] if hasattr(y_test, 'iloc')
else y_test[indices]

    base_pred = np.argmax(model.predict(X_sample, verbose=0),
axis=1)
    base_acc = accuracy_score(y_sample, base_pred)
    print(f" Базовая точность: {base_acc:.4f}")

    perm_top_features = []
    for i in range(n_features):
        X_permuted = X_sample.copy()
        np.random.shuffle(X_permuted[:, i])
        perm_pred = np.argmax(model.predict(X_permuted,
verbose=0), axis=1)
        perm_acc = accuracy_score(y_sample, perm_pred)
        perm_top_features.append((feature_names[i], max(0,
base_acc - perm_acc)))

        if (i + 1) % 30 == 0:
            print(f" Обработано {i + 1}/{n_features}")

    perm_top_features.sort(key=lambda x: -x[1])

    print(" 20 Наиболее важных признаков")
    for rank, (name, imp) in enumerate(perm_top_features[:20],
1):
        print(f" {rank:2d}. {name}: {imp:.6f}")

    except Exception as e2:
        print(f" Вторая попытка тоже не удалась: {e2}")
        perm_top_features = []

if perm_top_features:
    print("\n Визуализация важности признаков")

    fig, ax = plt.subplots(figsize=(12, 10))

    names = [f[0][:45] + '...' if len(f[0]) > 45 else f[0] for f
in perm_top_features[:20][::-1]]
    values = [f[1] for f in perm_top_features[:20][::-1]]
    y_pos = np.arange(len(names))

    bars = ax.barh(y_pos, values, color='steelblue', alpha=0.8,
edgecolor='navy')

    # Добавляем значения
    for i, (bar, val) in enumerate(zip(bars, values)):

```

```

        ax.text(val + 0.001, bar.get_y() + bar.get_height()/2,
                f'{val:.6f}', va='center', fontsize=8)

    ax.set_yticks(y_pos)
    ax.set_yticklabels(names, fontsize=9)
    ax.set_xlabel('Drop in Accuracy (Importance)', fontsize=12)
    ax.set_ylabel('Features', fontsize=12)
    ax.set_title('Feature Importance (Permutation Importance)',
    fontsize=14, fontweight='bold')
    ax.invert_yaxis()
    ax.grid(True, alpha=0.3, axis='x')

    plt.tight_layout()
    plt.savefig('models/feature_importance.png', dpi=150,
    bbox_inches='tight')
    plt.close()
    print(" График сохранен: models/feature_importance.png")

    # Сохраняем CSV
    importance_df = pd.DataFrame(perm_top_features,
    columns=['Feature', 'Importance'])
    importance_df.to_csv('models/feature_importance.csv',
    index=False)
    print(" CSV сохранен: models/feature_importance.csv")
else:
    print("\n Не удалось вычислить важность признаков")

print("\n Сохранение отчёта в файл")

with open('models/COMPREHENSIVE_TEST_REPORT.txt', 'w',
encoding='utf-8') as f:
    f.write("КОМПЛЕКСНЫЙ ОТЧЕТ О ТЕСТИРОВАНИИ МОДЕЛИ\n")

    f.write("Основные метрики:\n")
    f.write(f" Время обучения: {training_time:.2f} сек\n")
    f.write(f" Точность на тесте: {test_acc:.4f}
    ({test_acc*100:.2f}%) \n")
    f.write(f" Test Loss: {test_loss:.4f} \n\n")

    f.write("Уверенность модели:\n")
    f.write(f" Средняя уверенность (правильные):
    {mean_conf_correct:.4f} \n")
    f.write(f" Средняя уверенность (ошибки):
    {mean_conf_incorrect:.4f} \n")
    f.write(f" Разница в уверенности: {mean_conf_correct -
    mean_conf_incorrect:.4f} \n\n")

    f.write("Производительность:\n")
    f.write(f" Среднее время инференса (100 образцов):
    {avg_inference_time*1000:.2f} ms \n")
    f.write(f" Throughput: {100/avg_inference_time:.1f}
    образцов/сек \n\n")

    f.write("Распределение данных:\n")
    f.write(f" Train: {len(X_train):,} \n")
    f.write(f" Validation: {len(X_val):,} \n")
    f.write(f" Test: {len(X_test):,} \n\n")

```

```

f.write("CLASSIFICATION REPORT\n")
f.write(classification_report(y_test, y_pred,
target_names=list(class_mapping.values())))

f.write("Ошибки\n")
for (true_cls, pred_cls), count in
confusion_pairs.head(10).items():
    f.write(f" {true_cls} -> {pred_cls}: {count} ошибок\n")

f.write("Калибровка (ECE)\n")
for data in calibration_data:
    f.write(f" {data['class']}: {data['ece']:.4f}\n")

# Анализ важности признаков (если есть)
if perm_top_features:
    f.write("Анализ важности признаков\n")
    f.write(" Метод: Permutation Importance (единственно
корректный для нейросети)\n")
    f.write(" Показывает падение точности при удалении
признака\n\n")

    for rank, (name, imp) in enumerate(perm_top_features[:20],
1):
        f.write(f" {rank:2d}. {name}: {imp:.6f}\n")

    if len(perm_top_features) > 0:
        f.write(" Вывод:\n")
        f.write(f" • Самый важный признак:
{perm_top_features[0][0]}\n")
        if len(perm_top_features) > 1:
            f.write(f" • Второй по важности:
{perm_top_features[1][0]}\n")
        if len(perm_top_features) > 2:
            f.write(f" • Третий по важности:
{perm_top_features[2][0]}\n")

# Проблемные классы
if len(problem_classes) > 0:
    f.write(" Проблемные классы (F1-Score < 0.8)\n")
    for _, row in problem_classes.iterrows():
        f.write(f" {row['Class']}: F1={row['F1-Score']:.3f},
Precision={row['Precision']:.3f}, Recall={row['Recall']:.3f}\n")

rec_num = 1
if len(problem_classes) > 0:
    f.write(f" {rec_num}. Добавить данных для проблемных
классов\n")
    rec_num += 1
    if mean_conf_incorrect > 0.8:
        f.write(f" {rec_num}. Уменьшить уверенность (label
smoothing, dropout)\n")
        rec_num += 1
    if test_acc > 0.92:
        f.write(f" {rec_num}. Общая точность хорошая\n")

model.save('models/mlp_model.keras')

```

```

model.save_weights('models/mlp_weights.weights.h5')
print("Модель сохранена")

class MLPWrapper:
    def __init__(self, model, scaler, class_mapping):
        self.model = model
        self.scaler = scaler
        self.class_mapping = class_mapping

    def predict(self, X):
        if isinstance(X, pd.DataFrame):
            X = X.values
        X_scaled = self.scaler.transform(X)
        pred_probs = self.model.predict(X_scaled, verbose=0)
        return np.argmax(pred_probs, axis=1)

    def predict_proba(self, X):
        if isinstance(X, pd.DataFrame):
            X = X.values
        X_scaled = self.scaler.transform(X)
        return self.model.predict(X_scaled, verbose=0)

wrapper = MLPWrapper(model, scaler, class_mapping)
joblib.dump(wrapper, 'models/mlp_model.pkl')
print("Обертка сохранена")

# Показать все графики (если запускается интерактивно)
try:
    plt.show()
except:
    print("\n (графики сохранены в файлы)")

```

ПРИЛОЖЕНИЕ Б

(обязательное)

Программный анализатор сетевого трафика

Презентация

Выпускная квалификационная
работа на тему
«Программный анализатор
сетевого трафика»

Выполнил:
студент группы 24ВВим1
Тихонов А.А.



Цели и задачи работы



01

Разработка
эффективного
алгоритма передачи
сетевого трафика

02

Разработка
драйвера-
перехватчика
сетевого трафика

03

Анализ возможностей
нейронных сетей в
обнаружении сетевых
атак

04

Обучение нейронной
сети обнаружения
сетевых атак

05

Разработка
трёхкомпонентной системы
обнаружения сетевых атак



Разработанный алгоритм передачи данных

Для решения задачи по оптимизации скорости передачи сетевого трафика был разработан специальный алгоритм передачи данных, позволяющий быстро передавать сетевые пакеты с уровня ядра системы на пользовательский уровень

Ключевые особенности алгоритма:

- Разделяемая память работает по принципу кольцевого буфера
- Lock Free доступ к данным
- Синхронизация при помощи SpinLock и CompletionPort технологий



Драйвер перехвата

Перехват сетевого трафика для передачи осуществляется в драйвере фильтре NDIS. Данный драйвер встраивается в сетевой стек системы и позволяет перехватывать весь сетевой трафик, проходящий через сетевые адаптеры

Выполняемые задачи:

- Перехват сетевого трафика
- Создание кольцевого буфера
- Передача трафика на 3 кольцо
- Установка и перехват трафика с нескольких сетевых адаптеров



Выбор датасета

Главной особенностью набора данных CIC-IDS-2017 для обучения нейронной сети является тот факт что для анализа сетевой атаки используются признаки не отдельного сетевого пакета а целого сетевого потока.

Классы сетевых атак из датасета:

- benign
- ddos
- dos goldeneye
- dos hulk
- ftp-patator
- portscan



Популярные датасеты сетевых атак

➤ NSL-KDD

➤ KDD Cup 99

➤ UNSW-NB15

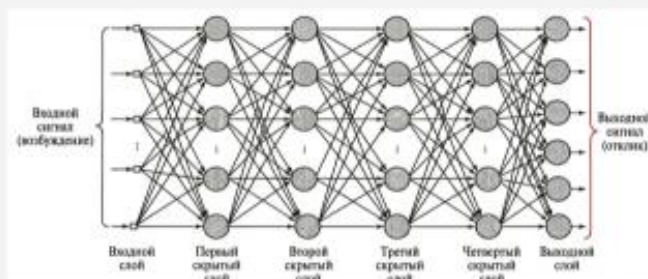
➤ AWD

➤ CIC-Modbus2023

Разработка нейронной сети

Компонент нейронной сети был разработан при помощи библиотек TensorFlow и Keras. В качестве архитектуры модели был выбран многослойный перцептрон со следующей структурой внутренних 4 слоёв:

- Слои нейронов размерностью от 512 до 64
- Функция активации ReLU
- Batch Normalization
- Dropout слои от 0.4 до 0.2
- Выходной слой Softmax



Разработка нейронной сети



01

Использование оптимизатора обучения Adam

02

Использование функций ранней остановки обучения

03

Использование функции корректировки скорости обучения ReduceLROnPlateau

04

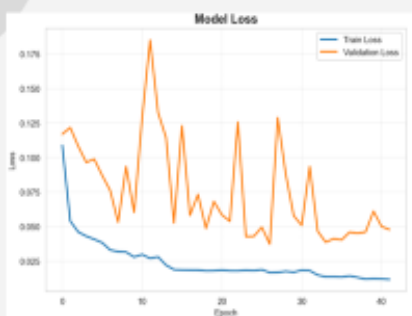
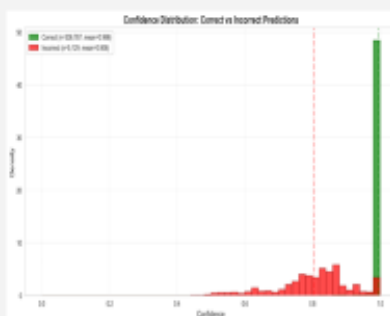
Подготовка данных обучения с отсечением нерепрезентативных классов

05

Стандартизация признаков классов и формирование весов классов для противодействия дисбалансу классов



Результаты обучения модели



Classification Report:				
	precision	recall	f1-score	support
benign	1.00	0.99	0.99	261477
ddos	1.00	1.00	1.00	19204
dos goldeneye	0.81	1.00	0.90	1544
dos hulk	0.93	1.00	0.96	34661
ftp-patator	0.93	1.00	0.96	1191
portscan	0.99	1.00	0.99	23839
accuracy			0.99	341916
macro avg	0.94	1.00	0.97	341916
weighted avg	0.99	0.99	0.99	341916

Основные результаты:

- Точность модели 98.5%
- Высокая точность обнаружения атак
- Высокая уверенность в правильных предсказаниях и низкая у неправильных
- Скорость обработки – 2155 образцов в секунду





Архитектура системы

Драйвер-фильтр
NDIS

- перехватывает трафик на уровне ядра

Сетевой анализатор
(сниффер)

- визуализация и передача данных

Нейронная сеть

- анализ трафика, обнаружение сетевых атак



Влияние ПО на скорость

Скорость с выключенным программным комплексом

```

C:\Users\Tibon\Desktop\iperf2\iperf2 -c 192.168.0.26 -t 10 -b 0
connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49660 connected to 192.168.0.26 port 5201
[ID] Interval      Transfer    Bandwidth
[ 4] 0.00-1.00 sec  533 MBytes  4.47 Gbits/sec
[ 4] 1.00-2.00 sec  573 MBytes  4.61 Gbits/sec
[ 4] 2.00-3.00 sec  575 MBytes  4.62 Gbits/sec
[ 4] 3.00-4.00 sec  578 MBytes  4.65 Gbits/sec
[ 4] 4.00-5.00 sec  582 MBytes  4.68 Gbits/sec
[ 4] 5.00-6.00 sec  575 MBytes  4.63 Gbits/sec
[ 4] 6.00-7.00 sec  586 MBytes  4.92 Gbits/sec
[ 4] 7.00-8.00 sec  560 MBytes  4.69 Gbits/sec
[ 4] 8.00-9.00 sec  586 MBytes  4.96 Gbits/sec
[ 4] 9.00-10.00 sec  574 MBytes  4.82 Gbits/sec
-----
[ID] Interval      Transfer    Bandwidth
[ 4] 0.00-10.00 sec  5.59 Gbytes  4.80 Gbits/sec  sender
[ 4] 0.00-10.00 sec  5.59 Gbytes  4.80 Gbits/sec  receiver
  
```

Скорость со включенным программным комплексом

```

C:\Users\Tibon\Desktop\iperf2\iperf2 -c 192.168.0.26 -t 10 -b 0
connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49667 connected to 192.168.0.26 port 5201
[ID] Interval      Transfer    Bandwidth
[ 4] 0.00-1.00 sec  279 MBytes  2.39 Gbits/sec
[ 4] 1.00-2.00 sec  381 MBytes  3.26 Gbits/sec
[ 4] 2.00-3.00 sec  396 MBytes  3.40 Gbits/sec
[ 4] 3.00-4.00 sec  279 MBytes  2.34 Gbits/sec
[ 4] 4.00-5.00 sec  267 MBytes  2.41 Gbits/sec
[ 4] 5.00-6.00 sec  283 MBytes  2.39 Gbits/sec
[ 4] 6.00-7.00 sec  304 MBytes  2.58 Gbits/sec
[ 4] 7.00-8.00 sec  385 MBytes  3.29 Gbits/sec
[ 4] 8.00-9.00 sec  386 MBytes  3.40 Gbits/sec
[ 4] 9.00-10.00 sec  277 MBytes  2.32 Gbits/sec
-----
[ID] Interval      Transfer    Bandwidth
[ 4] 0.00-10.00 sec  2.76 Gbytes  2.37 Gbits/sec  sender
[ 4] 0.00-10.00 sec  2.76 Gbytes  2.37 Gbits/sec  receiver
  
```





Влияние ПО на скорость



Скорость с выключенным программным комплексом

```
C:\Users\Tisha\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 200000000
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49792 connected to 192.168.0.26 port 5201
ID: Interval Transfer Bandwidth
 4: 0.00-1.00 sec 22.2 Mbytes 187 Mbits/sec
 4: 1.00-2.00 sec 24.1 Mbytes 202 Mbits/sec
 4: 2.00-3.02 sec 23.1 Mbytes 191 Mbits/sec
 4: 3.02-4.02 sec 23.9 Mbytes 200 Mbits/sec
 4: 4.02-5.00 sec 24.3 Mbytes 203 Mbits/sec
 4: 5.00-6.02 sec 23.2 Mbytes 192 Mbits/sec
 4: 6.02-7.00 sec 24.2 Mbytes 202 Mbits/sec
 4: 7.00-8.02 sec 23.4 Mbytes 195 Mbits/sec
 4: 8.02-9.02 sec 24.9 Mbytes 209 Mbits/sec
 4: 9.02-10.00 sec 23.4 Mbytes 199 Mbits/sec
-----
ID: Interval Transfer Bandwidth
 4: 0.00-10.00 sec 237 Mbytes 199 Mbits/sec sender
 4: 0.00-10.00 sec 237 Mbytes 199 Mbits/sec receiver
iperf Done.
```

Скорость со включенным программным комплексом

```
C:\Users\Tisha\Desktop\iperf3>iperf3 -c 192.168.0.26 -t 10 -b 200000000
Connecting to host 192.168.0.26, port 5201
[ 4] local 192.168.0.25 port 49790 connected to 192.168.0.26 port 5201
ID: Interval Transfer Bandwidth
 4: 0.00-1.00 sec 23.8 Mbytes 198 Mbits/sec
 4: 1.00-2.02 sec 23.6 Mbytes 196 Mbits/sec
 4: 2.02-3.02 sec 24.5 Mbytes 206 Mbits/sec
 4: 3.02-4.00 sec 23.4 Mbytes 195 Mbits/sec
 4: 4.00-5.00 sec 23.9 Mbytes 200 Mbits/sec
 4: 5.00-6.02 sec 23.6 Mbytes 195 Mbits/sec
 4: 6.02-7.00 sec 24.5 Mbytes 206 Mbits/sec
 4: 7.00-8.00 sec 23.6 Mbytes 198 Mbits/sec
 4: 8.00-9.00 sec 23.6 Mbytes 196 Mbits/sec
 4: 9.00-10.00 sec 23.4 Mbytes 195 Mbits/sec
-----
ID: Interval Transfer Bandwidth
 4: 0.00-10.00 sec 237 Mbytes 199 Mbits/sec sender
 4: 0.00-10.00 sec 237 Mbytes 199 Mbits/sec receiver
iperf Done.
```

Возможности дальнейшего развития



01

Возможность переноса нейросетового компонента на выделенный сервер

02

Добавление возможности дообучения нейронной сети

03

Добавление версии программного комплекса для ОС Linux

04

Расширение функционала графического интерфейса



Результаты работы

- Разработан эффективный алгоритм передачи данных с уровня ядра на пользовательский уровень
- Разработан драйвер фильтр NDIS, перехватывающий сетевой трафик и передающий его на пользовательский уровень
- Разработана нейронная сеть с многоклассовой классификацией сетевых атак с высокой скоростью обработки объектов
- Разработан графический интерфейс для обеспечения связи между драйвером и нейронной сетью

