

Вайб-кодинг в кибербезопасности

М.А. Аграновский, М.Г. Аталаева, М.В. Варфоломеева, К.Г. Вититнев, В.В. Горбенко, А.Д. Горошко, М.А. Дыскина, Е.И. Лаврова, М.П. Майстер, Д.Е. Миронова, Д.К. Мурадян, М.И. Опришко, Е.М. Орлова, А.Г. Пионткевич, Е.С. Письменная, В.А. Рябыкин, Д.С. Федоров, У.Н. Хасанова, Д.Е. Намиот

Аннотация – В статье представлены результаты масштабного эксперимента, в котором команды студентов магистратуры ВМК МГУ имени М.В. Ломоносова в рамках курса по безопасности систем Интернета Вещей составляли программы классификации сетевого трафика с помощью больших языковых моделей (LLM). Использование моделей машинного (глубокого) обучения является классической историей для систем обнаружения вторжений. В качестве исходных данных на выбор предлагались доступные датасеты сайта Kaggle. Соответственно, на этом же сайте, вместе с датасетами можно было найти запрограммированные “вручную” модели, которые могли послужить базовыми примерами для сравнения с автоматическими решениями. Задача состояла в разработке подсказок (промтов), проверке их на нескольких LLM и, отдельно, проверке работы англоязычного и русскоязычного варианта подсказок. В целом, во всех проведенных экспериментах LLM работающие модели, которые не уступали по метрикам существующим “ручным” вариантам. Тем не менее, полученные результаты требовали проверки, равно как архитектурные решения, выбранные LLM, не всегда были оптимальными, а иногда и просто неверными, например, в части утечки данных. Это оставляет пока пространство для работы программистов (ML-инженеров).

Ключевые слова—искусственный интеллект, LLM, автоматизация программирования.

I. ВВЕДЕНИЕ

В работе приводятся результаты масштабного эксперимента по использованию LLM для кодирования прикладных решений в области кибербезопасности. В рамках учебного курса, ориентированного на Интернет вещей и безопасность инфраструктурных технологий [1], зачетное задание предлагало использовать LLM для “написания” программы анализа и классификации сетевого трафика. Это классическая тема для приложений Искусственного интеллекта в киберзащите. Множество работ описывает системы глубокого обучения для анализа сетевого трафика [2-5].

Обнаружение вредоносного трафика с помощью глубокого обучения включает в себя использование нейронных сетей для выявления закономерностей, указывающих на атаки (например, вредоносное ПО, DDoS-атаки или несанкционированный доступ). Оно основано на классификации трафика как безопасного или вредоносного на основе извлеченных признаков.

Типичные подходы к анализу вредоносного трафика состоят в анализе потоков (метаданных): длина пакета, интервал между поступлениями, длительность, флаги

протоколов и т.д.

С одной стороны, это, практически, табличные данные, с другой стороны, есть и свои проблемы. Подобного рода датасеты практически всегда не сбалансированы (вредоносный трафик, к счастью, пока не превалирует) [6], а механическое использование всех заявленных признаков (атрибутов) чревато утечкой данных в машинном обучении [7,8].

Задание, которое получала каждая из команд, состояло в следующем. Было представлено, на выбор, несколько датасетов с сайта Kaggle¹, содержащих размеченные данные о сетевом трафике. Датасеты выбирались исходя из их размеров, поскольку все вычисления проводились на локальных вычислительных устройствах участников. Вместе с датасетами, на соответствующих страницах Kaggle присутствовали и готовые решения, которые (код на языке Python) содержали модели классификации трафика, обученные на этих датасетах. Цель работы состояла в разработке подсказок (промтов), которые порождали бы такого же рода код для моделей машинного (глубокого) обучения для классификации трафика. При этом можно было бы сравнить метрики моделей, запрограммированных “вручную” и с помощью LLM. Другими вопросами была проверка созданных подсказок на нескольких LMM по выбору команды, а также сравнение работы англоязычного и русскоязычного варианта скриптов. Результаты работы каждой команды существуют в отдельных презентациях и выложены на отдельные страницы в Github.

Оставшаяся часть работы структурирована следующим образом. Разделы, озаглавленные Попытка 1, Попытка 2 и т.д. представляют решения различных команд. И последний раздел представляет собой заключение.

II. Попытка 1 (В.В. Горбенко, Д.С. Федоров)

Эксперимент проводился с датасетом InSDN².

Промпт: (в первой попытке приводим его полностью, чтобы показать основную идею. В последующих описаниях авторы могут отмечать только основные положения, полностью подсказки доступны на Github)

Роль: Ты — эксперт по машинному обучению в кибербезопасности, специализирующийся на сетях SDN.

Задача: Напиши Python-скрипт для построения классификатора обнаружения

¹ <https://kaggle.com>

² <https://www.kaggle.com/datasets/badcodebuilder/insdn-dataset/data>

атак на основе датасета InSDN.

Детали датасета:

Файлы: Есть три файла: Normal_data.csv, OVS.csv, metasploitable-2.csv. В задаче нужно объединить их в один датафрейм для обучения.

Целевая колонка: В каждом файле есть столбец с типом трафика (например, DoS, DDoS, Probe, Botnet, Benign). Название столбца может отличаться. Определи его автоматически и объедини классы Benign/Normal в одну категорию. В файле Normal_data.csv все записи – нормальные.

Особенность: Датасет сгенерирован CICFlowMeter и содержит 252 столбца. Многие из них могут содержать бесконечности (inf) или пропуски.

Требования к скрипту:

Загрузка и объединение: Загрузи три CSV, объедини их, добавь колонку-источник (опционально). Проверь, что во всех файлах одинаковый набор колонок.

Предобработка:

Удали колонки, которые полностью пустые или содержат только одно уникальное значение.

Замени inf и -inf на NaN, затем заполни все NaN нулями.

Если есть временные метки или колонки, содержащие IP-адреса (например, Src IP, Dst IP), удали их, так как они могут привести к переобучению.

Целевая переменная: Создай целевую переменную Label. Все, что не Benign/Normal, пометь как Attack для бинарной классификации. Дополнительно: реализуй возможность многоклассовой классификации для определения конкретного типа атаки (используй существующие названия из столбца).

Балансировка: Используй SMOTE (из imblearn) только на тренировочной выборке.

Модель: Обучи Random Forest и XGBoost. Сравни их производительность.

Оценка: Выведи classification_report (precision, recall, f1-score) для бинарного и многоклассового сценариев. Построй матрицу ошибок (confusion matrix) в виде тепловой карты, чтобы видеть, какие типы атак путаются между собой.

Визуализация: Отобрази топ-20 наиболее важных признаков (feature importance) для лучшей модели.

Напиши код с подробными комментариями.

Учти, что датасет может быть большим (>300k строк), поэтому добавь аргумент n_rows для тестового запуска на части данных.

Как видно, здесь довольно подробно указывается архитектура требуемой модели (моделей). Автор явно понимает, что он хочет получить и использует LLM только как кодировщика.

Использовались 3 LLM: DeepSeek, GPT-4, Qwen 3.5-Plus. Итоговое достижение представлено в таблице 1:

Таблица 1

LLM	Размер кода (LOC)	Метрика F1
DeepSeek	308	0.998
GPT-4	302	0.999
Qwen 3.5+	667	0.997

Для сравнения: объем “ручного” кода 300-700 строк, F1 – 0.99. Но время разработки 6-8 часов, вместо 20-30 минут при генерации (составление подсказки и генерация). Полностью решение представлено в авторском Github³.

III. Попытка 2 (М.А. Аграновский, М.В. Варфоломеева, М.П. Майстер)

Для проведения эксперимента использовался набор данных сетевого трафика⁴, содержащий информацию о нормальных сетевых пакетах и пакетах, сгенерированных в ходе DDoS-атак. Набор включает 47 признаков, описывающих различные характеристики сетевых потоков. Ключевые группы признаков:

1. Идентификаторы и временные метки: Unnamed: 0, pkSeqID, stime, ltime, seq (техническая информация о порядке пакетов).
2. Сетевые адреса: saddr (Source IP), daddr (Destination IP), sport (Source Port), dport (Destination Port).
3. Метрики потока: pkts (количество пакетов), bytes (объем в байтах), rate (скорость передачи), dur (продолжительность соединения).
4. Целевые переменные: category (основной класс: DDoS или Normal), subcategory (конкретный тип атаки), attack (числовой индикатор атаки).

Особенностью датасета является экстремальный дисбаланс классов: около 2 000 000 записей вредоносного трафика (DDoS) и всего 477 записей нормального (Normal). Данная специфика требует от генерируемого кода обязательного применения методов балансировки для предотвращения переобучения моделей на мажоритарный класс.

Для генерации кода анализа данных и построения моделей машинного обучения использовалась единая подсказка (промт), направленная на проверку способности языковых моделей самостоятельно

³ <https://github.com/FMbic/cyberiot>

⁴ <https://www.kaggle.com/datasets/siddharthm1698/ddos-botnet-attack-on-iot-devices>

принимать архитектурные решения.

Ты — Data Scientist. Мне нужно создать Python-приложение в формате Jupyter Notebook для классификации сетевого трафика на нормальный (Normal) и вредоносный (DDoS).

Входные данные:

Датасет содержит следующие признаки: Unnamed: 0, pkSeqID, stime, flgs, flgs_number, proto, proto_number, saddr, sport, daddr, dport, pkts, bytes, state, state_number, ltime, seq, dur, mean, stddev, sum, min, max, spkts, dpkts, sbytes, dbytes, rate, srate, drate, TnBPSrcIP, TnBPDstIP, TnP_PSrcIP, TnP_PDstIP, TnP_PerProto, TnP_Per_Dport, AR_P_Proto_P_SrcIP, AR_P_Proto_P_DstIP, N_IN_Conn_P_DstIP, N_IN_Conn_P_SrcIP, AR_P_Proto_P_Sport, AR_P_Proto_P_Dport, Pkts_P_State_P_Protocol_P_DestIP, Pkts_P_State_P_Protocol_P_SrcIP, attack, category, subcategory.

Целевая переменная: 'category' (принимает значения 'DDoS' или 'Normal'). В датасете присутствует критический дисбаланс классов: около 2 000 000 записей DDoS и всего около 500 записей Normal.

Твоя задача — предложить решение и написать код, разбив его на логические блоки (ячейки Jupyter Notebook) такие, как загрузка и анализ данных, подготовка данных к обучению (что удалить, как работать с дисбалансом), обучение минимум двух моделей и оценка результатов с демонстрацией матрицы ошибок и сохранением результата в result.csv

В тестировании приняли участие три языковые модели: Gemini 3.1 Pro (бесплатная версия), AliceAI (отечественная модель от Yandex) и ChatGPT 5.5 Plus (по подписке). Все модели успешно сгенерировали синтаксически корректный Python-код, разбитый на логические ячейки для Jupyter Notebook, и обеспечили вывод промежуточных результатов. Однако подходы к машинному обучению существенно различались.

Gemini 3.1 Pro. Модель успешно распознала и удалила признаки, ведущие к утечке данных, включая attack и subcategory, а также сетевые идентификаторы. Для борьбы с дисбалансом был применен метод undersampling (соотношение 1:10), после чего данные были разделены на обучающую и тестовую выборки. Обученные модели (Random Forest и LightGBM) показали 100% точность (F1-macro: 1.00), что характерно для синтетических DDoS-датасетов при успешном выделении паттернов.

AliceAI. Модель допустила критическую ошибку в проектировании признаков. Она удалила некоторые идентификаторы, но оставила прямые признаки-утечки (attack, subcategory). Кроме того, стратегия балансировки оказалась спорной: undersampling применялся только к части нормального трафика, что исказило распределение. Обученные Logistic Regression и Random Forest показали F1-macro 0.99-1.00, однако из-за утечки данных эти результаты сложно считать валидными.

ChatGPT. Модель продемонстрировала наиболее профессиональный подход. В код были встроены функции оптимизации потребления памяти и защиты от ошибок отсутствия классов. Важным архитектурным решением стало разделение данных на Train/Test до балансировки. Это позволило сохранить в тестовой выборке реальное (экстремальное) распределение классов. Обученные модели (SGDClassifier, Random Forest, HistGradientBoosting) показали макро-F1 на уровне 0.96-0.97 на несбалансированном тесте.

Ручной подход предполагал создание полносвязной нейронной сети (MLP) на базе фреймворка PyTorch. Был реализован сложный конвейер обучения: применение метода SMOTE для балансировки классов (что увеличило объем данных до 2.6 млн строк), ручная настройка загрузчиков данных (DataLoader), использование планировщика шага обучения (ReduceLROnPlateau) и механизмов ранней остановки (early stopping). Разработка такого кода заняла около двух часов, а процесс обучения на CPU потребовал более 30 минут. Это решение от человека получило тот же 100% результат, что и решающее дерево от Gemini.

Все решения представлены в авторском GitHub⁵.

Проведенный эксперимент выявил качественные различия в подходах ИИ к проектированию систем машинного обучения: бесплатная версия Gemini сгенерировала лаконичный и точный код за несколько десятков секунд, успешно предотвратив утечку данных; подписочная версия ChatGPT потребовала значительно больше времени на генерацию, но предоставила наиболее проработанное решение с оптимизацией памяти, защитой от граничных условий и продвинутой валидацией; отечественная модель AliceAI продемонстрировала верное понимание современных ML-концепций и пайплайнов, подтвердив сокращение технологического отставания российских разработок от мировых лидеров до 1–1,5 лет, хотя и упустила из виду фильтрацию колонок-утечек. В заключение можно утверждать, что интеграция LLM в процесс разработки систем кибербезопасности демонстрирует высочайшую эффективность и кардинально снижает порог входа в отрасль, однако роль человека-эксперта остается критически важной — именно специалист должен осуществлять финальный контроль архитектурной логики, верифицировать методы обработки данных и решать задачи качественного prompt-инжиниринга.

⁵ <https://github.com/MeisterMich/The-Security-of-infrastructure-technologies.-Credit-Project>

IV Попытка 3 (М.Г. Аталаева, Д.К. Мурадян)

Эксперимент проводился на выборке CICIOT2023 объемом 1 048 575 записей с 47 признаками. Доля легитимного трафика BenignTraffic составляла 2,26%. Для генерации программ использовались Gemini 3.5 Flash, Claude 4.7 Opus и GPT 5.5. Полный промпт, исходные тексты программ и результаты представлены в репозитории команды⁶.

Всем трем LLM был направлен одинаковый русскоязычный промпт. Основные требования включали выборку 10% данных для снижения потребления памяти, очистку названий столбцов, кодирование целевой переменной, бинарную классификацию «норма/атака» и многоклассовую классификацию по группам атак. Предлагалось использовать Random Forest и LightGBM или XGBoost, стратифицированное разделение данных на обучающую и тестовую выборки в пропорции 80/20, отчеты о качестве и матрицы ошибок. В опубликованных программах оценка выполняется на отложенной тестовой выборке.

Программы Claude и GPT запустились с первой попытки, программа Gemini потребовала ручных исправлений и запустилась со второй. При этом успешный запуск не означал правильного решения задачи. Gemini не распознала BenignTraffic как нормальный трафик и сформировала бинарную цель DDoS-UDP_Flood против всех остальных записей. Поэтому ее результат в сценарии А исключен из сравнения. В сценарии Б Gemini сохранила отдельные метки и обучала модели на 32 классах. Claude и GPT выполнили группировку в восемь категорий: семь семейств атак и нормальный трафик.

Различались и способы работы с памятью. В программе Gemini выборка 10% выполняется после полной загрузки CSV. Claude предусмотрела дополнительный режим чтения файла частями с выборкой внутри каждой части, а также уменьшение разрядности числовых типов. GPT также использовала уменьшение разрядности типов и организовала заполнение пропусков и кодирование категориальных признаков через Pipeline и ColumnTransformer. Обучение этих преобразований внутри конвейера на тренировочных данных позволяет избежать использования статистик тестовой выборки на данном этапе предобработки.

В таблице 2 приведены результаты бинарной классификации. Из-за дисбаланса классов Accuracy дополнялась Macro-F1, усредняющей F1 по классам с одинаковым весом. Классификатор, всегда предсказывающий атаку, получил бы Accuracy 97,74%, не распознавая нормальный трафик. Для Random Forest, сгенерированного GPT, F1 нормального трафика составила 0,9191, а F1 атак — 0,9981; для варианта Claude — 0,9180 и 0,9981 соответственно.

Таблица 2. Бинарная классификация

LLM / алгоритм	Accuracy, %	Macro-F1
Claude / RF	99,63	0,9581
GPT 5.5 / RF	99,64	0,9586
GPT 5.5 / LGBM	99,59	0,9551
GPT 5.5 / XGB	99,57	0,9514

В многоклассовом сценарии (таблица 3) высокая Accuracy сочеталась с заметно меньшей Macro-F1. Среди приведенных результатов для восьми категорий наибольшее значение Macro-F1 получила Random Forest в программе Claude: 0,7474. У GPT с Random Forest оно составило 0,7470, тогда как максимальная Accuracy 99,37% достигалась с XGBoost. Для LightGBM в программе GPT F1 категории Web равнялась 0,00. Таким образом, выбор модели только по общей точности скрывает ошибки распознавания редких атак.

Таблица 3. Многоклассовая классификация

LLM / алгоритм	Классы	Accuracy, %	Macro-F1
Claude / RF	8	99,30	0,7474
GPT 5.5 / RF	8	99,29	0,7470
GPT 5.5 / LGBM	8	99,32	0,7164
GPT 5.5 / XGB	8	99,37	0,7260
Gemini / RF	32	99,00	0,7200
Gemini / LGBM	32	28,00	0,1500

Обозначения в таблицах: RF — Random Forest, LGBM — LightGBM, XGB — XGBoost. Строки Gemini относятся к иной целевой задаче с 32 классами и не являются прямым сравнением качества с вариантами на восьми категориях. В частности, низкие показатели LightGBM в программе Gemini характеризуют конкретный запуск; для установления причин необходим отдельный анализ.

Дополнительная проблема связана с редкими классами: после случайной выборки 10% данных для некоторых меток остаются единичные записи, что затрудняет стратифицированное разделение. Группировка по семействам уменьшает эту проблему. В программе Claude также предусмотрено исключение групп с числом записей ниже заданного порога, по умолчанию 10. Однако фильтрация сужает набор оцениваемых классов, что необходимо учитывать при интерпретации метрик.

Эксперимент показал, что LLM могут генерировать работоспособные программы классификации IoT-трафика, однако корректность целевой переменной, предобработки и оценки качества требует проверки специалистом. Преимущества Claude и GPT в данном опыте проявились в обработке нормального трафика, группировке атак и организации кода. Небольшие различия метрик сами по себе не доказывают устойчивого превосходства одной LLM. Рассмотренное сравнение относится к одному русскоязычному промпту; сопоставление языков промпта и оценка выигрыша

⁶ https://github.com/Khaotic10/IoT_proj

относительно ручной разработки в его результаты не входят.

V ПОПЫТКА 4 (М.А. Дыскина, Е.И. Лаврова, Д.Е. Миронова)

Эксперимент проводился с датасетом N-BalIoT⁷, содержащим около 7 млн записей реального сетевого трафика девяти коммерческих IoT-устройств, заражённых в лабораторных условиях ботнетами Mirai и BASHLITE. Каждая запись описывается 115 числовыми признаками. Это не содержимое пакетов, а поведенческие статистики потоков: количество пакетов, средний размер и дисперсия размера, джиттер, ковариации и корреляции между потоками, вычисленные для нескольких уровней агрегации и нескольких временных окон затухания.

Постановка задачи в нашем случае отличается от остальных разделов: она решалась как обнаружение аномалий без учителя, а не как классификация. Мы следовали схеме авторов датасета, в которой модель обучается исключительно на нормальном трафике и не видит ни одного примера атаки. Меток внутри CSV-файлов нет, класс записи определяется тем, из какого файла она взята. Под обучение отводится 2/3 нормального трафика, в тестовую выборку идёт оставшаяся треть вместе со всеми записями атак. Балансировка обучающей выборки при таком подходе не требуется, однако тестовая выборка оказывается крайне несбалансированной: атак в ней больше нормального трафика в 16–200 раз в зависимости от устройства.

Подсказка, написанная одной из тестируемых моделей, даёт этой модели преимущество, поэтому был применён мета-промптинг. Сначала составлялась подсказка для генерации подсказок, затем она передавалась модели Codex (GPT-5.5), не участвовавшей в основном эксперименте. Мета-промпт был написан на английском языке и задавал Codex роль эксперта по проектированию подсказок, а структуру итоговых промптов фиксировал четырьмя компонентами: «Роль — определить, кем является LLM (генератор Python-кода для кибербезопасности); Контекст — объяснить задачу (обнаружение ботнетов по аномалиям с помощью автоэнкодеров в трафике IoT); Данные — описать структуру датасета, чтобы модель понимала, с чем работает; Шаги — задать конкретную последовательность действий, которые должен выполнить сгенерированный скрипт». Отдельно оговаривалось, что оба промпта должны быть семантически идентичны, а русскоязычный — написан «естественным техническим русским языком, а не быть механическим переводом»; последнее требование существенно, поскольку иначе сравнение языков подсказки измеряло бы качество перевода, а не влияние языка. Codex сгенерировал англоязычный и русскоязычный промпты, и уже они подавались трём тестируемым моделям.

Основные требования к генерируемой программе

включали приём пути к данным и номера устройства через аргументы командной строки, обработку бесконечностей и пропусков, масштабирование признаков MinMaxScaler'ом, обученным только на тренировочной части, построение глубокого автоэнкодера из четырёх слоёв энкодера с «бутылочным горлышком» в 25 % от размерности входа и симметричным декодером, вычисление порога аномальности как суммы среднего и стандартного отклонения ошибки реконструкции на валидационной выборке, а также вывод accuracy, precision, recall, F1 и матрицы ошибок. Как видно, архитектура здесь задана полностью. Это было сделано намеренно: при свободном выборе модели различия в метриках отражали бы удачность случайно выбранной архитектуры, а не качество кодогенерации, и LLM в нашей постановке выступает именно как кодировщик.

Использовались 3 LLM, выбранные из разных стран разработки: Claude Opus 4.6, DeepSeek V4 и GigaChat. Каждая получала обе версии подсказки в отдельных диалогах, что дало шесть программ. В качестве «ручного» решения были реализованы Isolation Forest и One-Class SVM. Выбор определялся постановкой задачи: оба метода, как и автоэнкодер, обучаются только на нормальном трафике, тогда как случайный лес или градиентный бустинг потребовали бы размеченных примеров атак и в этой постановке неприменимы.

Таблица 4. Обнаружение аномалий на устройстве 1 (Danmini Doorbell)

LLM	Промпт	Acc	Prec	Rec	F1	LOC	Ош
Claude Opus 4.6	EN	0.9997	0.9997	1.00	0.9999	416	0
Claude Opus 4.6	RU	0.9998	0.9998	1.00	0.9999	482	0
DeepSeek V4	EN	0.9999	1.00	1.00	1.00	387	0
DeepSeek V4	RU	0.9999	1.00	1.00	1.00	430	2
GigaChat	EN	0.9999	1.00	1.00	1.00	258	0
GigaChat	RU	1.00	1.00	1.00	1.00	211	0

LOC — объём сгенерированного кода в строках;

Ошибки — количество дефектов, потребовавших ручного исправления для запуска. Результат GigaChat RU обсуждается ниже.

Результаты на первом устройстве приведены в таблице 4. Различия между шестью программами укладываются в четвёртый знак, и модели практически неразличимы по метрикам. Это ожидаемо: авторы датасета сообщают о 100 % TPR, а Mirai и BASHLITE меняют поведение заражённого устройства настолько резко, что ошибка реконструкции превышает порог на порядок. Содержательные различия проявились в качестве кода. Claude сгенерировал наиболее развёрнутый вариант с именованными слоями и подробными комментариями, GigaChat уложился вдвое короче, выполнив при этом все требования подсказки. Программа DeepSeek, полученная по русскоязычной подсказке, не запустилась с первой

⁷ <https://www.kaggle.com/datasets/mkashifn/nbaiot-dataset>

попытки: CSV читались с параметром `header=None`, из-за чего строка заголовков попадала в данные, а функция оценки обращалась к неинициализированной переменной. Оба дефекта потребовали ручного исправления, остальные пять программ запустились без вмешательства.

Отдельного комментария требует результат GigaChat по русскоязычной подсказке. Эта конфигурация даёт ассурасу ровно 1,00 на всех девяти устройствах, включая те, где остальные модели показывают 0,37–0,76. Такая

устойчивость к смене данных нехарактерна для корректно поставленного эксперимента и указывает на утечку данных в сгенерированном конвейере. Показательно, что подсказка обязывала обучать масштабирование на тренировочной выборке и вычислять порог по валидационной, но не содержала прямого запрета на использование тестовых данных. По этой причине конфигурация исключена из сводного сравнения.

Таблица 5. Accurasy по девяти устройствам

Устр-во	Claude EN	Claude RU	Deep Seek EN	DeepSeek RU	Giga Chat EN	IF	OC-SVM
1 Danmini Doorbell	0.9997	0.9998	0.9999	0.9999	0.9999	0.798	0.9998
2 Ecobee Thermostat	0.9998	0.9998	0.9999	0.9999	0.9999	0.429	0.762
3 Ennio Doorbell	0.375	0.376	0.377	0.377	0.376	0.040	0.9993
4 Philips Baby Monitor	0.9978	0.796	0.9983	0.9984	0.9978	0.059	0.9986
5 Provision PT-737E	0.734	0.734	0.9995	0.9997	0.9991	0.026	0.9994
6 Provision PT-838	0.748	0.748	0.9998	0.9997	0.9994	0.374	0.9994
7 Samsung Webcam	0.9976	0.9977	0.9988	0.9988	0.9982	0.051	0.390
8 Simple Home 1002	0.9991	0.9992	0.9994	0.9992	0.9992	0.426	0.9997
9 Simple Home 1003	0.760	0.760	0.760	0.760	0.760	0.078	0.9998

IF — Isolation Forest, OC-SVM — One-Class SVM на подвыборке 20 тыс. записей. Конфигурация GigaChat RU исключена (см. выше).

Сводные результаты по девяти устройствам приведены в таблице 5. На шести устройствах все конфигурации дают ассурасу выше 0,99. Устройства 3 и 9 дают низкий результат у всех моделей без исключения — около 0,38 и 0,76, причём значения совпадают до третьего знака. Совпадение у независимо сгенерированного кода означает, что причина лежит в методике, а не в кодогенерации: фиксированный порог оказывается неуниверсальным. Дверной звонок Ennio большую часть времени находится в режиме ожидания и порождает однообразный трафик, разброс ошибки реконструкции на нормальных данных мал, порог получается низким и перестаёт отделять «тихие» атаки вроде сканирования. На устройствах 5 и 6 обнаружилось единственное содержательное различие между моделями: Claude в обеих языковых версиях даёт 0,73–0,75, тогда как DeepSeek и GigaChat удерживают 0,999. Поскольку архитектура задавалась одинаково, различие объясняется выбором гиперпараметров, оставленных на усмотрение модели.

Isolation Forest показал себя крайне неустойчивым — от 0,798 до 0,026. Причина двойная: метод строит случайные разбиения в 115-мерном пространстве, не учитывая структуру данных, а заданное значение contamination задаёт слишком строгий порог, из-за чего

большая часть атак классифицируется как норма, а атаки составляют подавляющее большинство тестовой выборки. One-Class SVM по метрикам сопоставим с автоэнкодером, однако из-за квадратичной сложности применялся к подвыборке в 20 тыс. записей из почти миллиона, и результат оказался чувствителен к её составу: на устройствах 2 и 7 ассурасу падает до 0,762 и 0,390. Автоэнкодер обрабатывал полный объём данных без ограничений.

Эксперимент показал, что рабочее приложение для обнаружения ботнет-атак получается из одной подсказки: пять программ из шести запустились без правок. Язык подсказки практически не повлиял на метрики, различия лежат в четвёртом знаке, однако проявился иначе. Все три модели, получив русскоязычный текст, без явного указания перешли на русский язык в комментариях к коду. Объём кода у Claude и DeepSeek вырос на 11–16 %, а у GigaChat, напротив, сократился на 18 %. Единственные дефекты кода за весь эксперимент встретились именно в русскоязычной версии. Главный отрицательный результат касается утечки данных: требования к разделению выборок подсказка описывала корректно, но прямого запрета на использование тестовых данных в ней не было, и одна конфигурация из шести этим воспользовалась. Практическая рекомендация состоит в том, чтобы включать такой запрет непосредственно в текст подсказки. Наконец, одинаково низкие метрики у

независимо сгенерированных программ на устройствах 3 и 9 показывают, что корректная кодогенерация не спасает от неудачного архитектурного решения, заложенного в саму подсказку: способ определения порога аномальности LLM не оспаривала, а добросовестно воспроизводила. Полностью решение представлено в авторском Github⁸.

VI Попытка 5 (М.И. Опышко, А.Г. Пионткевич, Е.С. Письменная, В.А. Рябыкин)

Эксперимент проводился на наборе Network Malware Detection – Connection Analysis, содержащем размеченные записи сетевого трафика Интернета вещей из Stratosphere Laboratory. Использовался файл STU-IoT-Malware-Capture-1-1conn.log.labeled.csv, включающий 1 008 748 записей о соединениях и 23 столбца, в том числе поля разметки. Рассматривалась бинарная классификация: нормальному трафику (Benign) присваивалась метка 0, вредоносному (Malicious) – 1. Для оценки использовались доля правильных ответов (accuracy), точность (precision), полнота (recall) и F1-мера. Последние три метрики рассчитывались для класса Malicious.⁹

Исходный промпт был подготовлен с помощью Comproser 2.0 и отредактирован вручную. Русская и английская версии содержали одинаковые основные требования, описание структуры CSV и примеры записей. Требовалось сформировать Python-приложение в одном файле malware_classifier.py, реализующее загрузку и предобработку данных, исключение идентификаторов и IP-адресов, стратифицированное разбиение выборки с фиксированным random_state, расчет метрик и матрицы ошибок, а также сохранение

обученного конвейера. Набор допустимых библиотек ограничивался pandas, numpy, scikit-learn и joblib. Алгоритм классификации и его гиперпараметры выбирала LLM.

Для генерации кода использовались GPT-5.5, Claude 4.6 Opus и Yandex Alice AI. Каждая LLM оценивалась с русским и английским промптами; всего сравнивались шесть вариантов программы. В обоих вариантах GPT-5.5 использовался Random Forest с 200 деревьями без ограничения глубины, а в вариантах Alice AI – Random Forest со 100 деревьями. Для Claude английскому промпту соответствовал Random Forest с 200 деревьями и максимальной глубиной 25, русскому – Gradient Boosting с 300 деревьями и максимальной глубиной 6. Базовым решением служил классификатор Random Forest с 300 деревьями, запрограммированный вручную.

Сопоставимость результатов обеспечивалась общей процедурой загрузки данных и фиксированным разбиением выборки. Загрузчик обрабатывал обозначения пропусков, удалял пустые и постоянные столбцы. Из признаков исключались идентификаторы соединений, временная метка, IP-адреса, порты, поля history и tunnel_parents, а также дополнительная разметка detailed-label. Стратифицированное разбиение в отношении 80:20 при random_state=42 давало 806 998 обучающих и 201 750 тестовых записей. Из каждой программы вызывалась функция построения конвейера предобработки и классификации; полученный конвейер обучался и оценивался на соответствующих частях общей выборки. Приведенные метрики характеризуют эти конвейеры при унифицированной загрузке данных и не являются оценкой всех этапов автономного запуска сгенерированных приложений.

Результаты сравнения и объем программ приведены в таблице 6.

Таблица 6. Результаты эксперимента третьей группы

LLM / реализация	Язык промпта	Accuracy	Precision	Recall	F1	Число строк кода
GPT-5.5	EN	0,956198	0,925025	0,999073	0,960624	515
GPT-5.5	RU	0,956203	0,925018	0,999092	0,960629	581
Claude 4.6 Opus	EN	0,956565	0,924970	0,999889	0,960972	319
Claude 4.6 Opus	RU	0,956570	0,924978	0,999889	0,960976	302
Yandex Alice AI	EN	0,956213	0,925012	0,999120	0,960639	171
Yandex Alice AI	RU	0,956213	0,925012	0,999120	0,960639	171
Ручная реализация	–	0,956550	0,924990	0,999833	0,960957	156

Тестовая выборка – 201 750 записей. Precision, recall и F1 рассчитаны для класса Malicious. EN – английский промпт, RU – русский.

На тестовой выборке все решения показали близкие значения accuracy (около 0,956) и F1 (около 0,961). Максимальная F1 среди сгенерированных вариантов составила 0,960976 и была получена для Claude с русским промптом. Превышение над базовым решением равнялось 0,000019, а разность между максимальной и

минимальной F1 сгенерированных вариантов – 0,000352. Оценка на одном разбиении без анализа вариативности результатов не позволяет установить статистическую значимость этих различий. При полноте свыше 0,999 точность составляла около 0,925: примерно 7,5% соединений, отнесенных к вредоносным, принадлежали

⁸ <https://github.com/MariaDys/llm-botnet-detection>

⁹ <https://www.kaggle.com/datasets/agungpambudi/network-malware-detection-connection-analysis>

классу нормального трафика. Следовательно, высокая полнота обнаружения вредоносных соединений сопровождалась ложными срабатываниями.

В пределах проведенного сравнения переход от английского промпта к русскому не сопровождался снижением F1. Разность значений для русской и английской версий составила 0,000005 у GPT-5.5 и 0,000004 у Claude; у Alice AI значения совпали с точностью до шести знаков. Для Claude близкие результаты получены при использовании разных алгоритмов классификации. Эти наблюдения относятся к представленным вариантам генерации и не доказывают независимость качества от языка промпта в общем случае. В программах, полученных по русским запросам, комментарии также были написаны на русском языке, хотя отдельное требование к их языку не задавалось.

Объем сгенерированных программ составлял от 171 до 581 строки при 156 строках в ручной реализации (табл. 2). Увеличение объема кода не сопровождалось более высокими метриками: наиболее длинные программы GPT-5.5 имели несколько меньшую F1, чем базовый классификатор. Число строк рассматривалось как описательная характеристика программ; трудоемкость их разработки и проверки количественно не оценивалась.

Основное ограничение эксперимента связано с использованием случайного разбиения записей одного файла. Качество на независимом сетевом захвате и при разделении данных по времени не оценивалось. Кроме того, определение типов признаков и удаление постоянных столбцов выполнялись до выделения тестовой выборки. Для более строгой оценки эти операции необходимо выполнять только по обучающим данным, а полученные правила применять к тестовым без повторной настройки. Результаты подтверждают возможность генерации прототипов классификаторов, сопоставимых с ручной реализацией по выбранным метрикам в условиях данного эксперимента. Для оценки их применимости к другим условиям требуется проверка на независимом трафике.

Промпты на двух языках, исходный код шести сгенерированных программ и базового классификатора, сценарий сравнения, результаты и презентация размещены в репозитории команды на GitHub.¹⁰

VII Попытка 6 (К.Г. Вититнев, Е.М. Орлова)

Эксперимент проводился с датасетом IoT Intrusion Detection Dataset, содержащим 1 048 575 записей и 47 столбцов. Целевой столбец label содержит классы нормального и вредоносного сетевого трафика. Датасет является существенно несбалансированным: крупные классы представлены десятками тысяч записей, тогда как для некоторых редких типов атак число примеров составляет лишь несколько десятков. По этой причине основной метрикой при сравнении моделей был выбран macro F1-score, позволяющий учитывать качество классификации редких классов.

Целью работы была проверка возможности использования российских LLM для генерации Python-приложения классификации IoT-трафика. Сравнивались

ручная реализация и код, сгенерированный YandexGPT 5 Pro и GigaChat. Обе LLM получили одинаковый русскоязычный промпт с требованиями автоматически загрузить датасет, определить целевой столбец, выполнить предобработку данных, обучить несколько моделей машинного обучения, рассчитать метрики и сохранить лучшую модель. В качестве алгоритмов использовались Random Forest, Logistic Regression и Decision Tree.

Сгенерированный код в обоих случаях потребовал дополнительной проверки и уточняющих промптов. В первой версии, предложенной YandexGPT, было жестко указано неверное имя CSV-файла, из-за чего программа завершалась с FileNotFoundError. После исправления загрузки появились ошибки в логике выбора лучшей модели: рассчитанные метрики сохранялись в одной структуре, а последующая функция пыталась обратиться к ключам accuracy и F1 в другой структуре. После нескольких точечных исправлений программа была успешно выполнена. Лучший результат показала модель Decision Tree: accuracy — 0,9930, precision macro — 0,8455, recall macro — 0,8384, F1 macro — 0,8399.

GigaChat в первой версии допустил аналогичную ошибку с жестко заданным именем файла. После ее исправления возникла более существенная методологическая проблема: модель пыталась определить целевой столбец по числу уникальных значений и рассматривала в качестве возможной цели бинарные признаки: HTTP, TCP, UDP, ICMP и другие. Настоящий целевой столбец label содержал более десяти классов и поэтому не проходил заданное самой LLM условие. После уточняющего промпта логика была изменена: программа стала в первую очередь искать столбцы с характерными названиями: label, class, attack, attack_type, category. Финальная версия успешно выполнила обучение трех моделей. Лучший результат также показала Decision Tree с accuracy 0,9930 и F1 macro 0,8290.

Для сравнения была написана ручная baseline-реализация. В ней Decision Tree достигла F1 macro 0,8294, а Random Forest — 0,7369. Таким образом, итоговый результат GigaChat практически совпал с ручной реализацией Decision Tree, а у YandexGPT получилось немного более высокое значение macro F1.

Таблица 7 Результат эксперимента попытки 6

Подход	Лучшая модель	Accuracy	Precision macro	Recall macro	F1 macro
YandexGPT	Decision Tree	0,9930	0,8455	0,8384	0,8399
Ручная реализация	Decision Tree	0,9928	0,8305	0,8297	0,8294
GigaChat	Decision Tree	0,9930	0,8310	0,8300	0,8290
GigaChat	Random Forest	0,9930	0,8090	0,7250	0,7370
Ручная реализация	Random Forest	0,9932	0,8095	0,7248	0,7369

¹⁰ https://github.com/VladislavRya/CMC_iot_prompt_project

Результаты показывают, что обе российские LLM оказались способны сформировать работоспособный ML-пайплайн и получить метрики, сопоставимые с ручной реализацией. При этом ни одна из сгенерированных программ не получилась полностью корректной с первой попытки. Наиболее существенными оказались не синтаксические, а архитектурные и логические ошибки: неправильное определение целевой переменной, жестко заданные параметры входных данных и неверная логика выбора лучшей модели.

Отдельно следует отметить влияние дисбаланса данных. Например, Random Forest показывал ассигасу около 0,993, однако его macro F1 составлял только 0,737. Высокое значение ассигасы в данном случае не означает одинаково качественного распознавания всех типов атак: для ряда редких классов модели не формировали ни одного правильного предсказания. Поэтому выбор метрики также остается задачей разработчика и не должен полностью передаваться LLM.

Полностью решение представлено в Github¹¹.

В предоставленных материалах эксперимента использован русскоязычный промпт. Файл `prompt_en.md` в репозитории не содержит текста, поэтому сравнение результатов русскоязычной и англоязычной версий подсказки для данной попытки не проводилось.

VIII Попытка 7 (А.Д. Горошко, У.М. Хасанова)

Эксперимент проводился на датасете InSDN¹², собранном в программно-конфигурируемой сети на базе Mininet и контроллера ONOS. Датасет содержит три CSV-файла: `Normal_data.csv` (68424 записи нормального трафика, 20%), `OVS.csv` (136743 записи атак на OVS-коммутатор, 39,8%) и `metasploitable-2.csv` (138 772 записи атак на уязвимый сервер

Metasploitable 2, 40,3 %). Суммарный объем — около 344 тыс. записей, описанных более чем 80 признаками, извлеченными CICFlowMeter. В датасете встречаются следующие типы атак: DoS, DDoS, Probe, brute-force, Web_attack, Botnet, Exploitation.

Для генерации программ использовались две LLM: DeepSeek и GigaChat. Каждая модель получала два семантически эквивалентных промпта — русскоязычный и англоязычный, что дало четыре программы. В качестве эталона использовалась модель, написанная человеком, доступная на Kaggle¹³.

Основные требования к генерируемой программе включали: загрузку всех CSV-файлов и объединение их в один DataFrame; формирование бинарной целевой переменной `label` (0 — норма, 1 — атака) и многоклассовой `attack_type` (тип атаки / benign); обработку пропусков заполнением нулем или медианой; удаление константных признаков; стратифицированное разделение на train/val/test в пропорции 60/20/20 с учетом типа атаки; обучение трех моделей бинарной классификации (LogisticRegression, RandomForest, GradientBoosting) с подбором гиперпараметров через GridSearchCV; вывод метрик ассигасы, precision, recall, F1, ROC-AUC и матрицы ошибок; сохранение результатов предсказаний, метрик и обученных моделей; фиксированный `random_state=42`.

Бинарная классификация. В таблице 8 приведены результаты бинарной классификации на логистической регрессии. Все четыре сгенерированные модели показали результаты, близкие к эталону, а по части метрик его превосходили.

Таблица 8 Бинарная классификация

Модель	Acc	Prec	Rec	F1	ROC-AUC
Benchmark	0,9337	0,9500	0,9700	0,9600	0,9690
DeepSeek (RUS)	0,9962	—	—	0,9977	0,9990
DeepSeek (ENG)	0,9974	0,9971	0,9997	0,9984	0,9993
GigaChat (ENG)	0,9965	0,9961	0,9995	0,9978	0,9991
GigaChat (RUS)	0,9969	0,9965	0,9997	0,9981	0,9991

DeepSeek (ENG) на логистической регрессии достиг Ассигасы 0,9974 против 0,9337 у Benchmark. Прочерки в строке DeepSeek (RUS) означают, что сгенерированная программа не выводила Precision и Recall для класса

атаки в явном виде, ограничившись F1 и ROC-AUC; это не влияет на корректность решения, но затрудняет прямое сравнение по всем столбцам.

¹¹ <https://github.com/kirillvititnev/llm-iot-intrusion-detection>

¹² Mahmoud Said Elsayed, Nhien-An Le-Khac and Anca Jurecut, "InSDN: SDN Intrusion Dataset" IEEE Access Vol.8, pp 165263-165284, Sep 2020, DOI: 10.1109/ACCESS.2020.3022633

¹³ <https://www.kaggle.com/datasets/badcodebuilder/insdn-dataset/data>

Многоклассовая классификация. В таблице 9 приведены F1-меры по каждому типу атаки для многоклассового сценария. DeepSeek в обеих языковых версиях дал корректные предсказания для всех классов, включая редкие (Botnet, Exploitation). GigaChat (ENG) показал крайне низкие F1 для большинства типов атак, а GigaChat (RUS) не смог определить тип атаки ни для одного класса.

У GigaChat (RUS) все значения F1 равны нулю: модель не смогла определить тип атаки ни для одного класса,

тогда как англоязычный промпт для той же модели дал хотя бы частично рабочие результаты (DoS, DDoS, Probe). Это указывает на чувствительность GigaChat к языку промпта. DeepSeek, напротив, одинаково хорошо работал с обеими версиями промпта: расхождения между русскоязычной и англоязычной версиями лежат в пределах третьего знака, за исключением класса Exploitation, где англоязычная версия дала 0,75 против 0,89 у русскоязычной.

Таблица 9. Многоклассовая классификация: F1 по типам атак

Тип атаки	Benchmark	DeepSeek (RUS)	DeepSeek (ENG)	GigaChat (ENG)	GigaChat (RUS)
DoS	0,97	1,00	1,00	0,03	0,00
DDoS	0,98	0,99	0,99	0,13	0,00
Probe	0,92	1,00	1,00	0,10	0,00
brute-force	0,89	0,98	0,98	0,00	0,00
Web_attack	0,85	0,99	0,99	0,00	0,00
Botnet	0,91	0,95	0,95	0,00	0,00
Exploitation	0,79	0,89	0,75	0,00	0,00
benign	0,99	—	—	0,00	0,00

Таким образом, эксперимент показал, что современные LLM способны генерировать полнофункциональное Python-приложение для бинарной и многоклассовой классификации сетевого трафика по одному текстовому промпту. Программы запускались без ручных исправлений, что говорит о достаточной зрелости модели для задач такого класса.

По качеству бинарной классификации результаты DeepSeek оказались не просто сопоставимы с ручной реализацией, но и превосходили ее по ряду метрик. На логистической регрессии DeepSeek (ENG) достиг Ассурасу 0,9974 против 0,9337 у Benchmark, а ROC-AUC составил 0,9993 против 0,9690.

В многоклассовом сценарии DeepSeek также показал корректные предсказания для всех типов атак, включая редкие классы Botnet и Exploitation. F1 для Exploitation составила 0,89 в русскоязычной версии и 0,75 в англоязычной, для Botnet — 0,95 в обеих. Для сравнения, у Benchmark F1 по этим классам равнялась 0,79 и 0,91

соответственно. То есть даже на редких классах, где обычно проявляются проблемы дисбаланса, сгенерированный код не деградировал относительно ручной реализации, а в ряде случаев ее превзошел.

Иная картина наблюдается у GigaChat. В англоязычной конфигурации модель показала крайне низкие F1-меры для большинства типов атак: 0,03 для DoS, 0,13 для DDoS, 0,10 для Probe, а для brute-force, Web_attack, Botnet и Exploitation — 0,00. Формально программа запускалась и выдавала метрики, но фактически модель не решала задачу многоклассовой классификации. В русскоязычной конфигурации результат оказался ещё хуже: F1 равна нулю для всех классов без исключения, то есть модель не смогла определить тип атаки ни для одной записи, что указывает на сильную зависимость GigaChat от языка подсказки.

DeepSeek, напротив, продемонстрировал устойчивость к языку промпта: расхождения между русскоязычной и англоязычной версиями лежат, в большинстве случаев, в

пределах третьего знака. Таким образом, DeepSeek воспринимает семантически эквивалентные промпты как действительно эквивалентные, а не как разные задачи. Это отличает эту модель от GigaChat, у которой смена языка подсказки привела к качественному изменению результата.

Практический вывод состоит в том, что выбор LLM для генерации кода анализа трафика имеет значение не меньше, чем формулировка промпта.

Исходные тексты программ, промпты и презентация доступны в репозитории команды¹⁴.

ЗАКЛЮЧЕНИЕ

Что можно сказать по результатам такого масштабного синтеза моделей машинного (глубокого) обучения с помощью LLM для, в некотором роде, классического направления использования ИИ в кибербезопасности – анализ сетевого трафика?

Во-первых, главный вывод состоит в том, что кодогенерация реально работает. Практически во всех примерах достаточно было одного запроса. Метрики моделей, как минимум, не уступали метрикам “ручных” приложений.

Качество кода LLM зависит от качества промпта. При этом LLM может оптимизировать уже написанный код.

Русский язык поддержан во всех протестированных моделях, а не только в GigaChat и YandexGPT (Alice AI). Локализация, в принципе, осмысленная – при подсказке на русском языке в коде будут комментарии также на русском языке.

Качественно, различие в результатах генерации есть, хотя для данных примеров они и были небольшие. GigaChat и YandexGPT уступали в некоторых примерах, хотя отставание и нельзя назвать критическим. Какие-то явные аномалии нужно исследовать отдельно. Например, когда результат построенной с помощью GigaChat и английского промпта модели превосходил русскоязычный вариант промпта, или (в одном примере) просто не компилировался. В целом, большая часть вопросов (проблем) относилась именно к русскоязычным промптам для GigaChat.

Что было самым проблемным?

Во-первых, явно присутствовали утечки данных. И здесь разные модели, без явных указаний в промпте про утечки, вели себя по-разному. Какие-то (очевидно, по названию), решали, например, что атрибут IP (IP адрес) использовать нельзя, какие-то модели проблему утечки данных игнорировали.

Во-вторых, разные модели по-разному решали проблемы разбалансировки данных. И это было не всегда успешно.

Можно сказать по итогу, что место для программиста (ML инженера) еще есть – это архитектурные решения и оценка результатов.

Практически все эксперименты, по естественным причинам, велись с бесплатными LLM, за исключением одной команды, которая использовала коммерческие

версии ChatGPT и Claude. Можно отметить, что в данном, сравнительно небольшом испытании, они действительно отличались в лучшую сторону от бесплатных вариантов.

Проведенный эксперимент является продолжением серии практических заданий в рамках курсов по Интернету вещей и инфраструктурным технологиям. Предыдущее задание состояло в исследовании объяснимости моделей машинного обучения для классификации трафика и формированию состязательных атак на этой основе [9].

БИБЛИОГРАФИЯ

- [1] Намиот, Д. Е., and В. А. Сухомлин. "О кибербезопасности систем интернета вещей." *International Journal of Open Information Technologies* 11.2 (2023): 85-97.
- [2] Kalwar, Jawad Hussain, and Sania Bhatti. "Deep learning approaches for network traffic classification in the internet of things (iot): A survey." *arXiv preprint arXiv:2402.00920* (2024).
- [3] Almukhalifi, Hanan, Ayman Noor, and Talal H. Noor. "Traffic management approaches using machine learning and deep learning techniques: A survey." *Engineering Applications of Artificial Intelligence* 133 (2024): 108147.
- [4] Намиот, Д. Е. "О кибератаках с помощью систем Искусственного интеллекта." *International journal of open information technologies* 12.9 (2024): 132-141.
- [5] Erskine, Samuel Kofi. "Real-time large-scale intrusion detection and prevention system (IDPS) CICIoT dataset traffic assessment based on deep learning." *Applied System Innovation* 8.2 (2025): 52.
- [6] Qing, Yi, Xiangyu Liu, and Yanhui Du. "Mitigating data imbalance to improve the generalizability in IoT DDoS detection tasks." *The Journal of Supercomputing* 80.7 (2024): 9935-9960.
- [7] Bouke, Mohamed Aly, Hayate El Atigh, and Azizol Abdullah. "Towards robust and efficient intrusion detection in IoMT: A deep learning approach addressing data leakage and enhancing model generalizability." *Multimedia Tools and Applications* 84.30 (2025): 36523-36542.
- [8] Намиот, Д. Е. О киберрисках генеративного Искусственного Интеллекта / Д. Е. Намиот, Е. А. Ильющин // *International Journal of Open Information Technologies*. – 2024. – Т. 12, № 10. – С. 109-119. – EDN JZCUQS.
- [9] Егоров, М. Э., et al. "Объяснения моделей машинного обучения и состязательные атаки." *International Journal of Open Information Technologies* 13.9 (2025): 50-59.

Статья получена: 30.09.2026

М.А. Аграновский – МГУ имени М.В. Ломоносова, email: agrinscience@gmail.com

М.Г. Аталаева – МГУ имени М.В. Ломоносова, email: maryam.atalaeva@mail.ru

М.В. Варфоломеева – МГУ имени М.В. Ломоносова, email: masha.burdugova@mail.ru

К.Г. Вититнев – МГУ имени М.В. Ломоносова, email: vititnevkirill@gmail.com

В.В. Горбенко – МГУ имени М.В. Ломоносова, email: vvgorbenkovv@gmail.com

А.Д. Горошко – МГУ имени М.В. Ломоносова, email: nikto8748@gmail.com

М.А. Дыскина – МГУ имени М.В. Ломоносова, email: mariadyskina@gmail.com

Е.И. Лаврова – МГУ имени М.В. Ломоносова, email: katlin1702@mail.ru

М.П. Майстер – МГУ имени М.В. Ломоносова, email: mayster-m@bk.ru

Д.К. Мурадян – МГУ имени М.В. Ломоносова, email: itzkhaotic1@gmail.com

Д.Е. Миронова – МГУ имени М.В. Ломоносова, email: dashenka.mironova46@mail.ru

М.И. Опрышко – МГУ имени М.В. Ломоносова, email: mariopryshko@gmail.com

Е.М. Орлова – МГУ имени М.В. Ломоносова, email: vorobushek.ok@gmail.com

¹⁴ https://github.com/u-khasanova/Security_of_infrastructure_technologies_Khasanova_Goroshko_2026

А.Г. Пионткевич – МГУ имени М.В. Ломоносова, email: nastyapion1990@gmail.com	Д.С. Федоров – МГУ имени М.В. Ломоносова, email: fedkamedved@gmail.com
Е.С. Письменная – МГУ имени М.В. Ломоносова, email: Pismennayaka02@gmail.com	У.Н. Хасанова – МГУ имени М.В. Ломоносова, email: uma.khasanlany2001@gmail.com
В.А. Рябыкин – МГУ имени М.В. Ломоносова, email: ryabykin.vlad@mail.ru	Д.Е. Намиот – МГУ имени М.В. Ломоносова, email: dnamiot@gmail.com

Vibe coding in cybersecurity

Marat Agranovskiy, Maryam Atalaeva, Maria Varfolomeeva, Kirill Vititnev, Victoria Gorbenko, Artem Goroshko, Mariia Dyskina, Ekaterina Lavrova, Mikhail Maister, Darya Mironova, David Muradyan, Maria Oprishko, Ekaterina Orlova, Anastasiia Piontkovich, Ekaterina Pismennaya, Vladislav Riabykin, Dmitrii Fedorov, Uma Khasanova, Dmitry Namiot

Abstract - This article presents the results of a large-scale experiment in which teams of students from the M.V. Lomonosov Moscow State University's Computational Mathematics and Cybernetics (CM) program, as part of a course on Internet of Things security, developed network traffic classification programs using large-scale language models (LLMs). The use of machine learning (deep learning) models is a classic approach for intrusion detection systems. Available Kaggle datasets were offered as input data. Along with the datasets, hand-coded models were also available on the same website, serving as baseline examples for comparison with automated solutions. The task involved developing prompts, testing them against several LLMs, and, separately, testing the English and Russian language versions of the prompts. Overall, all experiments conducted produced working LLM models that performed as well as existing hand-coded models in terms of metrics. However, the obtained results required verification, just as the architectural solutions chosen by the LLM were not always optimal and sometimes simply incorrect, for example, in terms of data leakage. This still leaves room for programmers (ML engineers) to work.

Keywords— artificial intelligence, LLM, programming automation.

REFERENCES

- [1] Namiot, D. E., and V. A. Suhomlin. "O kiberbezopasnosti sistem interneta veshhej." International Journal of Open Information Technologies 11.2 (2023): 85-97.
- [2] Kalwar, Jawad Hussain, and Sania Bhatti. "Deep learning approaches for network traffic classification in the internet of things (iot): A survey." arXiv preprint arXiv:2402.00920 (2024).
- [3] Almukhalfi, Hanan, Ayman Noor, and Talal H. Noor. "Traffic management approaches using machine learning and deep learning techniques: A survey." Engineering Applications of Artificial Intelligence 133 (2024): 108147.
- [4] Namiot, D. E. "O kiberatakah s pomoshh'ju sistem Iskusstvennogo intellekta." International journal of open information technologies 12.9 (2024): 132-141.
- [5] Erskine, Samuel Kofi. "Real-time large-scale intrusion detection and prevention system (IDPS) CICIoT dataset traffic assessment based on deep learning." Applied System Innovation 8.2 (2025): 52.
- [6] Qing, Yi, Xiangyu Liu, and Yanhui Du. "Mitigating data imbalance to improve the generalizability in IoT DDoS detection tasks." The Journal of Supercomputing 80.7 (2024): 9935-9960.
- [7] Bouke, Mohamed Aly, Hayate El Atigh, and Azizol Abdullah. "Towards robust and efficient intrusion detection in IoMT: A deep learning approach addressing data leakage and enhancing model generalizability." Multimedia Tools and Applications 84.30 (2025): 36523-36542.
- [8] Namiot, D. E. O kiberriskah generativnogo Iskusstvennogo Intellekta / D. E. Namiot, E. A. Il'jushin // International Journal of Open Information Technologies. – 2024. – T. 12, # 10. – S. 109-119. – EDN JZCUQS.
- [9] Egorov, M. Je., et al. "Ob"jasnenija modelej mashinnogo obuchenija i sostjazatel'nye ataki." International Journal of Open Information Technologies 13.9 (2025): 50-59.