МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Доцент кафедры ИАОУ ФГАОУ ВО
«ЮУрГУ (НИУ)», к.т.н.

_____ А.А. Шинкарев

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка приложения для анализа сетевого трафика в режиме реального времени на основе методов машинного обучения

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2024.308-478.ВКР

Научный руководитель,
доцент кафедры СП, к.т.н.
_____ М.В. Сухов

Автор работы,
студент группы КЭ-228
_____ Г.П. Панюшкин

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистранта
студенту группы КЭ-228
Панюшкину Георгию Павловичу,
обучающемуся по направлению
09.04.04 «Программная инженерия»
(магистерская программа «Искусственный интеллект и инженерия данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка приложения для анализа сетевого трафика в режиме реального времени на основе методов машинного обучения.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**

3.1. Meng X., Lin C., Wang Y., Zhang Y. Generative Pretrained Transformer for Network Traffic [Электронный ресурс] // arXiv.org. 2023. Дата обновления: 17.05.2023 г. URL: https://arxiv.org/abs/2304.09513 (дата обращения: 13.05.2024 г.).
3.2. Abreu D., Abelem A. OMINACS: Online ML-based IOT network attack detection and classification system. // 2022 IEEE Latin-American Conference on Communications (LATINCOM), 2022. – С. 1–6.
3.3. Jonathan O., Misra S., Osamor V. Comparative analysis of machine learning techniques for network traffic classification. // IOP Conference Series: Earth and Environmental Science, 2021. – Т. 655. – №. 1. – С. 12–25.
3.4. Kumari K., Mrunalini M. Detecting denial of service attacks using machine learning algorithms. // Journal of Big Data, 2022. – Т. 9. – № 1. – С. 56–73.

4. Перечень подлежащих разработке вопросов

4.1. Провести анализ предметной области.

4.2. Собрать набор данных.

4.3. Выбрать модели машинного обучения.

4.4. Реализовать выбранные модели машинного обучения.

4.5. Спроектировать систему анализа трафика в реальном времени.

4.6. Реализовать систему анализа интернет-трафика в реальном времени.

4.7. Протестировать систему анализа интернет-трафика в реальном времени.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.т.н. М.В. Сухов

Задание принял к исполнению Г.П. Панюшкин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

Анализ сетевого трафика является важной задачей в разных сферах деятельности. Это может быть анализ с целью выделить основные устройства, с которых пользователи заходят на сайт, анализ трафика с целью понять откуда на ресурс приходят пользователи, а также это может быть анализ с целью выявления возможных вторжений или сетевых атак.

Последняя задача стоит особенно остро в текущее время. С каждым годом человечество все плотнее использует интернет в своей жизни: увеличивается количество умных устройств, развитие облачных технологий. Поэтому обеспечение безопасности и надежности сервисов становится критической задачей для множества компаний.

Согласно отчетам компании Kaspersky и Securelist количество и качество проводимых DDoS атак с каждым годом увеличивается. В период с первого квартала 2020 года по третий квартал 2022 года количество проводимых атак увеличилось практически в пять раз [1–3]. Динамика роста количества атак за этот период представлена на рисунках 1–3.

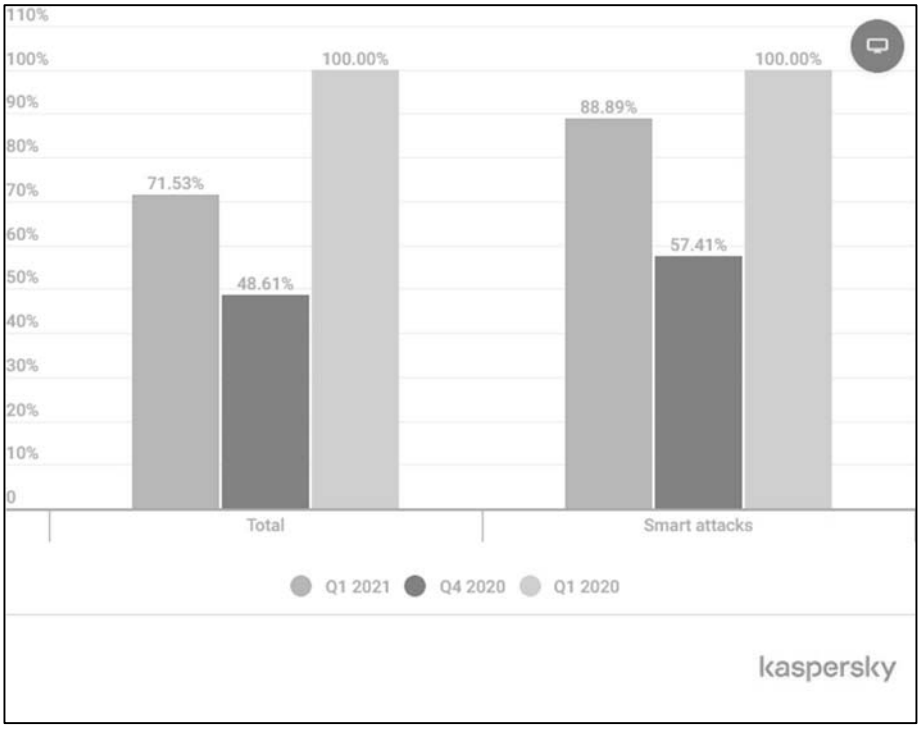

Рисунок 1 – Динамика роста количества атак за 2020 год [1]

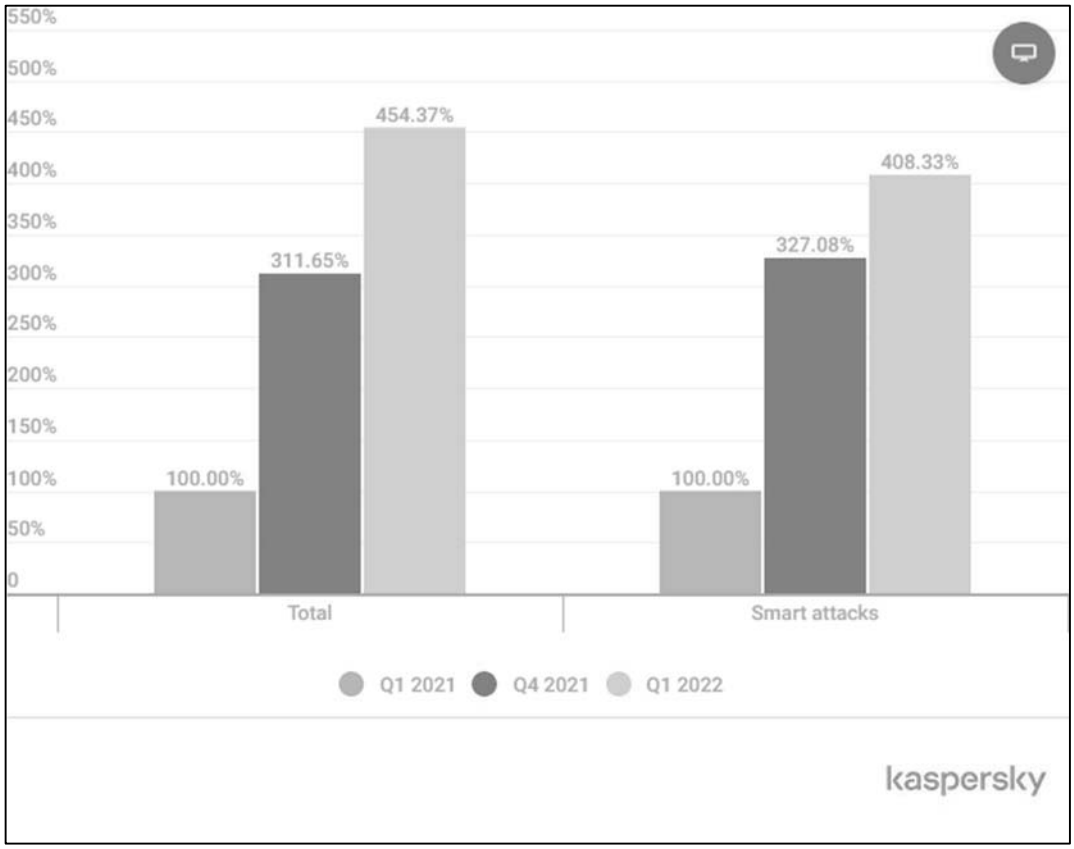

Рисунок 2 – Динамика количества роста атак за 2021 год [2]

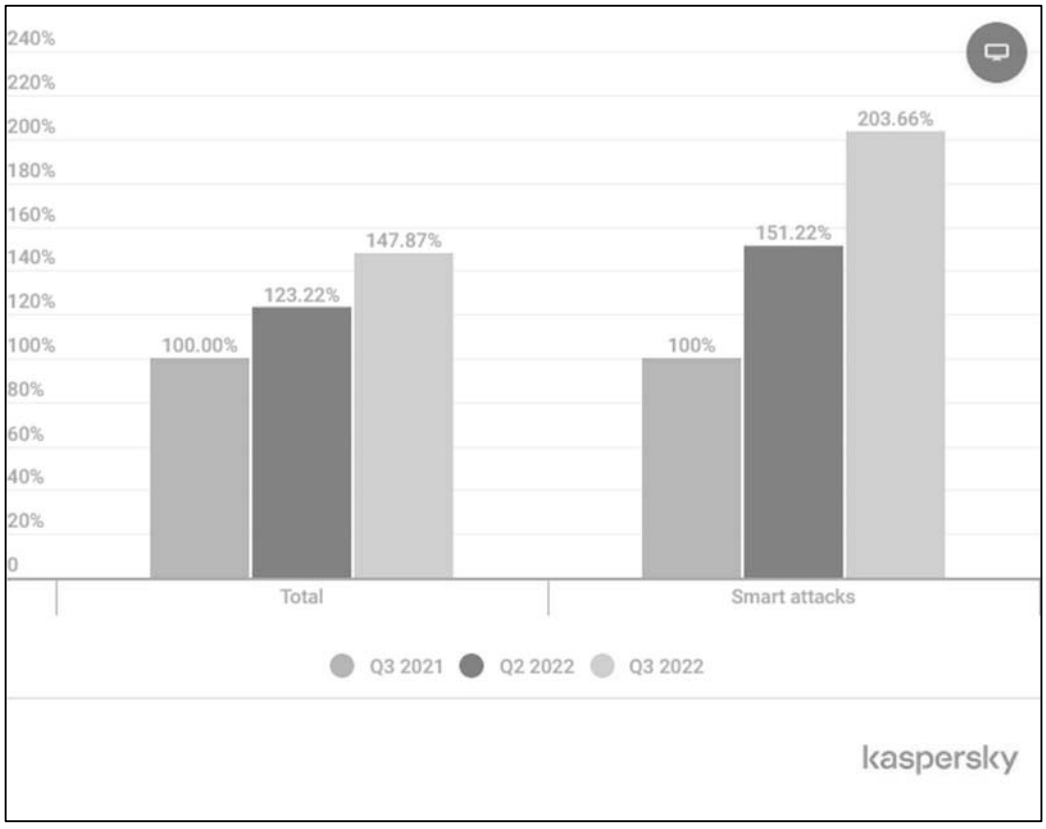

Рисунок 3 – Динамика роста количества атак за 2022 год [3]

В 2023 году тенденция роста количества DDoS-атак продолжилась. Согласно отчетам компании DDoS-Guard, общее число атак в 2023 году увеличилось практически в 2 раза. Компания также заявляет о значительном росте устройств, с которых производятся атаки. Среди таких устройств компания выделяет умные телевизоры, точки доступа, камеры видеонаблюдения. Тенденция использования таких устройств сохраняется последние несколько лет. Распределение атак по месяцам года и сравнение с прошлым годом представлено на рисунке 4 [4].

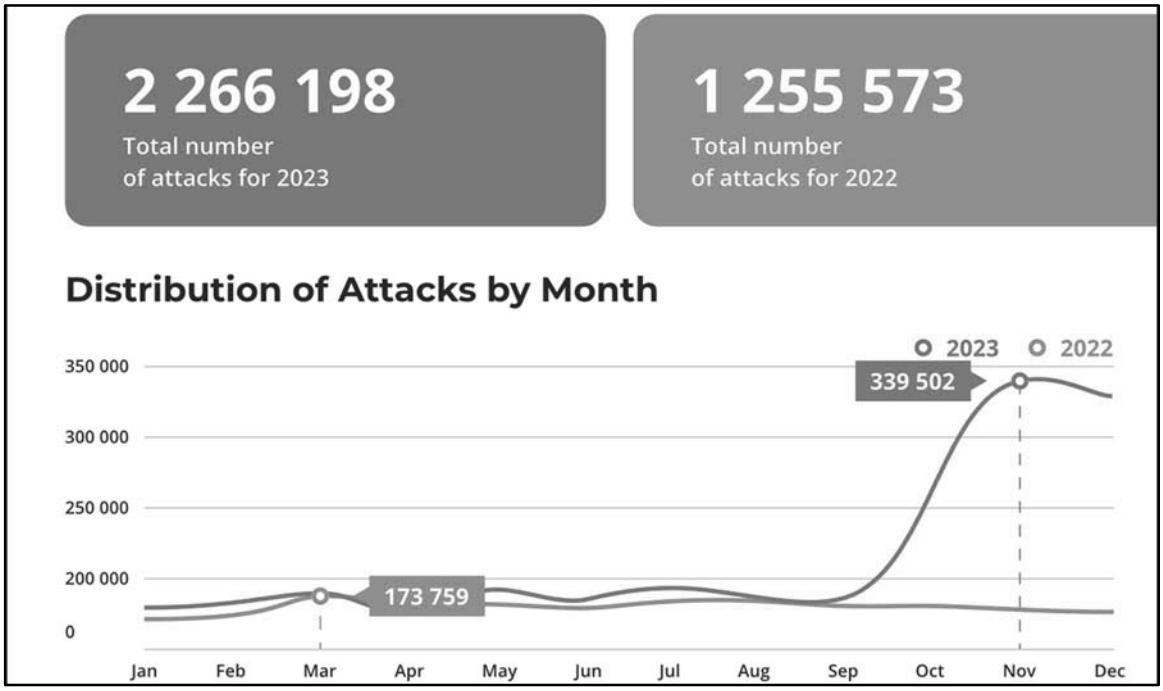

Рисунок 4 – Сравнение количества атак за 2022 и 2023 год [4]

Согласно отчету компании CLOUDFLARE, в 2023 году также наблюдался значительный рост мощности проводимых атак. В ходе самой большой DDoS-атаки, замеченной компанией в 2023-ем году, на сервис поступал 201 миллион запросов в секунду, что практически в 8 раз больше, чем в ходе самой крупной атаки в 2022 году [5]. Сравнительный отчет представлен на рисунке 5.

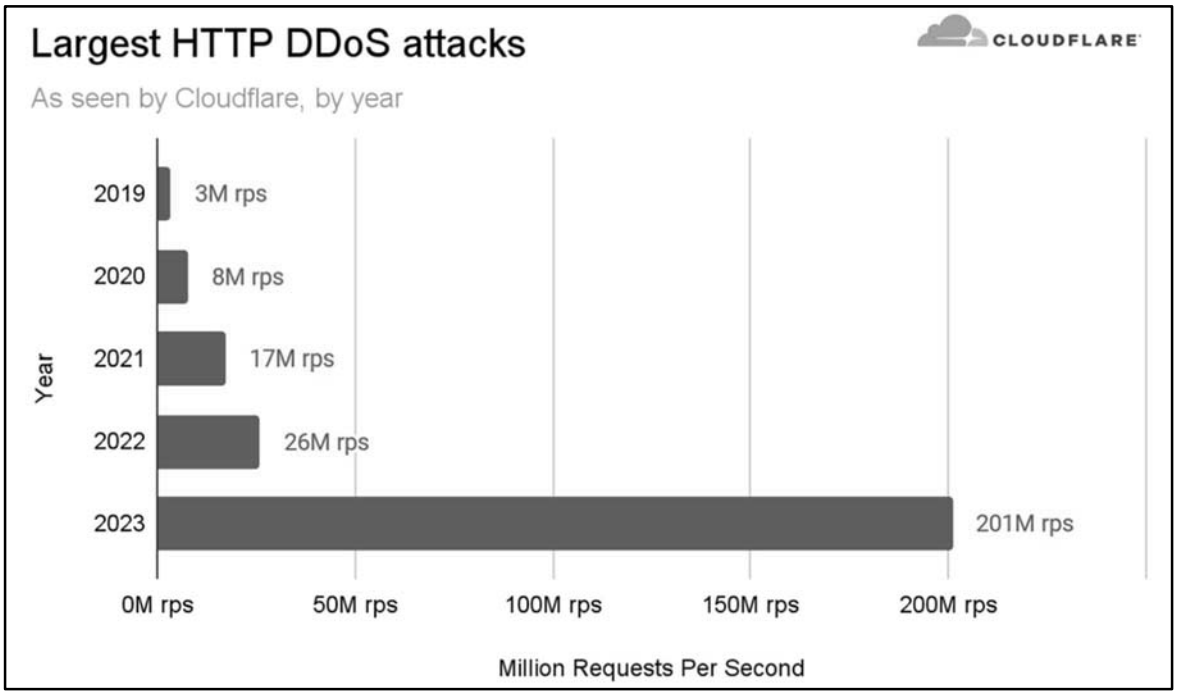

Рисунок 5 – Сравнение самых крупных атак по количеству запросов в секунду [5]

Исходя из приведенных отчетов был сделан вывод о необходимости развития новых систем защиты с использованием алгоритмов машинного обучения.

Постановка задачи

Целью выпускной квалификационной работы является разработка приложения для анализа сетевого трафика в режиме реального времени на основе методов машинного обучения. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области;
2) собрать набор данных;
3) выбрать модели машинного обучения;
4) реализовать выбранные модели машинного обучения;
5) спроектировать систему анализа трафика в реальном времени;
6) реализовать систему анализа интернет-трафика в реальном времени;

7) протестировать систему анализа интернет-трафика в реальном времени.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения, списка литературы. Объем работы составляет 66 страниц, объем списка литературы – 47 источников.

В первой главе производится обзор научной литературы.

Во второй главе рассматриваются алгоритмы машинного обучения Random Forest AdaBoost, SVM, LSTM, CatBoost. Проводится обзор и анализ наборов данных. Рассматривается задача классификации трафика, устройство сети, существующие способы анализа трафика, а также устройство анализаторов пакетов.

Третья глава посвящена определению основных требований к системе и проектированию. Была разработана диаграмма вариантов использования.

В четвертой главе описывается реализация выбранных моделей машинного обучения, анализатора пакетов и графического интерфейса пользователя.

Пятая глава посвящена тестированию разработанного приложения и оценке качества обученных моделей.

В приложениях содержатся графики A/B тестирования, исходный код анализатора пакетов, исходный код графического интерфейса приложения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Обзор научной литературы

«NetGPT: Generative Pretrained Transformer for Network Traffic» [6]

В данной работе авторы применяли подход больших предварительно обученных моделей для решения нескольких задач, связанных с анализом сетевого трафика. Такой подход должен был решить проблему использования множества небольших моделей для каждой отдельной задачи, обученных на специфичных наборах данных, которые могли быть недостаточно большими для успешного обучения моделей.

Для решения поставленной задачи авторами была предложена архитектура NetGPT. Архитектура NetGPT разделена на две части: предварительно обученную и донастроечную.

В процессе предварительного обучения сетевой трафик кодируется в скрытое пространство признаков. Для обеспечения генерализации модели, в предобученной части использовались неразмеченные записи. В процессе донастройки, увеличивается количество используемой информации путем перемешивания заголовков, сегментации семантики сетевых пакетов во время обучения, включения меток, необходимых для решения различных задач, а также генерации трафика на основе полученных данных.

Предложенная архитектура решает проблемы неоднородности реального трафика и разнообразия возможных задач с помощью использования предварительно обученной части и донастройки.

В качестве базовой модели для процесса предварительного обучения используется GPT-2. Для обучения генеративным задачам также используются модели семейства GPT.

На рисунке 6 представлена архитектура предварительно обученной части.

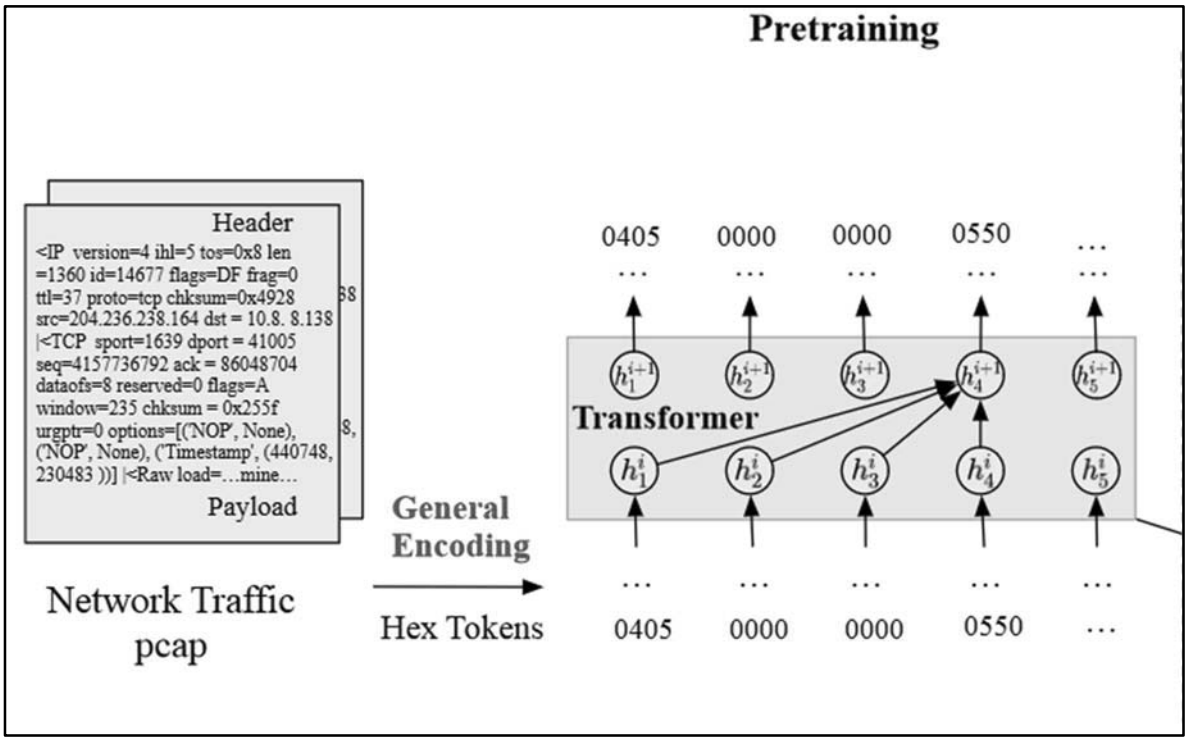

Рисунок 6 – Архитектура предварительно обученной части

На рисунке 7 представлена архитектура донастроечной части сети.

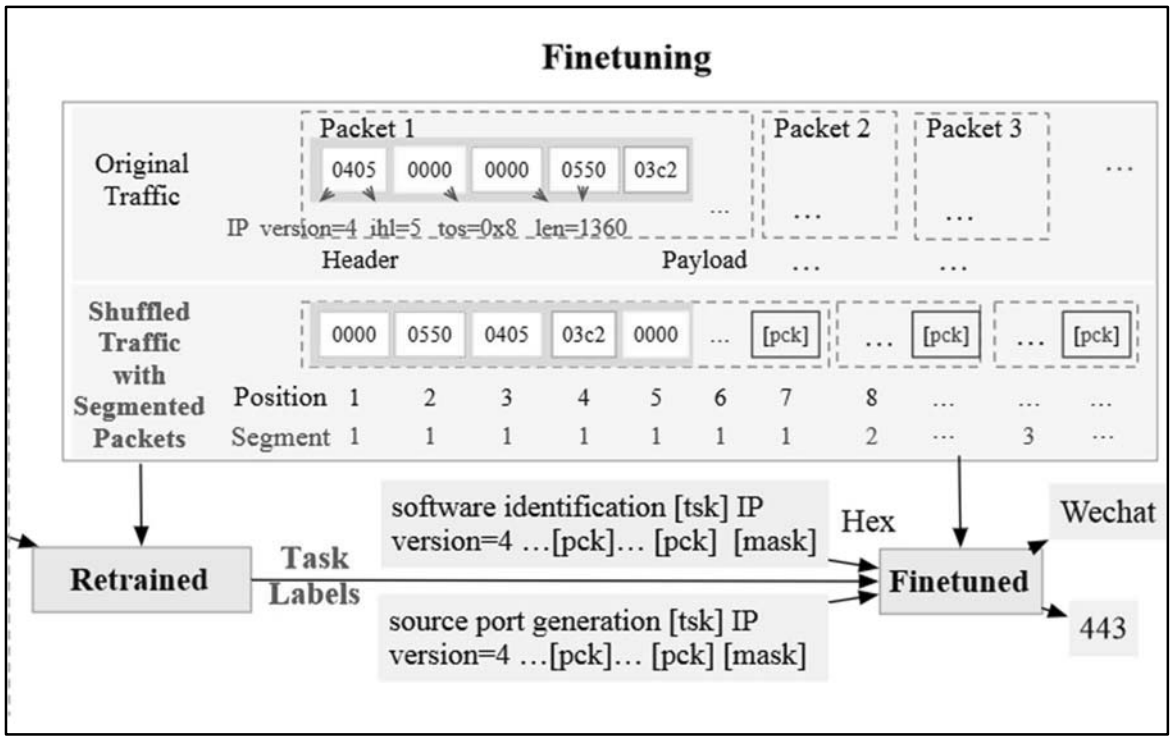

Рисунок 7 – Архитектура донастроечной части

Для обучения и тестирования модели были использованы 5 наборов данных: ISXW, DoHBrw, USTCTFC, PrivII, Cybermine. Распределение количества записей представлено на рисунке 8. Набор данных Cybermine не использовался для обучения и представлен только в тестировании. Были сформулированы 8 задач, которым модель должна была обучиться, среди них: обнаружение VPN и классификация приложений на наборе данных ISXW; обнаружение трафика DNS-over-HTTPS и генерация запросов DoH для набора данных DoHBrw; обнаружение сетевых атак и идентификация приложений в наборе данных USTCTFC; обнаружение кибермайнинга и идентификация криптовалют для набора Cybermine.

Dataset	ISXW	DoHBrw	USTCTFC	PrivII	Cybermine
# packets	1,338,300	77,149,018	1,010,534	38,250,867	7,862
# flows	1,262	2,497	5,039	36,926	2,617

Рисунок 8 – Используемые наборы данных

Для тестирования использовались метрики точности и макро f1 меры, а также дивергенция Йенсена – Шеннона. Валидация и тестирование проводились на случайных выборках пакетов размером 5 000. Для задач классификации трафика модель обучалась в течении 50 эпох. Для задач генерации трафика модель обучалась в течении 10 эпох.

Средняя точность архитектуры NetGPT на задаче анализа пакетов составила 98,56%, средняя точность на задаче анализа сетевых потоков составила 94,6%. Архитектура ET-Bert на тех же задачах достигает точности 98,25% и 90,8%.

Для анализа эффективности моделей на задачах генерации трафика использовалась метрика JSD. Метрика JSD может отражать достоверность полученных данных в задачах генерации, сравнивая реальные данные и полученные в процессе генерации для распределения, соответствующего предметной области. Среднее значение метрики JSD для модели NetGPT составило 0,0406, для модели GPT-2 составило 0,0417.

Исходя из полученных результатов, авторы делают вывод о высокой эффективности реализованной модели на задачах генерации трафика. NetGPT показывают лучшую среднюю производительность и превосходит GPT-2 на большинстве наборов данных, демонстрируя эффективность применения техники перемешивания при обучении задачам генерации. Высокие результаты на наборе данных Cybermine показывают высокую генерализацию модели и ее способность к решению сопутствующих задач.

Авторы также делают вывод о высокой степени соответствии сгенерированного трафика с исходным набором данных. Распределение заголовочных полей трафика, полученного с помощью NetGPT, совпадает с реальными данными. Результаты генерации представлены на рисунках 9–10.

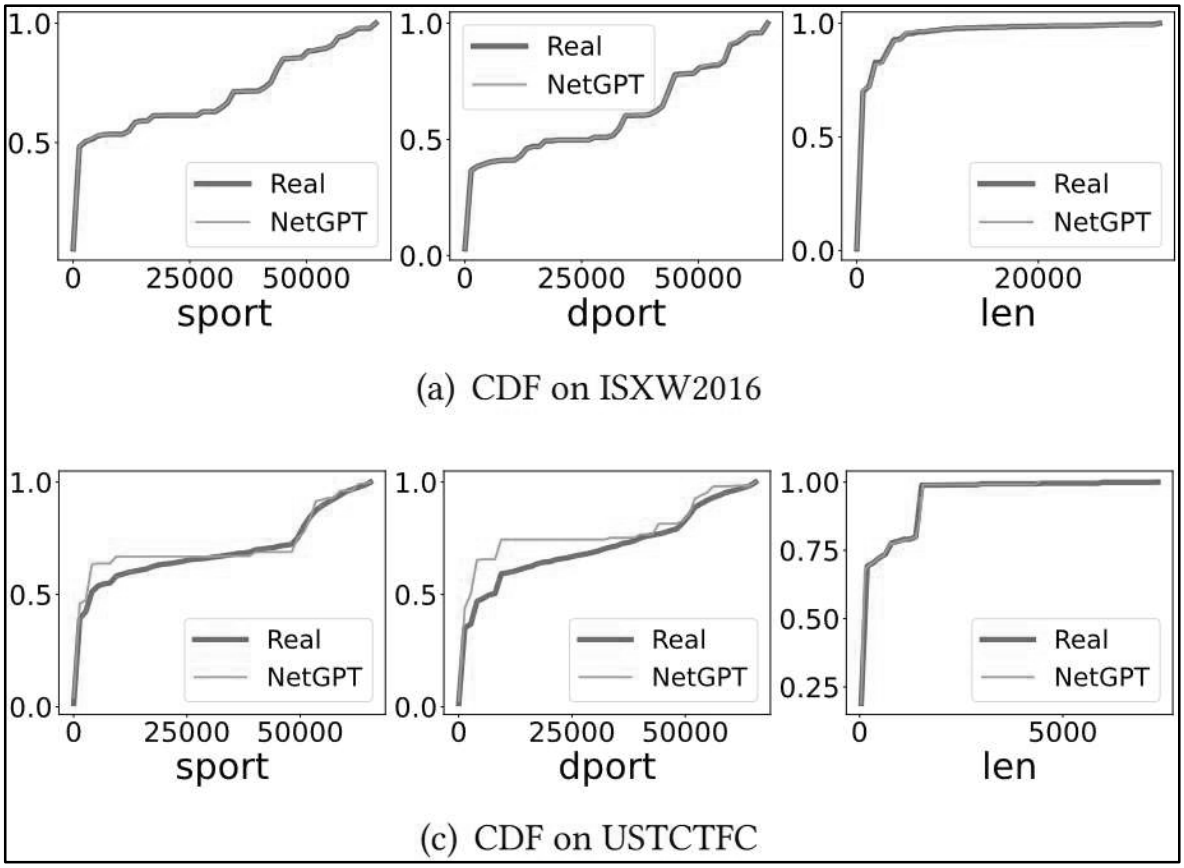

Рисунок 9 – Распределение сгенерированных заголовков NetGPT для наборов ISXW2016 и USTCTFC

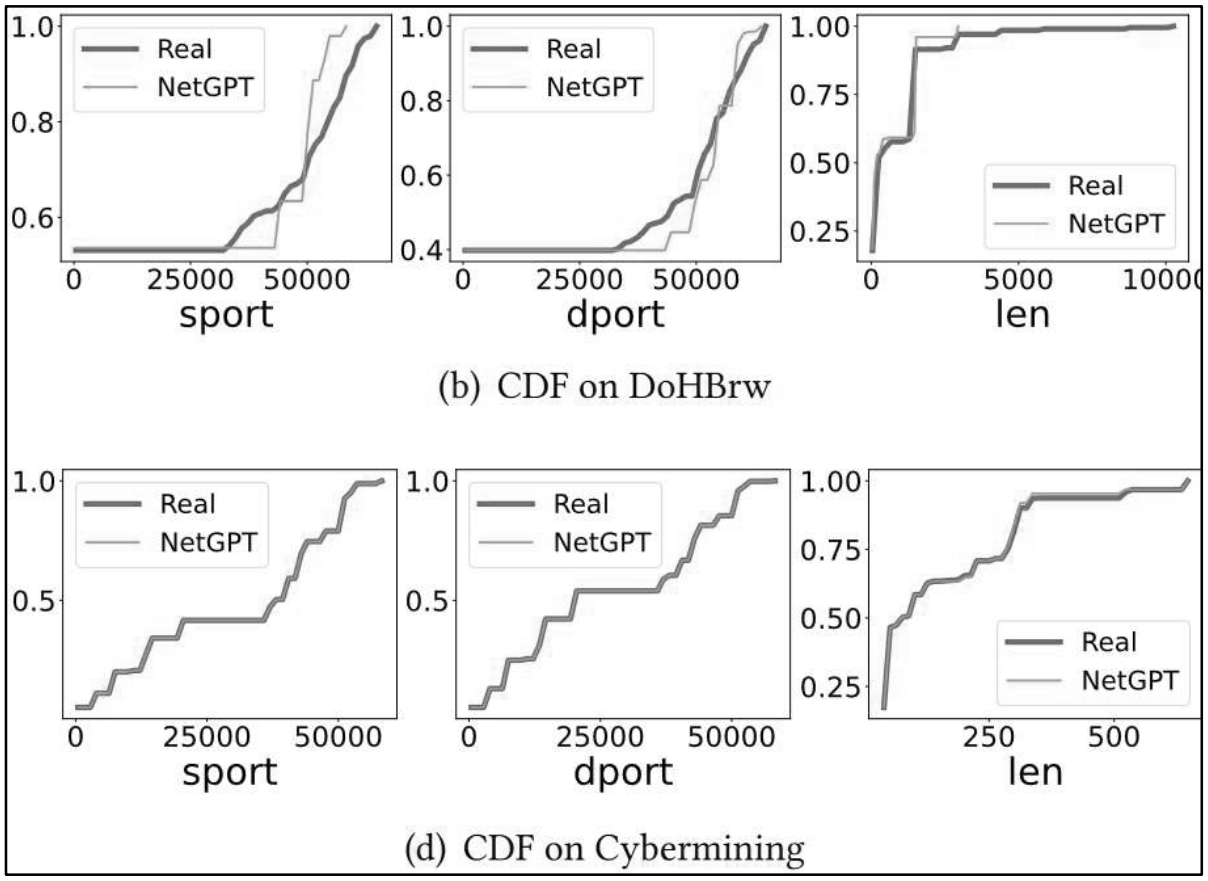

Рисунок 10 – Распределение сгенерированных заголовков NetGPT для наборов данных DoHBrw и Cyberminig

«OMINACS: Online ML-based IOT network attack detection and classification system» [7]

В данной работе авторы пытаются решить проблему быстрого устаревания обученных моделей, используемых в системах обнаружения угроз. Для решения этой проблемы был предложен способ обучения в реальном времени, объединяющий несколько моделей машинного обучения. Схема работы системы представлена на рисунке 11.

Система состоит из четырех компонентов. Первый компонент реализует алгоритм потокового машинного обучения на основе адаптивных деревьев Хоффдинга, использующихся для бинарной классификации. В зависимости от результата классификации записи передаются на вход второго и третьего компонента системы. Второй компонент реализует алгоритм потокового машинного обучения, используемый для классификации типа атаки.

Третий компонент осуществляет проверку данных, классифицированных как обычный трафик с помощью модели глубокого обучения. На вход третьего компонента поступают данные, отмеченные как обычный трафик, из первого, второго и четвертого компонентов системы. В четвертом компоненте производится проверка соответствия классов атак с помощью множества моделей. Каждая из моделей определяет достоверность присвоенного класса, корректен ли присвоенный класс или это ошибка. Для классификации на данном этапе используются ансамблевые модели.

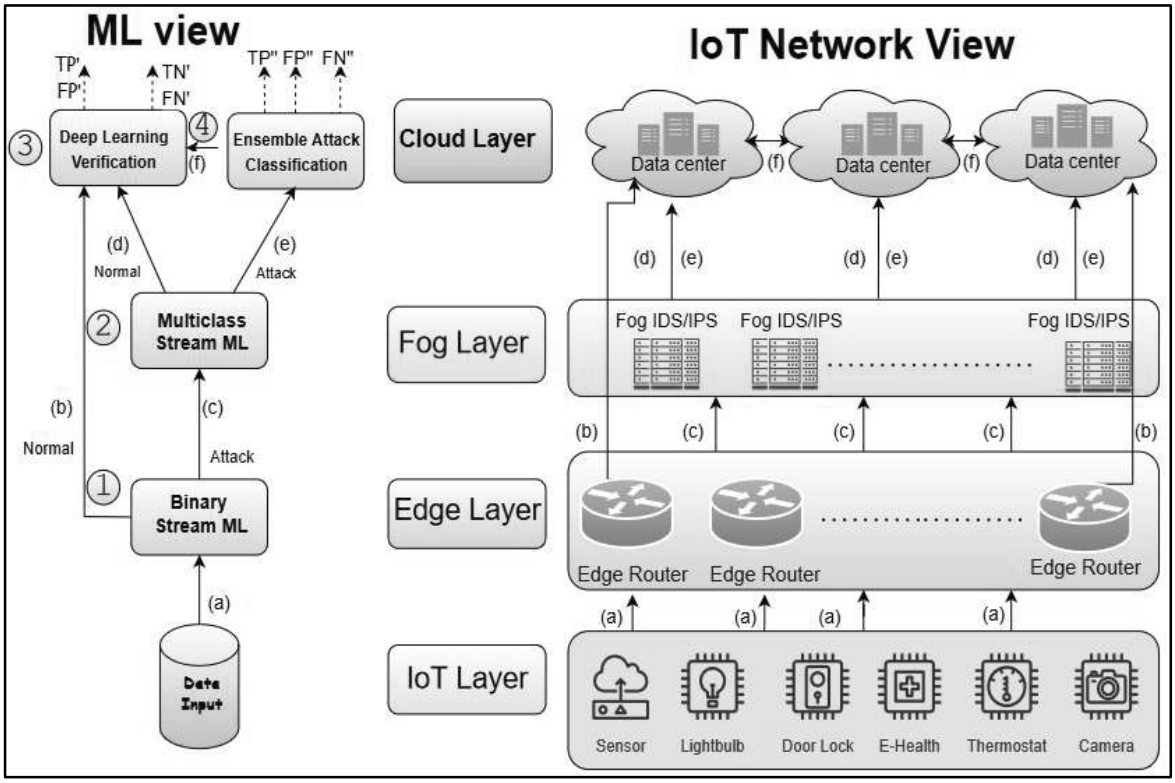

Рисунок 11 – Схема системы OMINACS

Для оценки качества обученных моделей использовались следующие метрики: Accuracy, Precision, True Positive Rate, False Alarm Rate, F1. Для обучения и тестирования были использованы наборы данных BoT-IOT, TON-IOT, CIC-IOT-2022. На рисунке 12 представлены результаты обучения и тестирования.

На рисунке 12 результаты обучения и тестирования, система достигает точности выше 90% при частоте ложных срабатываний меньше 3% на

всех представленных наборах данных. Самый высокий результат по метрике Accuracy был достигнут на наборе данных BOT-IOT и составил 98,91%. Что говорит о высокой эффективности разработанной системы.

Dataset	ACC%	Prec%	TPR%	FAR%	F1%
BOT-IOT	98.91	96.76	99.88	03.24	98.46
TON-IOT	94.33	90.45	99.83	9.55	94.35
IOT-22	96.55	96.97	99.89	03.03	97.16

Рисунок 12 –Результаты тестирования

«Comparative analysis of machine learning techniques for network traffic classification» [8]

В данной работе авторы сравнивают работу четырех алгоритмов машинного обучения на задаче классификации интернет-трафика, используя набор данных NIMS. Набор данных содержит информацию о 6 типах сетевых приложений: LFWD, RFWD, SCP, SFTP, SHELL, X11. Для каждого приложения было получено одинаковое количество записей. Общее количество записей в наборе данных составило 14 681.

Авторы выбрали для анализа четыре алгоритма машинного обучения: SVM, наивный байесовский классификатор, K-NN, C4.5 решающее дерево. Для отбора признаков использовались методы Information Gain, ReliefF, Symmetrical Uncertainty, Gain Ratio. Для измерения качества обученных моделей использовались матрицы возмущений, Accuracy, Precision, Recall и мера f1.

Без отбора признаков были получены результаты, представленные на рисунке 13. В этом случае наиболее точный результат был получен с помощью деревьев решений C4.5 с метрикой точности 98,25% и наименьшим временем работы.

Classifier	Accuracy	Precision	Recall	F-Measure	Runtime(s)
NB	85.64	0.873	0.871	0.858	0.38
KNN	98.12	0.993	0.983	0.96	0.18
SVM	94.47	0.934	0.951	0.952	1.35
C45	98.25	0.975	0.984	0.983	0.16

Рисунок 13 – Результаты обучения моделей

В ходе отбора признаков из набора данных NIMS были выделены 15 самых значимых. На рисунке 14 отражены отобранные признаки и методы, с помощью которых признаки были получены.

FS Method	Feature Names	Feature Values/Rankings
CFS	{maxbpktl, stdbpktl, stdfpktl, maxfpktl, stdfiat, meanfpktl, stdbiat, meanfiat, meanbkptl, meanbiat}	{0.383, 0.363, 0.327, 0.326, 0.325, 0.318, 0.310, 0.305, 0.298, 0.269}
GR	{maxfpktl, maxbpktl, minfiat, meanbkptl, meanfpktl, stdbpktl,maxfiat, maxbiat, stdfpktl, totalfvol}	{0.937,0.588,0.483,0.453,0.442, 0.403,0.353,0.348,0.334,0.293}
IG	{maxbiat, maxfiat, meanfpktl, stdbpktl, meanbkptl, stdfpktl, minfiat, meanfiat,minbiat, stdbiat}	{2.476, 2.475,1.992, 1.960, 1.724, 1.610, 1.572, 1.428, 1.425, 1.410}
RELIEFF	{meanfpktl, stdbpktl, maxfpktl, stdfpktl, maxfiat, maxbpktl, durattion, maxbiat, stdbiat, stdfiat, meanbkptl}	{0.203, 0.181, 0.174, 0.169, 0.153, 0.136, 0.127, 0.122, 0.112, 0.111}
SU	{maxfpktl, meanfpktl, meanbkptl, minfiat, stdbpktl, maxfiat, maxbiat, maxbpktl, stdfpktl, stdbiat}	{0.617, 0.562, 0.54, 0.539, 0.526, 0.515, 0.510, 0.455, 0.435, 0.370}

Рисунок 14 – Отбор признаков с помощью разных методов

После отбора признаков классификаторы были вновь обучены, результаты обучения до и после отбора признаков представлены на рисунке 15.

При отборе значимых признаков, точность и скорость работы всех используемых алгоритмов возросла. Лучшим алгоритмом остались деревья решения C4.5 достигнувшие точности 99,81% и скорости выполнения 0,07 секунд.

Classifier	Accuracy	Precision	Recall	F-Measure	Runtime(s)
NB	85.64	87.3	87.1	85.8	0.38
NB-FS	87.34	88.8	89.3	89.4	0.16
KNN	98.12	99.3	98.3	96	0.18
KNN-FS	99.72	99.7	99.7	99.7	0.09
SVM	94.47	93.4	95.1	95.2	1.35
SVM-FS	95.58	92.3	92.7	91.2	0.65
C45	98.25	97.5	98.4	98.3	0.16
C4.5-FS	99.81	99.8	99.8	99.8	0.07

Рисунок 15 – Результаты обучения моделей до и после отбора признаков

«Detecting denial of service attacks using machine learning algorithms» [9]

В статье рассматриваются два подхода к идентификации DDoS-атак. Первый способ заключается в построении математической модели, которая определяет отношение между временем поступления запросов и пропускной способностью с последующим анализом. В качестве моделей машинного обучения использовались логистическая регрессия и наивный байесовский классификатор.

Математическая модель использует две характеристики для классификации атак: скорость соединения и пропускная способность. Скорость соединения – это характеристика, отражающая сколько данных может быть передано по каналу связи. Пропускная способность – количество данных, которые были успешно переданы от источника до назначения.

Для тестирования модели использовался набор данных CAIDA 2007. Каждые 10 секунд на вход модели подавались данные с 20 090 различных IP адресов.

Для обучения моделей машинного обучения использовался набор данных CAIDA 2007. Набор данных был разделен для обучения 80%, тестов 20% и валидацию 10% набора данных.

Для оценки качества предложенной математической модели использовалась метрика $Miss\ Rate$, которая считается по формуле (1):

$$Miss\ Rate = \frac{False\ Negative}{False\ Negative + True\ Positive}, \quad (1)$$

где $False\ Negative$ – количество записей, ошибочно классифицированных как негативные;

$True\ Positive$ – количество записей, корректно классифицированных как положительные.

Значение метрики $Miss\ Rate$ на тестовом наборе данных составило 0,025.

Для оценки качества обучения моделей использовались метрики Accuracy, Mean Absolute Error (MAE), Recall. В результате обучения точность модели логистической регрессии составила от 99,83%, точность классификатора наивного байеса составила 98,67%. Значение метрики MAE для классификатора наивного байеса составило 0,0163, для модели логистической регрессии 0,0017. Значение метрики Recall составило 0,998 для модели логистической регрессии и 0,99 для наивного байесовского классификатора. Исходя из этих результатов был сделан вывод, что модель логистической регрессии лучше справляется с задачей классификации трафика.

Выводы по первой главе

В ходе анализа предметной области был проведен обзор литературы. Было выявлено, что исследования в области применения алгоритмов машинного обучения для построения современных систем обнаружения вторжений очень актуальны. Рассмотренные работы демонстрируют высокую эффективность применения алгоритмов машинного обучения для задачи анализа интернет-трафика.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1. Задача классификации трафика

Задача классификации в машинном обучении формулируется следующим образом: с помощью заранее заданного множества объектов, разделенных с помощью признаков на классы, требуется построить алгоритм, способный на определение класса произвольного объекта из исходного множества данных или нового объекта [10].

Задача классификации сетевого трафика может быть поставлена как: определение класса трафика на основе входных характеристик сетевого трафика. В качестве входных характеристик могут выступать как данные пакетов, так и различные частотные характеристики. В качестве выходных данных может использоваться идентификатор конкретного приложения, ответственного за генерацию трафика, или идентификатор вида трафика [11].

Устройство сети

Для передачи данных между сетевыми устройствами используется сетевая модель TCP/IP, состоящая из 4 уровней: канальный; межсетевой; транспортный; прикладной.

Прикладной уровень относится к программам, которым нужен стек технологий TCP/IP для взаимодействия друг с другом. Это уровень, на котором пользователи зачастую взаимодействуют с системами подобными электронной почте, платформами обмена сообщений. Содержит в себе множество протоколов: HTTP, HTTPS, SSH, NTP, FTP, SMTP и другие.

Транспортный уровень ответственен за предоставление надежного подключения между источником и получателем. На этом уровне данные разделяются на пакеты и нумеруются для создания последовательностей. Транспортный уровень определяет сколько данных должно быть отправлено, куда они должны быть отправлены и скорость соединения. Транспортный уровень обеспечивает безошибочную отправку данных в последовательностях и уведомляет о получении отправленных пакетов данных получателем.

Сетевой уровень ответственен за отправление сетевых пакетов и управление их передвижением по сети для подтверждения успешной передачи данных получателем [12].

Канальный уровень – описывает процесс обмена информацией на уровне сетевых устройств и определяет информацию, передаваемую от одного устройства другому. Также этот уровень вычисляет максимальное расстояние, на которое возможно передать пакеты, частоту сигнала, задержку ответа. Главным протоколом канального уровня является Ethernet [13].

2.2. Методы классификации трафика

Согласно классификации CISCO, существует два основных метода классификации трафика: метод, основанный на анализе содержимого интернет-пакетов; метод, основанный на статистическом анализе поведения трафика. Классификация представлена на рисунке 16.

Самым распространенным методом классификация является анализ содержимого интернет-пакетов. Данный метод подразделяется на базовый и продвинутый анализ содержимого.

Базовый подход для классификации основывается на анализе информации IP заголовков. В качестве данных, могут использоваться IP-адрес, MAC-адрес, а также используемые для передачи протоколы.

Данный подход является устаревшим и не позволяет классифицировать данные большинства приложений. Недостаток данного подхода заключается в том, что многие современные приложения не используют стандартные порты для передачи информации, а некоторые приложения обфусцируют свой трафик и выставляют себя за другое приложение. Таким образом, классификация трафика с использованием базового подхода может быть недостоверной.

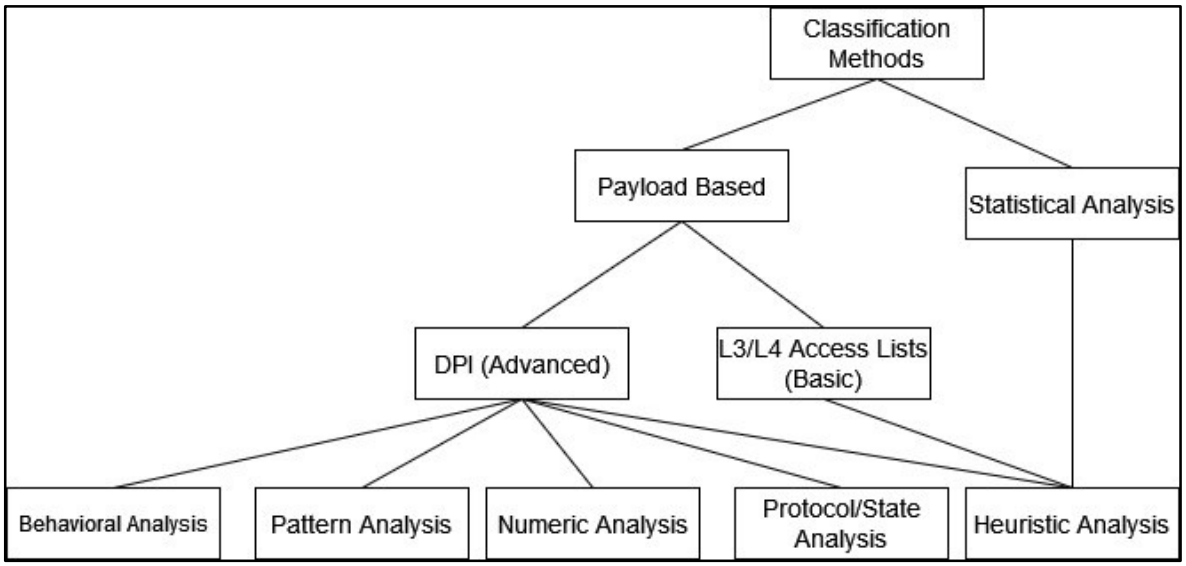

Рисунок 16 – Методы классификации трафика

Более продвинутым подходом для классификации является подход Deep Packet Inspection (DPI). Большинство DPI решений предлагают подход анализа сигнатур для анализа и верификации различных приложений. Сигнатуры – это уникальные отпечатки, соотносящиеся с конкретным приложением. Таким образом, из каждого приложения формируется уникальная сигнатура, характеризующая приложение. Из таких сигнатур формируется база данных. Классифицирующее приложение сравнивает полученный трафик с базой данных и определяет, какому приложению принадлежит трафик [14].

Анализатор пакетов

Анализатор пакетов (сниффер пакетов) – программное обеспечение или физическое оборудование, работающее в компьютерной сети, которое пассивно получает данные канального уровня сети в виде фреймов. Анализатор пакетов перехватывает данные, которые были адресованы другим устройствам, сохраняя их для дальнейшего анализа.

Данное программное обеспечение может быть использовано сетевыми администраторами для наблюдения и диагностирования ошибок сетевого трафика. Используя информацию, перехваченную анализатором, адми-

нистратор может идентифицировать аномальные пакеты, использовать полученную информацию для определения уязвимых мест сети и поддерживать эффективную работу сети [15].

2.3. Методы машинного обучения

Для достижения поставленной цели были выбраны следующие алгоритмы машинного обучения и архитектуры нейронных сетей: LSTM, AdaBoost, Random Forest, CatBoost, SVM. Общее описание выбранных алгоритмов представлено ниже.

Рекуррентные сети и LSTM

Рекуррентная нейронная сеть (RNN) – это архитектура искусственных нейронных сетей, специально адаптированная для работы с данными в форме последовательностей, таких как временные ряды. Такие сети предназначены для решения задач, в которых входные данные зависят от предыдущих данных, требуя сохранения информации о предыдущих состояниях сети [16].

В архитектуре такой нейронной сети каждое узел получает на вход состояние из предыдущего узла и передает результат своей работы на следующее звено. Пример архитектуры такой сети представлен на рисунке 17.

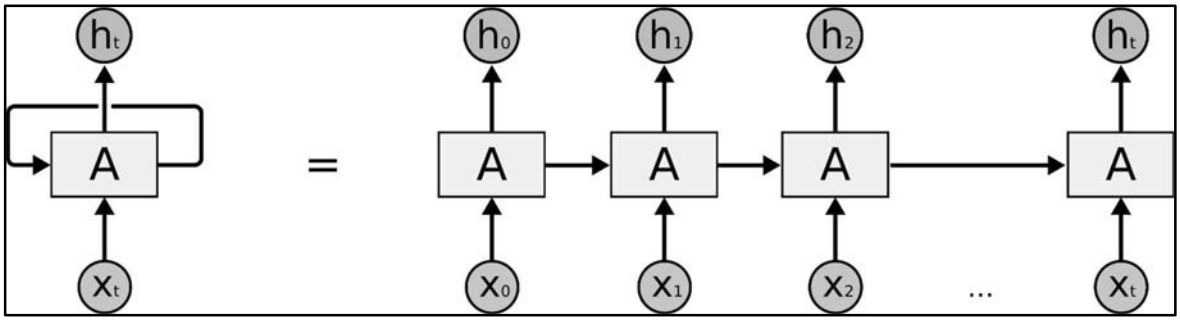

Рисунок 17 – Схема рекуррентной сети

Однако в таких нейронных сетях контекст предыдущей информации быстро исчезает. Поэтому для задач, где для принятия решений требуется больше информации из предыдущих состояний такая сеть не подходит.

Для решения этой проблемы была предложена архитектура Long Short-Term Memory. Long Short-Term Memory (LSTM) – подкласс рекуррентных нейронных сетей (RNN), способных обучаться долгосрочным зависимостям. Пример архитектуры LSTM представлен на рисунке 18.

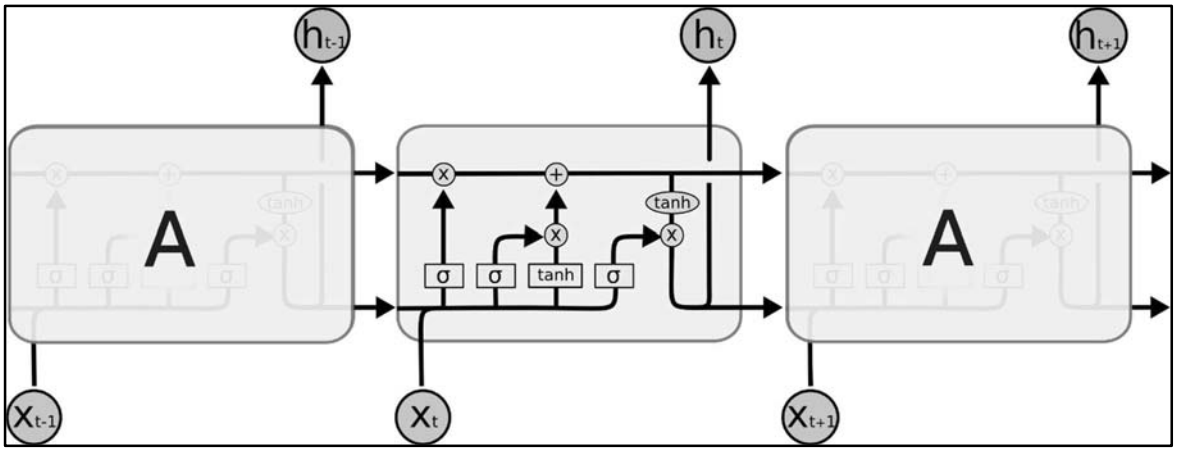

Рисунок 18 – Схема LSTM сети

Основным отличием LSTM сети от RNN является устройство узла. В RNN сетях для получения результата для текущего узла выходные данные предыдущей узла складываются с входными данными текущей узла, после чего к данным применяется функция активации tanh. В LSTM модуль состоит из нескольких частей. Основная составляющая модуля – это линия состояния узла. Информация поступает на эту линию из различных слоев сети.

В начале работы LSTM узла происходит конкатенация поданных на вход текущей ячейки данных и выходного значения с прошлого состояния.

Первый слой сети – слой фильтра забывания. На этом слое с помощью сигмоидной функции активации определяется, какая информация будет отброшена: значение 1 будет значить, что данные требуется оставить; значение 0 будет значить, что данные требуется забыть.

Следующий слой определяет информацию, которая будет добавлена в состояние узла. Данный слой состоит из двух этапов. На первом этапе с помощью сигмоидной функции активации определяются обновляемые значения. На втором этапе с помощью функции активации tanh создается вектор значений кандидатов, которые могут быть добавлены в линию состояния.

После чего данные, прошедшие через сигмоидную и tanh функцию активации, комбинируются для обновления состояния ячейки.

Следующий слой определяет выходную информацию ячейки. Для получения выходной информации данные из линии состояния узла, прошедшие через функцию активации tanh, комбинируются с прошедшими через сигмоидную функцию активации входными данными, получая выходной сигнал ячейки и состояние для следующего модуля LSTM сети.

Таким образом, модель LSTM позволяет сохранять необходимый для задач контекст дольше и преодолевать проблему долгосрочных зависимостей, достигая лучших результатов, чем модели, основанные на RNN архитектуре [17].

Ансамблевые методы

Ансамблевое обучение – это техника машинного обучения, которая увеличивает точность и достоверность предсказаний модели путем объединения предсказаний нескольких моделей. Цель такого подхода – уменьшить ошибки или смещения, которые могут присутствовать в отдельных моделях. При учете различных слабостей и сильных сторон простых моделей, улучшается общая производительность обученной системы. Также такой подход позволяет справляться с различными погрешностями данных.

Можно выделить три категории ансамблевых алгоритмов.

1. Stacking. В данной категории используются предсказания нескольких различных моделей для построения новой модели. Новая модель используется для предсказаний на тестовом наборе данных. Суть метода можно описать так, из каждой используемой в ансамбле модели на основе обучающей выборки создается набор признаков, на основе полученных признаков обучается итоговая ансамблевая модель.

2. Bagging или же Bootstrap Aggregation. В данной категории методов множество базовых моделей независимо и параллельно обучаются на уникальных выборках исходного набора данных, полученных с помощью тех-

ники Bootstrap, результат предсказания определяется голосованием. Bootstrap – это технология семплирования, в которой создаются выборки оригинального набора данных с повторениями. Размер полученных выборок соответствует размеру оригинального набора данных [18]. Суть метода можно описать так, генерация выборок исходного набора размером B из исходного набора данных размера N осуществляется путем случайного выбора с повторениями элементов в каждом из B раз [18].

3. Boosting. Boosting – это последовательная техника обучения ансамблевых моделей, в которой каждая следующая базовая модель обучается на ошибках предыдущей. Алгоритм обучения можно описать следующим образом, на каждом шаге обучения базовой модели увеличиваются веса некорректно классифицированных записей исходного набора данных. Следующая модель старается исправить ошибки предыдущей модели. Финальная модель представляет собой взвешенное среднее всех базовых моделей. Идея метода состоит в том, что базовые модели плохо работают на общем наборе данных, но могут хорошо справляться с частью набора, объединив несколько базовых моделей возможно увеличить общую эффективность ансамблевой модели [18].

Support Vector Machines (SVM)

Метод опорных векторов является одним из классических алгоритмов машинного обучения, который применяется для решения задач классификации и регрессии. Основная идея метода заключается в построении гиперплоскости, разделяющей объекты выборки оптимальным способом. Алгоритм работает в предположении, что чем больше расстояние между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет средняя ошибка классификатора [20].

Демонстрация работы метода представлен на рисунке 19.

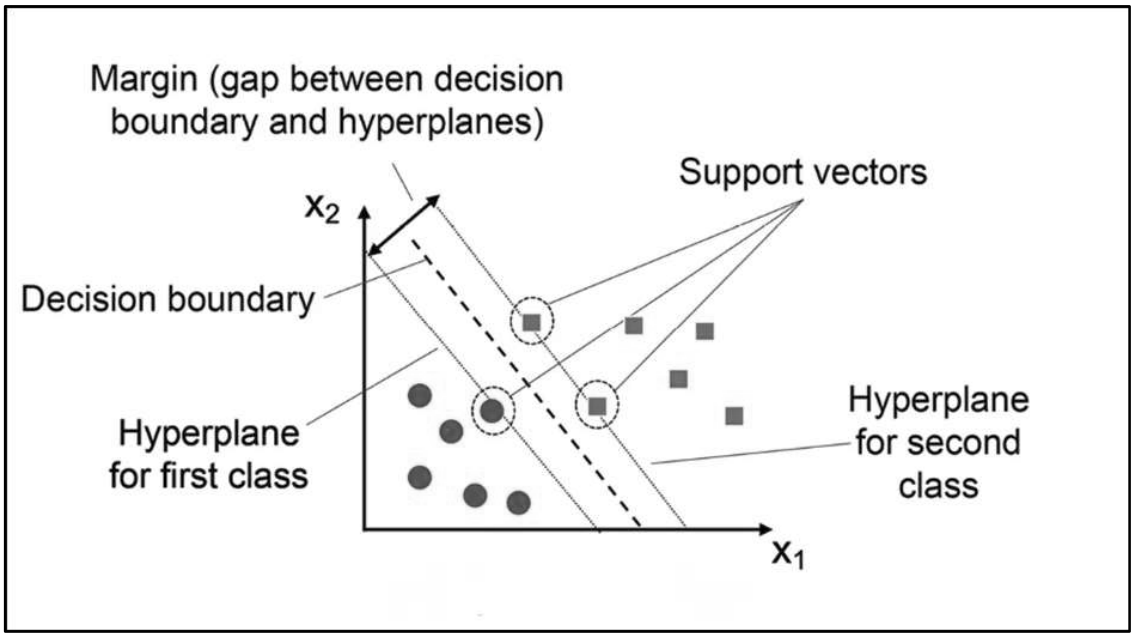

Рисунок 19 – Суть метода SVM [21]

Гиперплоскость – граница решений, которая разделяет данные на разные классы в многомерном пространстве. В N-мерном пространстве гиперплоскость имеет $N - 1$ размерность. С помощью гиперплоскости данные разделяются на два класса, оказавшись с одной или другой стороны плоскости.

Зазор – расстояние между гиперплоскостью и ближайшими точками данных для каждого класса. Зазор является мерой измерения качества разделения классов в пространстве признаков.

Опорные вектора – данные, находящиеся ближе всего к границе решений. Данные точки важны, потому что они определяют позицию и направление гиперплоскости, а также оказывают большое значение на точность классификации.

Алгоритм SVM определяет оптимальную гиперплоскость, которая максимизирует зазор между классами. Алгоритм SVM может работать как с линейно разделимыми данными, так и нелинейно разделимыми, проецируя данные в пространство большей размерности, где классы становятся разделимы гиперплоскостью [21].

CatBoost

CatBoost (Categorial Boosting) – библиотека с открытым исходным кодом, реализующая алгоритм градиентного бустинга на основе деревьев решения. Библиотека разработана компанией Яндекс. Первая версия библиотеки была представлена в 2017 году. На данный момент библиотека используется в проектах Яндекс, связанных с машинным обучением, а также во многих других компаниях, среди них: CERN, Cloudflare, Careem taxi.

Среди достоинств библиотеки отмечают: отсутствие необходимости в подборе параметров; поддержку категориальных признаков; наличие версии для GPU; высокую точность; быстрое время работы [22].

Повышение точности моделей достигается с помощью обработки категориальных признаков и комбинирования признаков. Обработка категориальных признаков заключается в замене категорий средним значением для класса на случайных измененных выборках данных. Комбинирование признаков позволяет объединять несколько категориальных признаков в один, что позволяет более эффективно обрабатывать категориальные признаки и терять меньше информации [23].

2.4. Наборы данных

Для обучения и тестирования были выбраны следующие наборы данных: CIC-IDS2017 [24]; CSE-CIC-IDS2018 [25]. Представленные наборы данных содержат информацию о нормальном трафике, а также информацию о распространенных современных атаках. Собранные данные отображают реальную структуру трафика. Для повышения достоверности наборов данных, в них был включен сгенерированный реалистичный фоновый трафик. Фоновый трафик был получен с помощью B-Profile системы. Система способна моделировать абстрактное человеческое поведение и создавать реалистичный нормальный трафик.

B-Profile

Система инкапсулирует поведение пользователей, используя различные методы статистики и машинного обучения. Инкапсулированные признаки – распределения количественных характеристик трафика для отдельных протоколов. Среди характеристик трафика выделяют размер пакетов, количество пакетов в потоке, определенные паттерны нагрузки, нагрузка, а также распределение времени запросов. Таким образом получают данные для симуляции трафика по следующим протоколам: HTTPS, HTTP, SMTP, POP3, IMAP, SSH, FTP.

CICFlowMeter

CICFlowMeter – приложение для генерации и анализа трафика, разработанное институтом UNB и используемое для получение полезной информации в наборах данных, распространяемых институтом.

Приложение может быть использовано для генерации бинаправленных сетевых потоков, получения более 80 статистических характеристик трафика, среди которых: длительность подключения, количество пакетов, количество байтов, длина пакетов и т.д. Приложение позволяет получать характеристики отдельно для исходящих и принимаемых потоков. Также приложение позволяет отбирать используемые признаки, добавлять новые признаки и контролировать время работы потоков.

Временем работы потока в приложении по умолчанию является 600 секунд.

Результат работы приложения сохраняется в формате CSV файла. CSV файл содержит такую информацию как: описание потока (идентификатор потока, ip-адрес источника, ip-адрес назначения, порт источника, порт назначения, сетевой протокол); признаки трафика [26].

На рисунке 20 представлены признаки пакетов, которые были получены с помощью CICFlowMeter. Первые шесть признаков содержат информацию из IP заголовков, в последней колонке содержится класс трафика.

No.	Feature	No.	Feature	No.	Feature
1	Flow ID	29	Fwd IAT Std	57	ECE Flag Count
2	Source IP	30	Fwd IAT Max	58	Down/Up Ratio
3	Source Port	31	Fwd IAT Min	59	Average Packet Size
4	Destination IP	32	Bwd IAT Total	60	Avg Fwd Segment Size
5	Destination Port	33	Bwd IAT Mean	61	Avg Bwd Segment Size
6	Protocol	34	Bwd IAT Std	62	Fwd Avg Bytes/Bulk
7	Time stamp	35	Bwd IAT Max	63	Fwd Avg Packets/Bulk
8	Flow Duration	36	Bwd IAT Min	64	Fwd Avg Bulk Rate
9	Total Fwd Packets	37	Fwd PSH Flags	65	Bwd Avg Bytes/Bulk
10	Total Backward Packets	38	Bwd PSH Flags	66	Bwd Avg Packets/Bulk
11	Total Length of Fwd Pck	39	Fwd URG Flags	67	Bwd Avg Bulk Rate
12	Total Length of Bwd Pck	40	Bwd URG Flags	68	Subflow Fwd Packets
13	Fwd Packet Length Max	41	Fwd Header Length	69	Subflow Fwd Bytes
14	Fwd Packet Length Min	42	Bwd Header Length	70	Subflow Bwd Packets
15	Fwd Pck Length Mean	43	Fwd Packets/s	71	Subflow Bwd Bytes
16	Fwd Packet Length Std	44	Bwd Packets/s	72	Init_Win_bytes_fwd
17	Bwd Packet Length Max	45	Min Packet Length	73	Act_data_pkt_fwd
18	Bwd Packet Length Min	46	Max Packet Length	74	Min_seg_size_fwd
19	Bwd Packet Length Mean	47	Packet Length Mean	75	Active Mean
20	Bwd Packet Length Std	48	Packet Length Std	76	Active Std
21	Flow Bytes/s	49	Packet Len. Variance	77	Active Max
22	Flow Packets/s	50	FIN Flag Count	78	Active Min
23	Flow IAT Mean	51	SYN Flag Count	79	Idle Mean
24	Flow IAT Std	52	RST Flag Count	80	Idle Packet
25	Flow IAT Max	53	PSH Flag Count	81	Idle Std
26	Flow IAT Min	54	ACK Flag Count	82	Idle Max
27	Fwd IAT Total	55	URG Flag Count	83	Idle Min
28	Fwd IAT Mean	56	CWE Flag Count	84	Label

Рисунок 20 – Признаки, полученные с помощью инструмента CICFlowMeter [27]

CIC-IDS2017

Общее количество записей, содержащихся в наборе, составляет 2 830 743 записи. Набор данных был собран в течении пяти дней, в каждый из дней проводились уникальные атаки.

В ходе работы по сбору наборов данных было создано поведение 25 пользователей, которые использовали HTTP, HTTPS, FTP, SSH и почтовые протоколы.

Для извлечения признаков из набора данных было использовано приложение CICFlowMeter. Таким образом каждая запись в наборе характеризуется с помощью 77 параметров.

Набор данных содержит информацию о типах трафика, представленных ниже.

1. Benign – 2 273 097 записей. Обычный трафик.
2. DDoS – 128 027 записей.
3. DoS slowloris – 5 796 записей. После установки TCP соединения со стороны атакующей машины не отправляется ACK пакет. Это позволяет оставить открытую сессию, которая использует ресурсы устройства [28].
4. DoS Slowhttptest – 5 499 записей. В ходе данной атаки злоумышленник отправляет множество HTTP Get запросов, занимая все доступные подключения на сервере [27].
5. DoS Hulk – 231 073 записей. Атака с помощью утилиты HULK. В ходе атаки создается уникальный и обфусцированный трафик. Вызывает ошибку отказ обслуживания [29].
6. DoS GoldenEye – 10 293 записей. Атака, использующая инструмент GoldenEye. Приложение использует техники KeepAlive и Cache-Control для создания и поддержания подключения с помощью сокетов. В ходе атаки все доступные сокеты сервера становятся недоступны [30].
7. Heartbleed – 11 записей. Атака, использующая уязвимость библиотеки OpenSSL, которая позволяла читать память устройств, защищенной устаревшей версией библиотеки OpenSSL. Таким образом могли быть скомпрометированы секретные ключи, используемые для идентификации проводников сервиса и для шифровки трафика, логинов и паролей пользователей, а также содержимое [31].
8. PortScan – 158 930 записей. Распространенная техника, используемая хакерами для обнаружения открытых портов устройства и получения статуса устройств. Также позволяет получить используемые устройством устройства защиты [32].
9. Bot – 1 966 записей. Botnet атака, проводящаяся с использованием программного пакета Ares. Ares – инструмент удаленного доступа, написанный на языке программирования Python [33].

10. FTP-Patator – 7 938 записей. Brute Force атака, использующая протокол FTP, с использование программного пакета Patator [34].

11. SSH-Patator – 5 897 записей. Brute Force атака по протоколу SSH с использование программного пакета Patator [34].

12. Web Attack-Brute Force – 1 507 записей. В ходе данной атаки, злоумышленник пытается получить защищенную информацию путем перебора [27].

13. Web Attack-XSS – 652 записей. В ходе данной атаки злоумышленник проникает на защищенные сайты с помощью приложения, отправляющего вредоносный код [27].

14. Web Attack-SQL Injection – 21 записей. Технология инъекции, используемая для атаки ориентированных на работу с данными приложений, при котором вредоносные инструкции SQL вставляются в поле ввода выполнения [27].

15. Infiltration – 36 записей. Злоумышленник получает несанкционированный доступ к сетевым системным данным [27].

CSE-CIC-IDS2018

Данный набор содержит информацию о семи сценариях сетевых атак. Каждая запись в наборе данных характеризуется 80 признаками, которые были получены с помощью приложения CICFlowMeter.

В ходе сбора данных поведение пользователей для генерации фонового трафика основывалось на использовании B-Profile системы. Основными протоколами пользователей были HTTP и HTTPS.

Набор состоит из 16 232 943 записей и содержит данные о 15 типах трафика, представленных ниже.

1. Benign – 13 484 708 записей. Обычный трафик.

2. HOIC – 686 012 записей. Атака проводится с использование программного обеспечения High Orbit Ion Cannon (HOIC). Данное программное обеспечение позволяет отправлять множество HTTP запросов для перегрузки веб серверов [35].

3. LOIC-HTTP – 576 191 записей. Атака по протоколу HTTP с использованием программного обеспечения Low Orbit Ion Cannon (LOIC) [36].

4. DoS Hulk – 461 912 записей.

5. Bot – 286 191 записей.

6. FTP-BruteForce – 193 360 записей.

7. SSH-BruteForce – 187 589 записей.

8. Infiltration – 161 934 записей.

9. DoS Slowhttptest – 139 890 записей.

10. DoS GoldenEye – 41 508 записей.

11. DoS slowloris – 10 990 записей.

12. LOIC-UDP – 1 730 записей. Атака с использованием программного обеспечения Low Orbit Ion Cannon (LOIC), использующая протокол UDP. В ходе атаки на сервер жертвы отправляется множество UDP пакетов на случайные порты устройства [36].

13. Brute Force-Web – 611 записей.

14. Brute Force-XSS – 230 записей.

15. SQL Injection – 87 записей.

Выводы по второй главе

Во второй главе были описаны такие пункты, как постановка задачи классификации трафика, предлагаемые к разработке ансамблевые методы машинного обучения, архитектура нейронной сети, классический алгоритм SVM, рассмотрение сетевой модели TCP/IP, определение анализатора пакетов, а также был проведен анализ исходных наборов данных.

3. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

3.1. Функциональные и нефункциональные требования

Функциональные требования

Функциональные требования – это перечень функций или сервисов, которые должна выполнять система, а также ограничения на данные и поведение системы [38].

Разрабатываемое приложение должно удовлетворять следующим функциональным требованиям:

1) пользователь должен иметь возможность выбрать предварительно обученную модель для анализа трафика;
2) приложение должно выводить информацию о результате классификации и ошибках пользователя;
3) приложение должно вести журнал работы;
4) пользователь должен иметь возможность выбрать сетевой интерфейс для анализа трафика в реальном времени;
5) пользователь должен иметь возможность сохранить полученные записи о работе приложения;
6) пользователь должен иметь возможность запускать и останавливать анализ трафика.

Нефункциональные требования

Нефункциональные требования определяют условия и среду выполнения функций (например, защита и доступ к БД, секретность и др.), они непосредственно не связаны с функциями, а отражают пользовательские потребности к выполнению функций. Они характеризуют принципы взаимодействия со средами или другими системами, а также учитывают время работы, защиту данных, а также стандарты качества для достижения отдельных показателей или атрибутов качества [38].

Разрабатываемое приложение должно удовлетворять следующим нефункциональным требованиям:

1) приложение должно быть разработано на языке Python;

2) в приложении должны быть реализованы пять алгоритмов машинного обучения;

3) интерфейс пользователя должен быть графическим.

3.2. Варианты использования приложения

Была разработана диаграмма вариантов использования системы, представленная на рисунке 21.

Диаграмма вариантов использования – диаграмма, на которой изображаются отношения между актерами и вариантами использования. Создание диаграммы вариантов использования преследует следующие цели: определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы; сформулировать общие требования к функциональному поведению проектируемой системы; разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей; подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями. Сам вариант использования представляет собой спецификацию общих особенностей поведения или функционирования моделируемой системы без рассмотрения внутренней структуры этой системы.

На диаграмме представлен главный актер – «Пользователь», который взаимодействует с системой. Пользователь имеет один основной вариант использования системы – «Запустить анализ трафика». Для того, чтобы запустить анализ трафика, пользователь должен выбрать предварительно обученную модель, а также выбрать режим работы. В приложении предусмотрена работа в двух режимах: режим реального времени; режим анализа локальных файлов. В зависимости от выбранного режима работы, пользователь должен выбрать сетевой интерфейс, данные с которого будут прослушиваться, либо выбрать файл, который будет анализироваться. После за-

пуска анализа трафика пользователь может остановить процесс работы приложения, а также сохранить файл, содержащий информацию о результатах работы приложения.

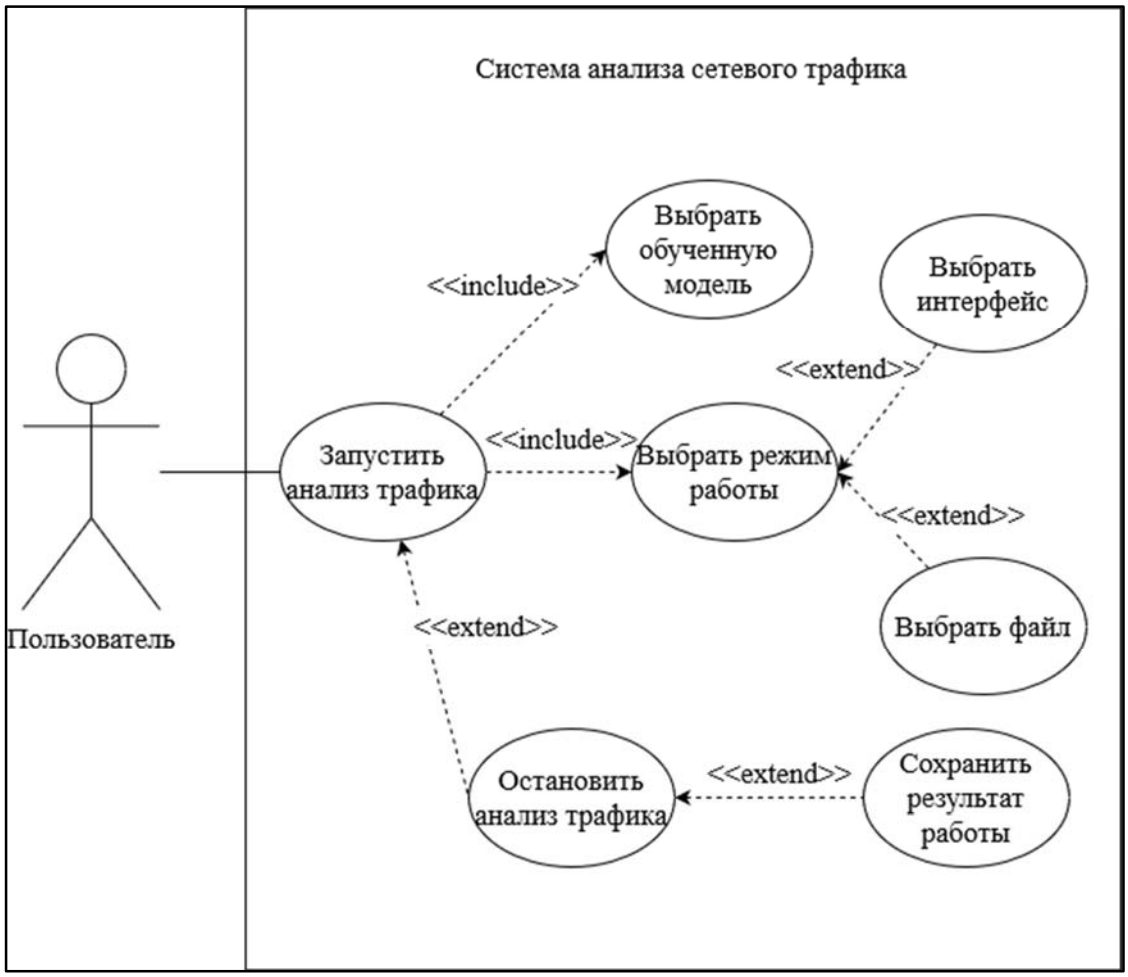

Рисунок 21 – Диаграмма вариантов использования

Выводы по третьей главе

В данном разделе были описаны функциональные и нефункциональные требования к системе. На основе этих требований была разработана диаграмма вариантов использования, отражающая взаимодействие пользователя с разрабатываемой системой.

4. РЕАЛИЗАЦИЯ

4.1. Среда выполнения и программные средства разработки

В качестве основного языка программирования для разработки был выбран язык Python версии 3.12.0. Для данного языка существует множество библиотек, упрощающих работу с данными и машинным обучением.

Для создания и обучения моделей машинного обучения была выбрана среда Google Colaboratory.

Google Colaboratory – облачный сервис, предоставляющий бесплатный доступ к вычислительным ресурсам GPU и TPU, работающий на основе Jupyter Notebook. Выбранная среда позволяет запускать код в отдельных ячейках, что позволяет быстро отлаживать работу разрабатываемых приложений. Также в Google Colaboratory предустановлены многие популярные библиотеки для работы с данными и машинным обучением [40].

Для реализации моделей машинного обучения были использованы библиотеки scikit-learn, catboost и PyTorch Lightning. Библиотека scikit-learn содержит готовые решения по реализации алгоритмов машинного обучения, наборы данных, а также методы для обработки данных и результатов моделирования. PyTorch Lightning – библиотека глубокого обучения основанная на фреймворке PyTorch. Библиотека предоставляет гибкий интерфейс для создания, обучения, тестирования и анализа обученных моделей глубокого обучения, позволяющий упрощать данные процессы [41].

Реализация анализатора пакетов выполнена с использованием библиотеки Scapy. Scapy – библиотека для интерактивной манипуляции интернет-пакетами, написанная на Python [42].

В качестве среды разработки выбрана программа Visual Studio Code [43].

4.2. Предобработка набора данных

Исходные наборы данных были предварительно обработаны для повышения эффективности работы моделей. Предварительная обработка набора состояла из нескольких этапов.

Первоначально были изменены названия признаков в используемых наборах данных. Были удалены дублирующие признаки, признаки, содержащие единственное значение, и записи, содержащие пустые значения.

Наборы данных содержат несбалансированное количество записей на класс (рисунок 22–23). Для повышения качества моделей была проведена балансировка для каждого набора данных. Были отброшены классы, содержащие малое количество записей. Для каждого из оставшихся классов было выбрано одинаковое количество записей.

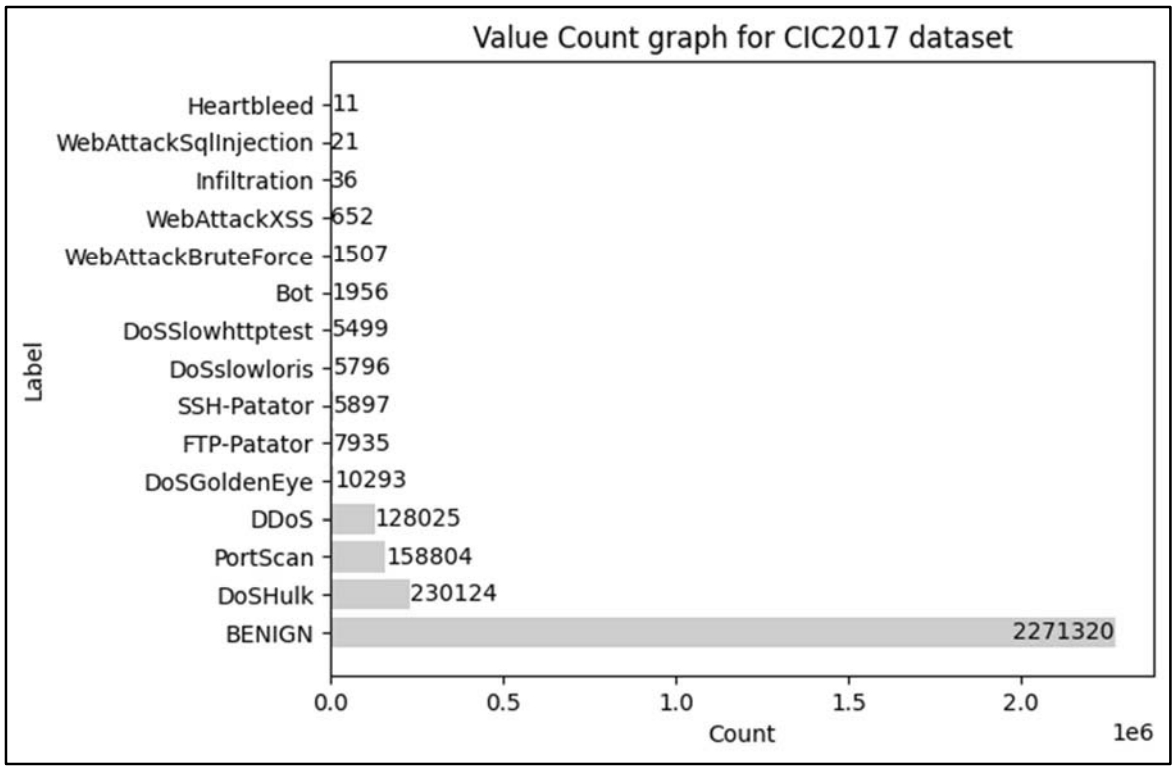

Рисунок 22 – Распределение записей в наборе данных CIC-IDS-2017

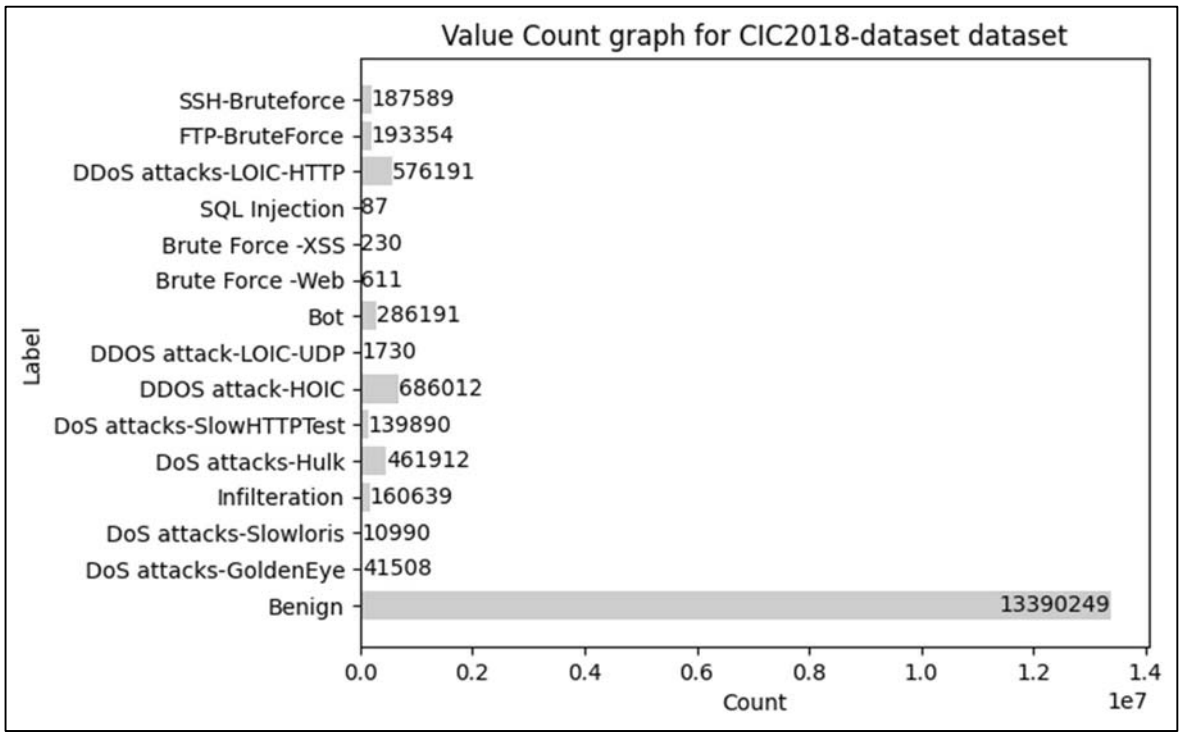

Рисунок 23 – Распределение данных в наборе данных CSE-CIC-IDS2018

В ходе балансировки набора данных CIC-IDS-2017 были выбраны 5 499 записей для классов, представленных ниже.

1. DoS Slowhttptest.
2. DoS slowloris.
3. SSH-Patator.
4. FTP-Patator.
5. DoS GoldenEye.
6. DDoS.
7. PortScan.
8. DoS Hulk.
9. BENIGN.

По результатам балансировки набора данных CSE-CIC-IDS2018 были выбраны 10 990 записей для классов, представленных ниже.

1. DDoS attacks-LOIC-HTTP.
2. DoS attacks-SlowHTTPTest.
3. DoS attacks-Hulk.

4. SSH-BruteForce.
5. FTP-BruteForce.
6. DoS attacks-Slowloris.
7. DoS attacks-GoldenEye.
8. Bot.
9. DDOS attack-HOIC.
10. Infilteration.
11. Benign.

Данные в наборе данных были обработаны с помощью метода StandardScaler [44], реализованного в библиотеке sklearn. Данный метод масштабирует данные так, чтобы среднее значение стало равно нулю, а стандартное отклонение стало равно 1.

Дальнейшая предобработка заключалась в отборе наиболее значимых признаков. Используемые наборы данных имеют общий набор признаков, отбор проводился с использованием набора данных CIC-IDS-2017. Предварительно, веса признаков были получены в ходе обучения модели `RandomForestClassifier`. Исходя из полученных результатов, был сделан вывод, что в наборе присутствуют признаки, минимально влияющие на результат классификации.

Для отбора признаков использовался метод Recursive Feature Elimination [45] из библиотеки sklearn. Данный метод принимает на вход модель-оценщик и количество признаков, которые требуется оставить в исходном наборе данных. В начале отбора модель-оценщик обучается на полном наборе признаков. На каждой следующей итерации определяются веса признаков, наименее важные признаки отбрасываются, а оценщик обучается на меньшем наборе признаков. Код отбора признаков представлен в листинге 1.

Листинг 1 – Функция отбора признаков

```
def select_features(train_x, train_y):
  model = RandomForestClassifier(25, random_state=RANDOM_STATE)
  rfe = RFE(estimator = model, n_features_to_select=38)
  rfe.fit(train_x, train_y)
  feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(),
train_x.columns)]
```

```
  selected_features = [v for i, v in feature_map if i==True]
  return  pd.DataFrame(rfe.support_,index=train_x.columns,columns=["Rank"]),
selected_features
```

Как видно из листинга, функция возвращает таблицу, содержащую информацию об использовании признаков и список отобранных признаков. Отобранные признаки представлены на рисунке 24.

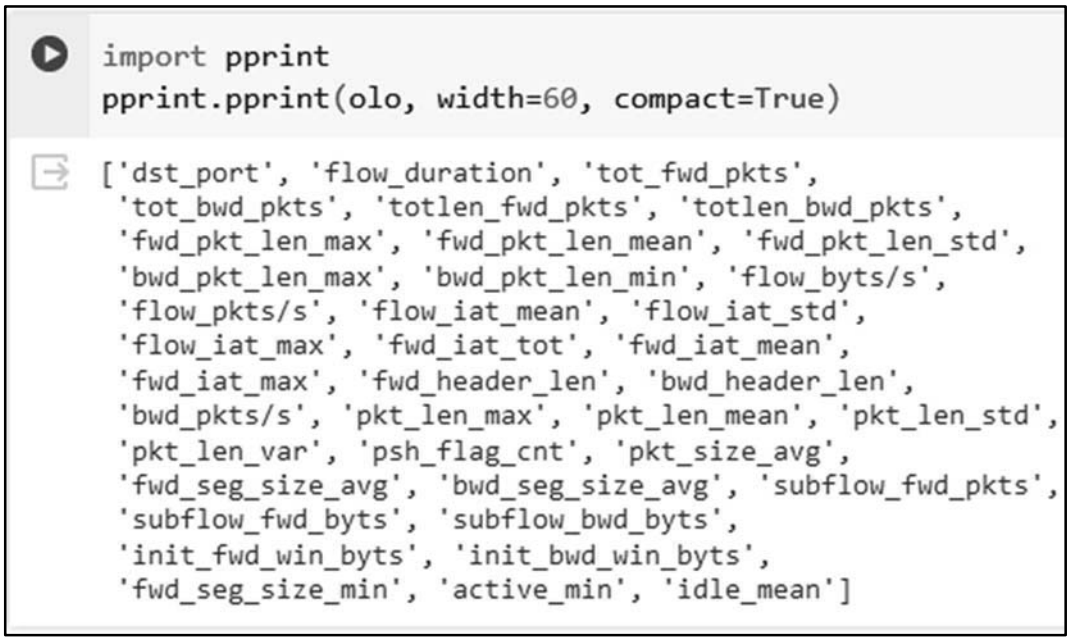

Рисунок 24 – Отобранные признаки

В ходе предварительной обработки исходные наборы данных были унифицированы, были отброшены малозначимые признаки, признаки с единственными значениями, а также была проведена балансировка наборов данных.

4.3. Реализация алгоритмов машинного обучения

Реализация Random Forest

Для реализации модели был взят класс `RandomForestClassifier` из библиотеки scikit-learn. Для подбора гиперпараметров было проведено A/B тестирование. В ходе тестирования сравнивалась точность модели в зависимости от количества участников ансамбля. Для остальных параметров, описанных в документации, были выбраны стандартные значения. Результаты

тестирования находятся в приложении А. Из результатов тестирования видно, что точность модели выходит на плато и перестает значительно расти при значении 40.

Реализация AdaBoost

Для реализации модели был использован класс `AdaBoostClassifier` из библиотеки scikit-learn. Подбор гиперпараметров осуществлялся в ходе A/B тестирования. В ходе тестирования оценивалась зависимость точности обученного ансамбля от количества участников ансамбля и глубины дерева выбора слабого ученика. Остальные параметры, описанные в документации, принимают стандартные значения. Результаты тестирования представлены в приложении А. Как видно из результатов тестирования, точность модели перестает значительно расти при максимальной глубине дерева равной 5 и количестве участников ансамбля равном 40.

Реализация SVM

Для реализации модели был использован класс `SVC` из библиотеки scikit-learn. Для того, чтобы модель могла классифицировать множество классов был указан дополнительный параметр `decision_function_shape`, который принимает значение «ovo». При указании данного параметра классификация происходит по методу one versus one. Это значит, что для решения мультиклассовой проблемы создается множество бинарных классификаторов. Итоговый результат в этом случае определяется голосованием всех классификаторов. Класс, который был выбран чаще всего, назначается итоговым классом.

Реализация CatBoost

Для реализации модели был использован класс `CatBoostClassifier` из библиотеки catboost. Для определения гиперпараметров модели было проведено A/B тестирование. В ходе A/B тестирования проверялась зависимость точности модели от количества слабых учеников. Результаты тестирования представлены в приложении А. По результатам проведенного те-

стирования был сделан вывод об оптимальном количестве участников ансамбля равном 25. При увеличении количества участников ансамбля не наблюдается значительного роста точности модели.

Реализация LSTM

Для реализации модели LSTM был написан класс `LSTMModel`, в котором определена архитектура нейронной сети, заданы обучающий и тестовый шаг. Нейронная сеть состоит из слоя LSTM, функции активации ReLU и полносвязного слоя. Код модели представлен в листинге 2.

Листинг 2 – Класс `LSTMModel`

```
class LSTMModel(pl.LightningModule):
  def __init__(self, input_size, hidden_size, num_layers, num_classes, lr):
  super().__init__()
  self.hidden_size = hidden_size
  self.num_layers = num_layers
  self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
  self.fc = nn.Linear(hidden_size, num_classes)
  self.relu = nn.ReLU()
  self.learning_rate = lr

  def forward(self, x):
   h0   =   Variable(torch.zeros(self.num_layers,   x.size(0),   self.hid-
den_size).to(self.device))
   c0   =   Variable(torch.zeros(self.num_layers,   x.size(0),   self.hid-
den_size).to(self.device))
   out, (hn, cn) = self.lstm(x, (h0, c0))
   hn = hn.view(-1, self.hidden_size)
   out = self.relu(hn)
   out = self.fc(out)
   return out

  def training_step(self, batch, batch_idx):
   x, y = batch
   y_hat = self(x)
   loss = F.cross_entropy(y_hat, y)
   self.log('train_loss', loss, on_step=False, on_epoch=True, prog_bar=True)
   return loss

def validation_step(self, batch, batch_idx):
   x, y = batch
   y_hat = self(x)
   loss = F.cross_entropy(y_hat, y)
   self.log('val_loss', loss, on_step=False, on_epoch=True, prog_bar=True)
   return loss

  def test_step(self, batch, batch_idx):
   x, y = batch
   y_hat = self(x)
   loss = F.cross_entropy(y_hat, y)
   self.log('test_loss', loss)
   return loss

  def configure_optimizers(self):
   return torch.optim.Adam(self.parameters(), lr=self.learning_rate)
```

Для работы с данными был реализован класс `CICIDSDataset`, который хранит полученные данные в формате DataFrame и содержит переопределенные методы `__len__` и `__get_item__` для получения размера набора данных и получения предварительно обработанной записи по индексу. Код класса представлен в листинге 3.

Листинг 3 – Класс `CICIDSDataset`

```
class CICIDSDataset(Dataset):
  def __init__(self, X: pd.DataFrame, y: pd.DataFrame):
   self.X = X
   self.y = y

  def __len__(self):
   return len(self.X)

  def __getitem__(self, idx):
   return torch.tensor(self.X.iloc[idx], dtype=torch.float32).unsqueeze(0),
torch.tensor(self.y[idx], dtype=torch.int64)
```

Для обучения модели использовался объект класса `Trainer` из библиотеки pytorch-lightning. Данный класс управляет обучением, валидацией и тестированием моделей. Код обучения модели представлен в листинге 4.

Листинг 4 – Обучение LSTM модели

```
input_size = 38
hidden_size = 128
num_layers = 1
num_classes = len(l_encoder.classes_)
model = LSTMModel(input_size, hidden_size, num_layers, num_classes, 1e-4)
train_dataset = CICIDSDataset(train_x, train_y)
train_dataloader = DataLoader(train_dataset, batch_size=32, shuffle=True)
trainer = pl.Trainer(max_epochs=50, callbacks=[pl.callbacks.EarlyStop-
ping(monitor="train_loss", mode="min")])
trainer.fit(model, train_dataloader)
```

4.4. Оценка качества обученных моделей

Для обучения и тестирования исходный набор данных был разделен на обучающую и тестовую часть. Обучающая часть составляет 80% от исходного набора данных, тестовая часть составляет 20%.

По результатам обучения можно вычислить метрики качества. Для оценки качества обученных моделей использовались следующие метрики: Accuracy, Precision, Recall, F1-score.

Accuracy – метрика, отражающая отношение правильно предсказанных записей к общему количеству предсказанных записей. Precision – метрика, вычисляемая как отношение истинных положительных результатов к общему числу предсказанных положительных результатов. Recall – соотношение истинных позитивно предсказанных записей в классе к общему количеству правильно предсказанных записей. F1-score – гармоническая мера Precision и Recall, высокие значения этой метрики говорят о высоких значениях Precision и Recall.

По результатам вычисления метрик оценки качества был построен отчет об обучении. Отчет представлен на рисунках 25–26.

	Model	Accuracy	Precision	Recall	F1 score
0	Random Forest	0.997980	0.997983	0.997980	0.997980
1	AdaBoost	0.995858	0.995911	0.995858	0.995857
2	SVM	0.961511	0.963758	0.961511	0.961587
3	CatBoost	0.997879	0.997881	0.997879	0.997878
4	LSTM	0.986766	0.987180	0.986766	0.986769

Рисунок 25 – Результаты обучения моделей с помощью набора данных CIC-IDS-2017

	Model	Accuracy	Precision	Recall	F1 score
0	Random Forest	0.894822	0.915810	0.894822	0.889139
1	AdaBoost	0.891306	0.894610	0.891306	0.889864
2	SVM	0.881256	0.910099	0.881256	0.872426
3	CatBoost	0.909463	0.915243	0.909463	0.907609
4	LSTM	0.893209	0.899780	0.893209	0.890937

Рисунок 26 – Результаты обучения моделей с помощью набора данных CSE-CIC-IDS2018

Исходя из рисунков выше был сделан вывод, что все обученные модели достигли высокой точности на тестовой выборке. Наилучшего результата по метрике F1 достигла модель Random Forest, обученная на наборе данных CIC-IDS-2017.

4.5. Реализация анализатора пакетов

Анализатор пакетов был реализован с использованием библиотеки Scapy. Для получения необработанных пакетов с пользовательского интерфейса был использован класс `AsyncSniffer`. Данный класс позволяет получать интернет-пакеты асинхронно, а также управлять процессом получения пакетов.

Получение информации из пакета реализовано с помощью класса `PacketInfo`. Класс `PacketInfo` описывает структуру хранения параметров пакетов и содержит методы установки и получения различных параметров пакета. Параметры устанавливаются путем преобразования информации с уровней TCP/IP архитектуры.

Для получения и обработки интернет-пакетов был реализован класс Worker. Данный класс управляет процессом получения пакетов, содержит методы по обработке входящих и исходящих интернет-пакетов, а также метод классификации интернет-пакетов.

Для обработки пакетов, полученных с помощью `AsyncSniffer`, был реализован метод `newPacket`. В этом методе описан процесс получения информации из необработанного пакета и реализована логика управления потоками. Исходный код метода `newPacket` представлен в Приложении Б. В методе осуществляется проверка принадлежности пакета существующим потокам. Если пакет не принадлежит какому-либо потоку, создается новый поток на основе полученного пакета. В программе определено время бездействия потока. При превышении времени бездействия вызывается метод классификации, текущий поток уничтожается и создается новый поток на основе последнего полученного потока. Если последний полученный пакет содержит флаг окончания потока вызывается метод классификации и поток уничтожается.

При остановке работы `AsyncSniffer` для каждого оставшегося потока вызывается метод классификации.

Классификация потока реализована в методе `classify`. В методе `classify` происходит масштабирование полученных данных, классификация данных и вывод информации в консоль и пользовательский интерфейс. Код метода представлен в листинге 5.

Листинг 5 – Классификация пакетов

```
def classify(self, features):
  # preprocess
  try:
   f = features
   lv = 0
   features = self.normalization.transform([f])
   result = self.model.predict(features)

   feature_string = [str(i) for i in f]
   classification = [str(result[0])]
   result_string = f"[{datetime.now().strftime(self.format)}] - {classifica-
tion[0]}"
   if result in ['BENIGN', 'Benign']:
    lv = 1

   self.signals.prints.emit([result_string, lv])
  except Exception as e:
   print(result)
   exctype, value = sys.exc_info()[:2]
   self.signals.error.emit((exctype, value, traceback.format_exc()))
  return feature_string + classification
```

Для вычисления статистических характеристик и описания сетевых потоков был реализованы классы `Flow` и `FlowFeature`. В классе `Flow` реализованы методы `new` и `terminated`. Метод `new` описывает процесс обновления характеристик потока при получении нового пакета. Метод `terminated` описывает процесс получения финальных характеристик перед уничтожением потока. Исходный код методов `new` и `terminated` находится в Приложении Б. Характеристики сетевого потока хранятся в классе `FlowFeature`. Класс `FlowFeature` содержит методы для установки и получения характеристик потока.

4.6. Реализация пользовательского интерфейса

Основываясь на диаграмме вариантов использования, был реализован пользовательский интерфейс приложения. Пользовательский интерфейс ре-

ализован с использованием библиотеки PyQT6 [47]. Библиотека PyQT6 является оберткой для мультиплатформенного фреймворка Qt, предназначенного для создания графических интерфейсов.

Главное окно приложения реализовано с помощью класса `MainWindow`, наследуемого от класса `QMainWindow`. Создание новых элементов внутри окна приложения реализовано с помощью добавления виджетов. С помощью соответствующих виджетов были добавлены следующие элементы: кнопки, чекбоксы, выпадающие списки.

Взаимодействие с элементами интерфейса в библиотеке обрабатывается с помощью сигналов и слотов. Сигналы – это данные, которые отправляет виджет при взаимодействии. Слот – это функция, которая обрабатывает полученный сигнал. Пример обработки сигнала от кнопки представлен в листинге 6.

Листинг 6 – Обработка взаимодействия с чекбоксом

```
self.workMode = QCheckBox('Use offline analysis')
self.workMode.setCheckState(Qt.CheckState.Unchecked)
self.workMode.stateChanged.connect(self.activateCheckBox)

def activateCheckBox(self, s):
  if s == 0:
  self.workModeLayout.setCurrentIndex(0)
  self.selectedWorkMode = "Live"
  if s == 2:
  self.workModeLayout.setCurrentIndex(1)
  self.selectedWorkMode = "Offline"
```

Интерфейс пользователя состоит из следующих элементов: кнопка «Run Detection»; выпадающий список «model»; кнопка «Stop»; выпадающий список «interface»; чекбокс «Use offline analysis»; кнопка «Select File»; текстовое поле; кнопка «Save Logs».

Кнопка «Run detection» отвечает за запуск процесса анализа трафика с выбранными параметрами. При нажатии на кнопку отправляется сигнал и вызывается функция `runButton_was_clicked`. В этой функции фиксируется выбранная модель, выбранный интерфейс или выбранный файл для анализа, зафиксированные параметры передаются экземпляру класса `Worker`. После чего кнопки становятся неактивными, и с помощью класса

`QThreadPool` запускается анализ трафика. Исходный код функции представлен в приложении В.

Кнопка «Stop» отвечает за остановку процесса анализа трафика. При нажатии на кнопку в экземпляре класса `Worker` вызывается метод `stop`. При вызове метода `stop` завершается работа `AsyncSniffer` и отправляется сигнал о завершении работы в `MainWindow`. Обработка сигнала происходит в функции `thread_complete`. Данная функция обнуляет данные экземпляра класса Worker и разблокирует ранее заблокированные кнопки.

В выпадающем списке «model» пользователь может выбрать предварительно обученную модель машинного обучения, которая затем будет использована в процессе анализа трафика.

В выпадающем списке «interface» пользователь может выбрать сетевой интерфейс, который будет прослушиваться в ходе анализа трафика.

Чекбокс «Use offline analysis» отвечает за то, в каком режиме будет работать приложение. При активации чекбокса интерфейс изменяется и пользователю предлагается выбрать локальный файл в формате pcap.

Кнопка «Select File» вызывает диалоговое окно выбора файла. Путь до выбранного файла сохраняется в локальную переменную, которая передается экземпляру класса `Worker`.

В текстовом поле печатаются результаты работы анализатора трафика, а также системные сообщения о работе программы.

Кнопка «Save Logs» вызывает функцию `saveLogsButton_was_clicked`. Данная функция получает информацию из текстового поля и сохраненяет е в файл с датой последнего запуска. При нажатии на кнопку также очищается текстовое поле.

На рисунке 27 представлен итоговый вид графического интерфейса пользователя.

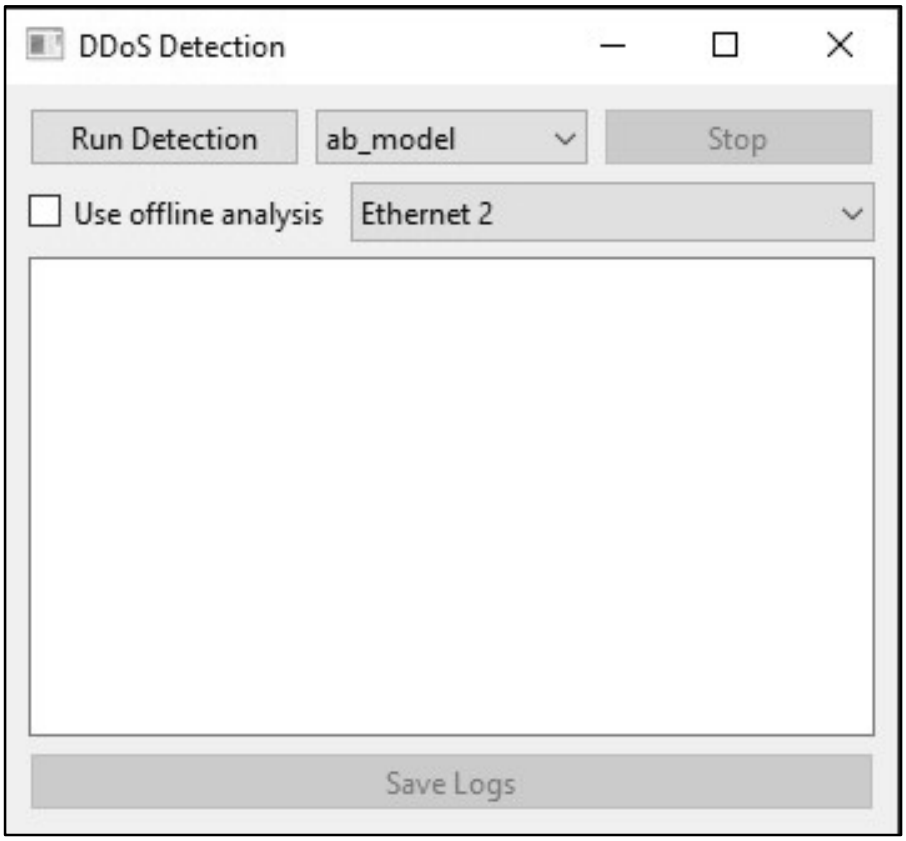

Рисунок 27 – Графический интерфейс пользователя

Выводы по четвертой главе

В данной главе описаны средства разработки, используемые при создании приложения. Описан процесс предварительной обработки исходных наборов данных. Был описан процесс и результаты обучения моделей машинного обучения. В соответствии с поставленными задачами были реализованы требуемые компоненты системы, а также реализован графический интерфейс пользователя. Был реализован анализатор пакетов, работающий в оффлайн и онлайн режиме.

5. ТЕСТИРОВАНИЕ

5.1. Функциональное тестирование

В ходе данного тестирования было проверено соответствие системы заявленным функциональным требованиям. Для проведения тестирования были созданы тестовые случаи. Результаты тестирования представлены в таблице 1.

Таблица 1 – Функциональное тестирование

№	Название теста	Действия	Результат	Тест пройден?
1	Выбор модели	1. Нажать на выпадающее меню с моделями. 2. Выбрать модель.	В текстовом поле выводится запись, что модель была выбрана.	Да
2	Результат классификации	Нажать на кнопку «Run Detection».	Приложение начинает процесс анализа трафика, в текстовом поле выводится информация о ходе работы.	Да
3	Ведение журнала	1. Запустить приложение. 2. Нажать на кнопку «Run Detection».	Приложение начинает писать информацию о ходе работы.	Да
4	Смена сетевого интерфейса	1. Нажать на выпадающее меню с сетевым интерфейсом. 2. Выбрать сетевой интерфейс.	В графическом интерфейсе сменится интерфейс. В текстовом поле появится запись об успешной смене.	Да
5	Сохранение записей	1. Нажать на кнопку «Run Detection». 2. Нажать на кнопку «Stop». 3. Нажать на кнопку «Save Logs»	При нажатии на кнопку «Save Logs» текстовое поле очищается, а в папке output появляется текстовый файл с результатами работы.	Да
6	Повторный запуск	1. Нажать на кнопку «Run Detection». 2. Нажать на кнопку «Stop». 3. Нажать на кнопку «Run Detection»	После повторного нажатия на кнопку «Run Detection» процесс анализа трафика запускается снова.	Да

5.2. Unit тестирование

Разработка приложения велась по методологии TDD. Методология TDD подразумевает разработку через написание тестов. В связи с используемой методологией, по ходу разработки был создан перечень unit тестов,

определяющих поведение системы. В ходе разработки были написаны тесты, проверяющие установку параметров работы приложения, реакцию графического интерфейса на действия пользователя, работу анализатора пакетов. В соответствии с написанными тестами была реализована функциональность системы.

A/B тестирование

В ходе A/B тестирования сравниваются два объекта «как было», «как стало» с целью определить, как повлияют вносимые изменения на результат. В рамках машинного обучения такое тестирование можно провести, варьируя количество тестовых данных или изменяя гиперпараметры.

Для выбранных моделей было проведено A/B тестирование, в ходе которого варьировалось количество записей при балансировке, а также исходные данные без балансировки. Результаты представлены в таблице 2.

Таблица 2 – A/B тестирование

Кол-во записей на класс	Random Forest	AdaBoost	SVM	CatBoost	LSTM
1 000	99,3%	98,7%	92,3%	99,5%	94,1%
2 000	99,6%	94,0%	94,5%	99,8%	97,3%
3 000	99,5%	99,5%	94,7%	99,7%	97,4%
5 499	99,8%	99,3%	95,4%	99,8%	98,5%
Без балансировки	99,8%	98,9%	85,3%	99,8%	98,1%

Из таблицы видно, что проведенная балансировка улучшила качество обученных моделей.

Выводы по пятой главе

В данной главе был описан процесс тестирования разработанной системы. Было проведено функциональное тестирование, unit тестирование и A/B тестирование. Все тесты были успешно пройдены, что обеспечивает высокую надежность системы.

ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано приложение для анализа сетевого трафика в режиме реального времени на основе методов машинного обучения. При этом были решены следующие задачи.

1. Проведен анализ предметной области.
2. Подготовлен набор данных.
3. Выбраны модели машинного обучения.
4. Реализованы выбранные модели машинного обучения.
5. Выполнено проектирование системы анализа трафика в реальном времени.
6. Реализована спроектированная система.
7. Осуществлена оценка качества разработанного приложения.

В будущем планируется совершенствование разработанного приложения, расширение списка используемых методов машинного обучения, расширение функциональности, поддержка консольного режима работы приложения, поддержка общего интерфейса получения характеристик пакетов для использования большего количества моделей. Также планируется разработка и внедрение механизмов предотвращения вторжений. Для улучшения качества классификации можно использовать следующие методы: разработка более глубоких архитектур нейронных сетей, увеличение набора данных, более глубокий анализ используемых признаков.

ЛИТЕРАТУРА

1. DDoS attacks in Q1 2021. [Электронный ресурс] URL: https://securelist.com/ddos-attacks-in-q1-2021/102166/ (дата обращения: 24.02.2024 г.).

2. DDoS attacks in Q1 2022. [Электронный ресурс] URL: https://securelist.com/ddos-attacks-in-q1-2022/106358/ (дата обращения: 24.02.2024 г.).

3. DDoS attacks in Q3 2022. [Электронный ресурс] URL: https://securelist.com/ddos-report-q3-2022/107860/ (дата обращения 24.02.2024 г.).

4. DDoS Attack Trends and Insights in 2023. [Электронный ресурс] URL: https://ddos-guard.net/en/blog/ddos-attack-trends-2023 (дата обращения: 24.02.2024 г.).

5. DDoS threat report for 2023 Q4. [Электронный ресурс] URL: https://blog.cloudflare.com/ddos-threat-report-2023-q4 (дата обращения: 24.02.2024 г.).

6. Meng X., Lin C., Wang Y., Zhang Y. Generative Pretrained Transformer for Network Traffic [Электронный ресурс] // arXiv.org. 2023. Дата обновления: 17.05.2023. URL: https://arxiv.org/abs/2304.09513 (дата обращения: 13.05.2024 г.).

7. Abreu D., Abelem A. OMINACS: Online ML-based IOT network attack detection and classification system. // 2022 IEEE Latin-American Conference on Communications (LATINCOM), 2022. – С. 1–6.

8. Jonathan O., Misra S., Osamor V. Comparative analysis of machine learning techniques for network traffic classification. // IOP Conference Series: Earth and Environmental Science, 2021. – Т. 655. – № 1. – С. 12–25.

9. Kumari K., Mrunalini M. Detecting denial of service attacks using machine learning algorithms. // Journal of Big Data, 2022. – Т. 9. – № 1. – С. 56–73.

10. Классификация. [Электронный ресурс] URL: http://www.machinelearning.ru/wiki/index.php?title=Классификация (дата обращения: 07.04.2024 г.).

11. Гетьман А.И., Маркин Ю.В., Евстропов Е.Ф., Обыденков Д.О. Обзор задач и методов их решения в области классификации сетевого трафика. // Труды ИСП РАН, 2017. №3 – С. 117–150.

12. What is Transmission Control Protocol TCP/IP? [Электронный ресурс] URL: https://www.fortinet.com/resources/cyberglossary/tcp-ip (дата обращения: 07.04.2024 г.).

13. Руководство по стеку протоколов TCP/IP для начинающих. [Электронный ресурс] URL: https://selectel.ru/blog/tcp-ip-for-beginners/ (дата обращения: 07.04.2024 г.).

14. Cisco WAN and Application Optimization Solution Guide – Traffic Classification [Support] | Cisco. [Электронный ресурс] URL: https://www.cisco.com/c/en/us/td/docs/nsite/enterprise/wan/wan_optimization/wan_opt_sg/chap05.html (дата обращения: 07.04.2024 г.).

15. Qadeer M.A., Siddiqui M.R., Zahid M., Iqbal A. Network traffic analysis and Intrusion Detection Using Packet Sniffer. // 2010 Second International Conference on Communication Software and Networks, 2010. – Т. 1. – № 1. – С. 313–317.

16. An Introduction to Recurrent Neural Networks and the Math That Powers Them. [Электронный ресурс] URL: https://machinelearningmastery.com/an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/ (дата обращения: 07.04.2024 г.).

17. Understanding LSTM Networks. [Электронный ресурс] URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (дата обращения: 07.04.2024 г.).

18. A Comprehensive Guide to Ensemble Learning (with Python codes). [Электронный ресурс] URL: https://www.analyt-icsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/ (дата обращения: 07.04.2024 г.).

19. Ensemble methods: bagging, boosting and stacking. [Электронный ресурс] URL: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205 (дата обращения: 07.04.2024 г.).

20. Метод опорных векторов (SVM). [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Метод_опорных_векторов_(SVM) (дата обращения: 07.04.2024 г.).

21. Support Vector Machines (SVM): An Intuitive Explanation. [Электронный ресурс] URL: https://medium.com/low-code-for-advanced-data-science/support-vector-machines-svm-an-intuitive-explanation-b084d6238106 (дата обращения: 08.04.2024 г.).

22. CatBoost – open-source gradient boosting library. [Электронный ресурс] URL: https://catboost.ai/ (дата обращения: 07.04.2024 г.).

23. Dorogush A., Ershov V., Gulin A. CatBoost: gradient boosting with categorical features support [Электронный ресурс] // arXiv.org. 2018. Дата обновления: 24.10.2018. URL: https://arxiv.org/abs/1810.11363 (дата обращения: 13.05.2024 г.).

24. Intrusion detection evaluation dataset (CIC-IDS2017). [Электронный ресурс] URL: https://www.unb.ca/cic/datasets/ids-2017.html (дата обращения: 07.04.2024 г.).

25. CSE-CIC-IDS2018 on AWS. [Электронный ресурс] URL: https://www.unb.ca/cic/datasets/ids-2018.html (дата обращения: 07.04.2024 г.).

26. Applications. [Электронный ресурс] URL: https://www.unb.ca/cic/research/applications.html#CICFlowMeter (дата обращения: 07.04.2024 г.).

27. Abdulhammed R., Musafer H., Alessa A., Faezipour M., Abuzneid A., Features Dimensionality Reduction Approaches for Machine Learning Based Network Intrusion Detection. // Electronics, 2017 – Т.8. – № 3. – С. 322–349.

28. Slowloris DDoS Attack. [Электронный ресурс] URL: https://ddos-guard.net/en/terms/ddos-attack-types/slowloris (дата обращения: 07.04.2024 г.).

29. Hulk DDoS Tool: Complete Installation & Usage with Examples. [Электронный ресурс] URL: https://allabouttesting.org/hulk-ddos-tool-complete-installation-usage-with-examples/ (дата обращения: 07.04.2024 г.).

30. Golden Eye DDoS Tool: Installation and Tool usage with examples. [Электронный ресурс] URL: https://allabouttesting.org/golden-eye-ddos-tool-installation-and-tool-usage-with-examples/ (дата обращения: 07.04.2024 г.).

31. The Heartbleed Bug. [Электронный ресурс] URL: https://heartbleed.com/ (дата обращения: 07.04.2024 г.).

32. What Is a Port Scan? How to Prevent Port Scan Attacks? [Электронный ресурс] URL: https://www.fortinet.com/resources/cyberglossary/what-is-port-scan (дата обращения: 07.04.2024 г.).

33. Ares. [Электронный ресурс] URL: https://github.com/sweetsoftware/Ares (дата обращения: 07.04.2024 г.).

34. Patator. [Электронный ресурс] URL: https://github.com/lanjelot/patator (дата обращения: 07.04.2024 г.).

35. High Orbit Ion Cannon. [Электронный ресурс] URL: https://www.imperva.com/learn/ddos/high-orbit-ion-cannon/ (дата обращения: 07.04.2024 г.).

36. Low Orbit Ion Cannon. [Электронный ресурс] URL: https://www.imperva.com/learn/ddos/low-orbit-ion-cannon/ (дата обращения: 07.04.2024 г.).

37. Ferrag A. M., Hamouda D., Shu L. Deep Learning-Based Intrusion Detection for Distributed Denial of Service Attack in Agriculture 4.0. // Electronics, 2021 – Т.10. – № 11. – С. 1257–1283.

38. Лекция 4: Методы определения требований в программной инженерии. [Электронный ресурс] URL: https://intuit.ru/studies/courses/945/237/lecture/6122 (дата обращения: 07.04.2024 г.).

39. Лекция 3: Элементы графической нотации диаграммы вариантов использования. [Электронный ресурс] URL: https://intuit.ru/studies/courses/32/32/lecture/1004 (дата обращения: 09.04.2024 г.).

40. Google Colaboratory. [Электронный ресурс] URL: https://colab.google/ (дата обращения 01.05.2024 г.).

41. Официальная документация Pytorch Lightning. [Электронный ресурс] URL: https://lightning.ai/docs/pytorch/stable//index.html (дата обращения 01.05.2024 г.).

42. Официальная документация Scapy. [Электронный ресурс] URL: https://scapy.readthedocs.io/en/latest/ (дата обращения 01.05.2024 г.).

43. Visual Studio Code – Code Editing. Refined. [Электронный ресурс] URL: https://code.visualstudio.com/ (дата обращения 01.05.2024 г.).

44. StandardScaler. [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (дата обращения 01.05.2024 г.).

45. RFE. [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html (дата обращения 01.05.2024 г.).

46. Performance Metrics in Machine Learning [Complete Guide]. [Электронный ресурс] URL: https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide (дата обращения 05.05.2024 г.).

47. Официальная документация PyQT6. [Электронный ресурс] URL: https://www.riverbankcomputing.com/static/Docs/PyQt6/ (дата обращения 05.05.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Подбор гиперпараметров моделей

На рисунках 1–4 представлены графики зависимости точности моделей от выбранных гиперпараметров из главы «Реализация».

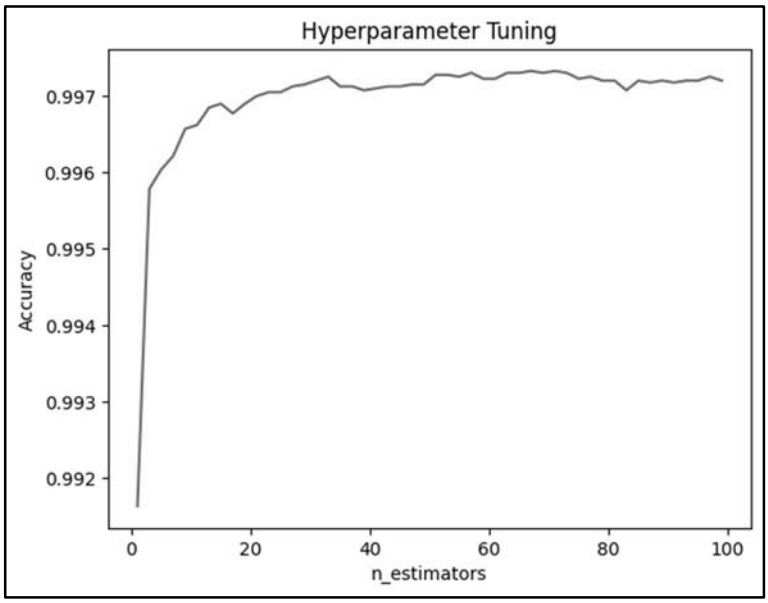

Рисунок 1 – Зависимость точности Random Forest Classifier от параметра n_estimators

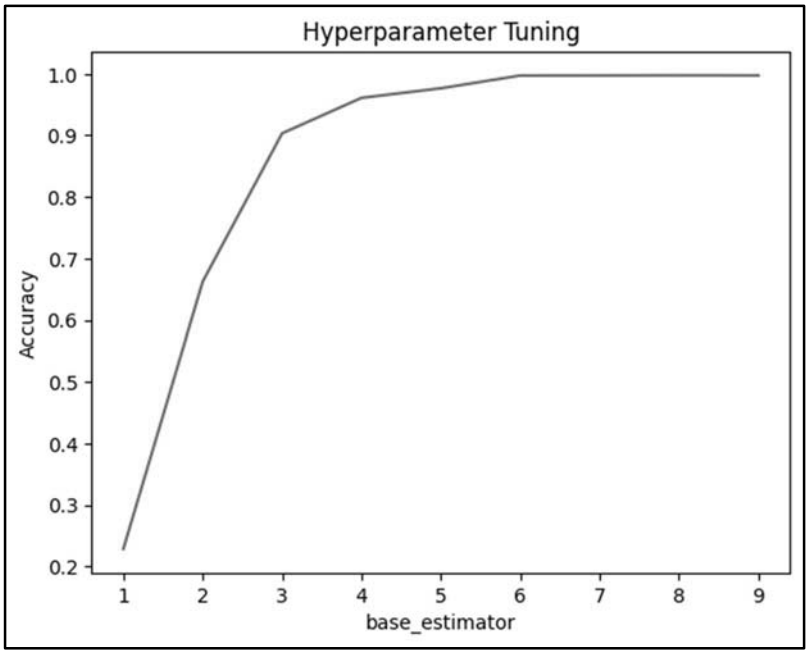

Рисунок 2 – Зависимость точности AdaBoostClassifier от параметра max_depth

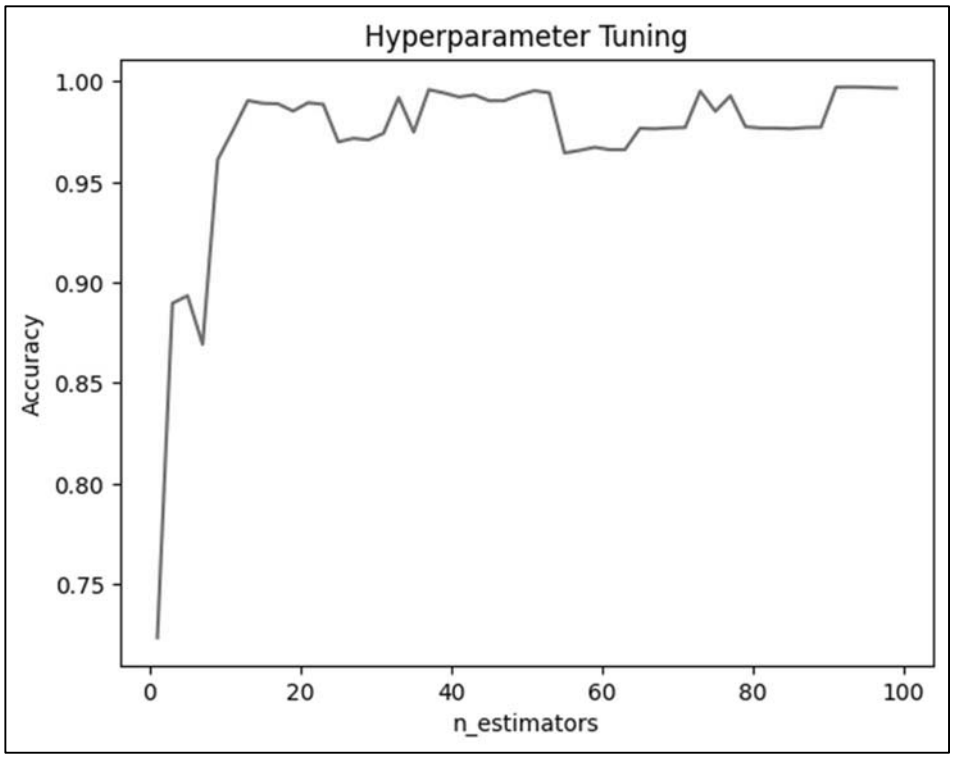

Рисунок 3 – Зависимость точности AdaBoostClassifier
от параметра n_estimators

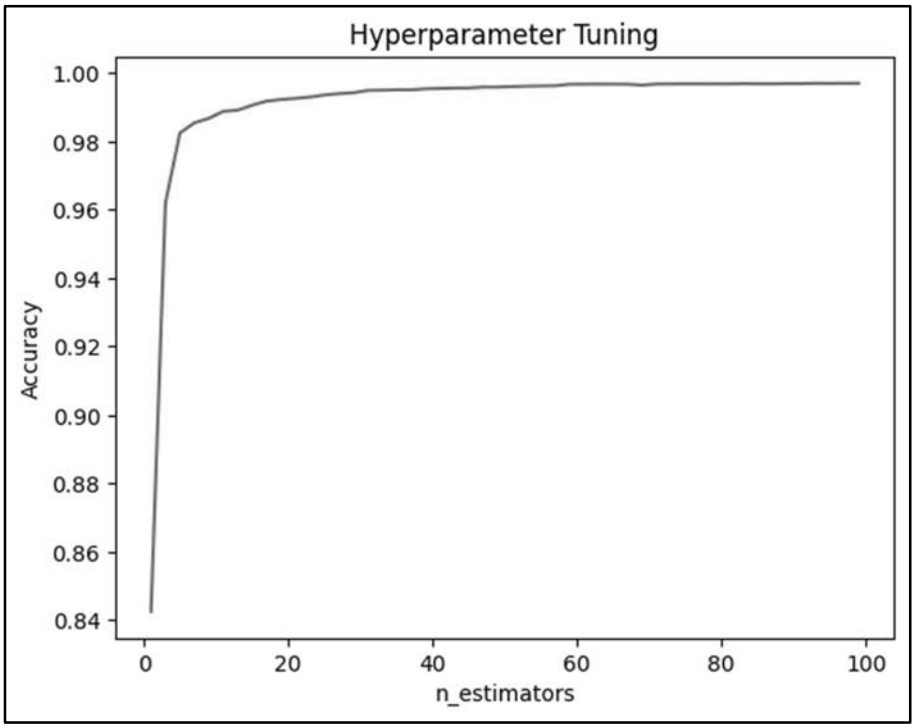

Рисунок 4 – Зависимость точности CatBoostClassifier
от параметра n_estimators

Приложение Б. Исходный код анализатора пакетов

Код функции обработчика пакетов представлен в листинге 1.

Листинг 1 – Код функции обработчика пакетов

```
def newPacket(self, p):
  try:
  packet = PacketInfo()
  packet.setDest(p)
  packet.setSrc(p)
  packet.setSrcPort(p)
  packet.setDestPort(p)
  packet.setProtocol(p)
  packet.setTimestamp(p)
  packet.setPSHFlag(p)
  packet.setFINFlag(p)
  packet.setSYNFlag(p)
  packet.setACKFlag(p)
  packet.setURGFlag(p)
  packet.setRSTFlag(p)
  packet.setPayloadBytes(p)
  packet.setHeaderBytes(p)
  packet.setPacketSize(p)
  packet.setWinBytes(p)
  packet.setFwdID()
  packet.setBwdID()

  if packet.getFwdID() in self.self.current_flows.keys():
  flow = self.current_flows[packet.getFwdID()]

  # check for timeout
  if (packet.getTimestamp() - flow.getFlowStartTime()) > FlowTimeout:
    self.classify(flow.terminated())
    del self.current_flows[packet.getFwdID()]
    flow = Flow(packet)
    self.current_flows[packet.getFwdID()] = flow

  # check for fin flag
  elif packet.getFINFlag() or packet.getRSTFlag():
    flow.new(packet, 'fwd')
    self.classify(flow.terminated())
    del self.current_flows[packet.getFwdID()]
    del flow

  else:
    flow.new(packet, 'fwd')
    self.current_flows[packet.getFwdID()] = flow

  elif packet.getBwdID() in self.current_flows.keys():
  flow = self.current_flows[packet.getBwdID()]

  # check for timeout
  if (packet.getTimestamp() - flow.getFlowStartTime()) > FlowTimeout:
    self.classify(flow.terminated())
    del self.current_flows[packet.getBwdID()]
    del flow
    flow = Flow(packet)
    self.current_flows[packet.getFwdID()] = flow

  elif packet.getFINFlag() or packet.getRSTFlag():
    flow.new(packet, 'bwd')
```

```
    self.classify(flow.terminated())
    del self.current_flows[packet.getBwdID()]
    del flow
  else:
    flow.new(packet, 'bwd')
    self.current_flows[packet.getBwdID()] = flow
  else:

  flow = Flow(packet)
  self.current_flows[packet.getFwdID()] = flow
  # current flows put id, (new) flow

  except AttributeError:
  # not IP or TCP
  return

  except:
  traceback.print_exc()
```

В листинге 2 представлен исходный код метода `new` класса `Flow`.

Листинг 2 – Исходный код метода `new`

```
def new(self, packet: PacketInfo, direction: str):
  if direction == 'bwd':
  self.bwdPacketInfos.append(packet)

  if self.bwd_packet_count == 0:
  self.flowFeatures.p_stats.bwdStats.setBwdPacketLenMax(packet.getPay-
loadBytes())
   else:
  self.flowFeatures.p_stats.bwdStats.setBwdPacketLenMax(
  max(self.flowFeatures.p_stats.bwdStats.getBwdPacketLenMax,
     packet.getPayloadBytes()))
  self.bwdIAT.append((packet.getTimestamp() - self.bwdLastSeen) * 1000 *
1000)

  self.bwd_packet_count = self.bwd_packet_count + 1
  self.bwdLastSeen = packet.getTimestamp()

  else:
  self.fwdPacketInfos.append(packet)
  self.fwdIAT.append((packet.getTimestamp() - self.fwdLastSeen) * 1000 *
1000)
  self.flowFeatures.flags.setFwdPSHFlags(max(1 if packet.getURGFlag() else 0,
       self.flowFeatures.getFwdPSHFlags()))
  self.fwd_packet_count = self.fwd_packet_count + 1
  self.fwdLastSeen = packet.getTimestamp()

  self.flowFeatures.p_stats.flStats.setMaxPacketLen(max(self.flowFea-
tures.p_stats.flStats.getMaxPacketLen(),packet.getPayloadBytes()))

  if packet.getFINFlag():
  self.flowFeatures.flags.setFINFlag(1)
  if packet.getSYNFlag():
  self.flowFeatures.flags.setSYNFlag(1)
  if packet.getPSHFlag():
  self.flowFeatures.flags.setPSHFlag(1)
  if packet.getACKFlag():
```

```
  self.flowFeatures.flags.setACKFlag(1)
  if packet.getURGFlag():
  self.flowFeatures.flags.setURGFlag(1)

  time = packet.getTimestamp()
  if time - self.endActiveTime > threshold:
  if self.endActiveTime - self.startActiveTime > 0:
  self.flowActive.append(float(self.endActiveTime - self.startActiveTime))
  self.flowIdle.append(time - self.endActiveTime)
  self.startActiveTime = time
  self.endActiveTime = time
  else:
  self.endActiveTime = time

  self.packet_count = self.packet_count + 1
  self.packetInfos.append(packet)
  self.flowIAT.append((packet.getTimestamp() - self.flowLastSeen) * 1000 *
1000)
  self.flowLastSeen = packet.getTimestamp()
```

В листинге 3 представлен исходный код метода terminated класса `Flow`.

Листинг 3 – Исходный код метода `terminated`

```
def terminated(self):
  duration = (self.flowLastSeen - self.flowStartTime) * 1000 * 1000
  self.flowFeatures.setFlowDuration(duration)
  fwd_packet_lens = [x.getPayloadBytes() for x in self.fwdPacketInfos]
  self.flowFeatures.p_len.setTotalFwdPacketsNum(len(fwd_packet_lens))
  self.flowFeatures.flow_bytes.setTotalLenFwdPackets(sum(fwd_packet_lens))
  if len(fwd_packet_lens) > 0:
  self.flowFeatures.p_stats.fwdStats.setFwdPack-
etLenMax(max(fwd_packet_lens))
  self.flowFeatures.p_stats.fwdStats.setFwdPacketLenMean(statis-
tics.mean(fwd_packet_lens))
  if len(fwd_packet_lens) > 1:
  self.flowFeatures.p_stats.fwdStats.setFwdPacketLenStd(statis-
tics.stdev(fwd_packet_lens))
  bwd_packet_lens = [x.getPayloadBytes() for x in self.bwdPacketInfos]
  self.flowFeatures.flow_bytes.setTotalLenBwdPackets(sum(bwd_packet_lens))
  if len(bwd_packet_lens) > 0:
  self.flowFeatures.p_stats.bwdStats.setBwdPack-
etLenMax(max(bwd_packet_lens))
  self.flowFeatures.p_stats.bwdStats.setBwdPack-
etLenMin(min(bwd_packet_lens))
  self.flowFeatures.p_stats.bwdStats.setBwdPacketLenMean(statis-
tics.mean(bwd_packet_lens))
  if len(bwd_packet_lens) > 1:
  self.flowFeatures.p_stats.bwdStats.setBwdPacketLenStd(statis-
tics.stdev(bwd_packet_lens))
  total_packets_len = sum([x.getPayloadBytes() for x in self.packetInfos])
  self.flowFeatures.p_time.setFlowBytesRate(total_packets_len / duration)
  self.flowFeatures.p_time.setFlowPacketsRate(self.packet_count / duration)
  self.flowFeatures.p_time.setBwdPacketsRate(sum(bwd_packet_lens) / dura-
tion)
  if len(self.flowIAT) > 0:
  self.flowFeatures.iat.flowIAT.setFlowIATMean(statis-
tics.mean(self.flowIAT))
  self.flowFeatures.iat.flowIAT.setFlowIATMax(max(self.flowIAT))
```

```
  self.flowFeatures.iat.flowIAT.setFlowIATMin(min(self.flowIAT))
  if len(self.flowIAT) > 1:
  self.flowFeatures.iat.flowIAT.setFlowIATStd(statis-
tics.stdev(self.flowIAT))
  if len(self.fwdIAT) > 0:
  self.flowFeatures.iat.fwdIAT.setFwdIATTotal(sum(self.fwdIAT))
  self.flowFeatures.iat.fwdIAT.setFwdIATMean(statistics.mean(self.fwdIAT))
  self.flowFeatures.iat.fwdIAT.setFwdIATMax(max(self.fwdIAT))
  self.flowFeatures.iat.fwdIAT.setFwdIATMin(min(self.fwdIAT))
  if len(self.fwdIAT) > 1:
  self.flowFeatures.iat.fwdIAT.setFwdIATStd(statistics.stdev(self.fwdIAT))
  if len(self.bwdIAT) > 0:
  self.flowFeatures.iat.bwdIAT.setBwdIATTotal(sum(self.bwdIAT))
  self.flowFeatures.iat.bwdIAT.setBwdIATMean(statistics.mean(self.bwdIAT))
  self.flowFeatures.iat.bwdIAT.setBwdIATMax(max(self.bwdIAT))
  self.flowFeatures.iat.bwdIAT.setBwdIATMin(min(self.bwdIAT))
  if len(self.bwdIAT) > 1:
  self.flowFeatures.iat.bwdIAT.setBwdIATStd(statistics.stdev(self.bwdIAT))
  packet_lens = [x.getPayloadBytes() for x in self.packetInfos]
  packet_sizes =[x.getPacketSize() for x in self.packetInfos]
  if len(packet_lens) > 0:
  self.flowFeatures.p_stats.flStats.setPacketLenMean(statis-
tics.mean(packet_lens))
  self.flowFeatures.p_stats.flStats.setAvgPacketSize(sum(packet_sizes)    /
self.packet_count)
  if len(packet_lens) > 1:
  self.flowFeatures.p_stats.flStats.setPacketLenStd(statis-
tics.stdev(packet_lens))
  self.flowFeatures.p_stats.flStats.setPacketLenVar(statistics.vari-
ance(packet_lens))
  if self.fwd_packet_count != 0:
  self.flowFeatures.flow_bytes.setMinSegSizeFwd(min(fwd_packet_lens))
  self.flowFeatures.flow_bytes.setAvgFwdSegmentSize(sum(fwd_packet_lens)   /
self.fwd_packet_count)
  if self.bwd_packet_count != 0:
  self.flowFeatures.flow_bytes.setAvgBwdSegmentSize(sum(bwd_packet_lens)   /
self.bwd_packet_count)
  if len(self.flowActive) > 0:
  self.flowFeatures.active_stats.setActiveMin(min(self.flowActive))
  if len(self.flowActive) > 1:
  self.flowFeatures.active_stats.setActiveSTD(statis-
tics.stdev(self.flowActive))
  if len(self.flowIdle) > 0:
  self.flowFeatures.idle_stats.setIdleMean(statistics.mean(self.flowIdle))
  self.flowFeatures.idle_stats.setIdleMax(max(self.flowIdle))
  self.flowFeatures.idle_stats.setIdleMin(min(self.flowIdle))
  if len(self.flowIdle) > 1:
  self.flowFeatures.idle_stats.setIdleStd(statistics.stdev(self.flowIdle))
  fwd_header_length = sum(packet[IP].ihl*4 if TCP in packet else 8 for packet
in self.fwdPacketInfos)
  bwd_header_length = sum(packet[IP].ihl*4 if TCP in packet else 8 for packet
in self.bwdPacketInfos)
  self.flowFeatures.flow_bytes.setFwdHeaderLength(fwd_header_length)
  self.flowFeatures.flow_bytes.setBwdHeaderLength(bwd_header_length)
  # Duplicated features
  self.flowFeatures.subflow.setSubflowBwdBytes(self.flowFeatures.getTotal-
LenBwdPackets)
  self.flowFeatures.subflow.setSubflowBwdPackets(self.flowFeatures.getTo-
talBwdPackets)
```

```
  self.flowFeatures.subflow.setSubflowFwdBytes(self.flowFeatures.getTo-
talLenFwdPackets)
  return [self.flowFeatures.getDestPort(),
  self.flowFeatures.getFlowDuration(),
  self.flowFeatures.p_len.getTotalFwdPacketsNum(),
  self.flowFeatures.flow_bytes.getTotalLenFwdPackets(),
  self.flowFeatures.p_stats.fwdStats.getFwdPacketLenMax(),
  self.flowFeatures.p_stats.fwdStats.getFwdPacketLenMean(),
  self.flowFeatures.p_stats.fwdStats.getFwdPacketLenStd(),
  #backward stats
  self.flowFeatures.p_stats.bwdStats.getBwdPacketLenMax(),
  self.flowFeatures.p_stats.bwdStats.getBwdPacketLenMin(),
  self.flowFeatures.p_stats.bwdStats.getBwdPacketLenMean(),
  self.flowFeatures.p_stats.bwdStats.getBwdPacketLenStd(),
  #Rates
  self.flowFeatures.p_time.getFlowBytesRate(),
  self.flowFeatures.p_time.getFlowPacketsRate(),

  #Flow Iat
  self.flowFeatures.iat.flowIAT.getFlowIATMean(),
  self.flowFeatures.iat.flowIAT.getFlowIATStd(),
  self.flowFeatures.iat.flowIAT.getFlowIATMax(),
  #fwd iat
  self.flowFeatures.iat.fwdIAT.getFwdIATTotal(),
  self.flowFeatures.iat.fwdIAT.getFwdIATMean(),
  self.flowFeatures.iat.fwdIAT.getFwdIATStd(),
  #headers
  self.flowFeatures.flow_bytes.getFwdHeaderLength(),
  self.flowFeatures.flow_bytes.getBwdHeaderLength(),
  self.flowFeatures.p_time.getBwdPacketsRate(),
  #flstats
  self.flowFeatures.p_stats.flStats.getMaxPacketLen(),
  self.flowFeatures.p_stats.flStats.getPacketLenMean(),
  self.flowFeatures.p_stats.flStats.getPacketLenStd(),
  self.flowFeatures.p_stats.flStats.getPacketLenVar(),
  self.flowFeatures.flags.getPSHFlag(),
  self.flowFeatures.p_stats.flStats.getAvgPacketSize(),
  #flags
  #segments
  self.flowFeatures.flow_bytes.getAvgFwdSegmentSize(),
  self.flowFeatures.flow_bytes.getAvgBwdSegmentSize(),

  #subflow
  self.flowFeatures.subflow.getSubflowFwdPackets(),
  self.flowFeatures.subflow.getSubflowFwdBytes(),
  self.flowFeatures.subflow.getSubflowBwdPackets(),
  self.flowFeatures.subflow.getSubflowBwdBytes(),
  #initbytes
  self.flowFeatures.init_win.getInitWinBytesFwd(),
  self.flowFeatures.init_win.getInitWinBytesBwd(),
  #active stat
  self.flowFeatures.flow_bytes.getMinSegSizeFwd(),
  self.flowFeatures.active_stats.getActiveMin()
  ]
```

Приложение В. Исходный код графического интерфейса

В листинге 4 представлен исходный код функции `runButton_was_clicked`.

Листинг 4 – Функция `runButton_was_clicked`

```
def runButton_was_clicked(self):
 try:
  self.worker = Worker()
  self.last_startTime = datetime.now().strftime("[%y.%m.%d;%H-%M-%S]")

  if self.selectedModel is None:
   self.selectedModel = self.modelComboBox.itemText(self.modelComboBox.cur-
rentIndex())
   self.worker.set_params(model_name=self.selectedModel)
   self.worker.set_params(model_path=self.availableModels[self.selected-
Model])
   self.worker.set_model()
   print(f"Selected model: {self.selectedModel}")

  if self.selectedInterface is None and self.selectedWorkMode!="Offline":
   self.selectedInterface = self.interfaceComboBox.itemText(self.interface-
ComboBox.currentIndex())
   self.worker.set_params(interface=self.selectedInterface)
   print(f"Selected interface: {self.selectedInterface}")

  if not self.selectedPCAP and self.selectedWorkMode=="Offline":
   print("Haven't selected PCAP")
   pass
  else:
   self.worker.set_params(pcap_path=self.selectedPCAP)

   self.worker.set_params(mode=self.selectedWorkMode)

   print("Run Button Clicked!")

   self.runButton.setEnabled(False)
   self.stopButton.setEnabled(True)

   self.modelComboBox.setEnabled(False)
   print("Model Selection is disabled")

   self.interfaceComboBox.setEnabled(False)
   print("Interface Selection is disabled")

   self.workMode.setEnabled(False)
   self.browseButton.setEnabled(False)

   self.run_thread()
 except:
  sys.stderr = self.old_stderr
  sys.stdout = self.old_stdout
  traceback.print_exc()
```