УДК 004.056
DOI: 10.26102/2310-6018/2026.56.5.008

Распознавание начала функций в бинарных файлах с использованием рекуррентных нейронных сетей

А.С. Шайханов✉

Московский государственный технический университет имени Н.Э. Баумана, Москва, Российская Федерация

Резюме. В статье рассматривается задача распознавания начал функций в бинарных файлах, которая является одной из ключевых подзадач реверс-инжиниринга программного обеспечения. Актуальность исследования обусловлена ограничениями традиционных детерминированных методов, основанных на эвристиках, сигнатурном анализе и анализе графов потока управления, а также недостаточной универсальностью существующих нейросетевых решений, ориентированных преимущественно на архитектуры x86 и x86–64. Целью работы является разработка и экспериментальная оценка модели машинного обучения, способной эффективно распознавать начала функций в бинарных файлах, собранных под альтернативные машинные архитектуры, с учетом прикладной специфики задач обратной разработки. В качестве базового подхода предложено использование рекуррентной нейронной сети, обрабатывающей последовательности байтов бинарного файла. Проведен сравнительный анализ существующих нейросетевых моделей распознавания функций, выявлены их преимущества и ограничения, что позволило обосновать выбор простой и воспроизводимой архитектуры RNN. В рамках исследования детально изучено влияние ключевых гиперпараметров модели, включая длину входной последовательности, количество нейронов в рекуррентном слое и веса функции потерь, на качество распознавания. Эксперименты выполнены на бинарных файлах микроконтроллеров ESP32 архитектуры Xtensa Little Endian и STM32WBA6 с ядром Cortex-M33 архитектуры ARMv8-M с использованием как стандартного, так и случайного выравниваний, что позволило оценить устойчивость модели к изменению структуры бинарных данных. Результаты показывают, что длина входной последовательности является наиболее значимым гиперпараметром, в то время как влияние весов функции потерь носит вторичный характер. Установлено, что модель не обладает обобщаемостью между различными типами выравниваний, что требует предварительного анализа бинарного файла перед применением. На основе разработанной модели реализовано расширение для дизассемблера IDA Pro, демонстрирующее практическую применимость предложенного подхода в реальных задачах реверс-инжиниринга.

Ключевые слова: реверс-инжиниринг, распознавание функций, бинарный файл, начало функции, рекуррентная нейронная сеть, IDA Pro.

Для цитирования: Шайханов А.С. Распознавание начала функций в бинарных файлах с использованием рекуррентных нейронных сетей. *Моделирование, оптимизация и информационные технологии.* 2026;14(5). URL: https://moitvivt.ru/ru/journal/article?id=2273 DOI: 10.26102/2310-6018/2026.56.5.008

Recognizing function prologues in binary files with recurrent neural networks

A.S. Shaykhanov✉

Bauman Moscow State Technical University, Moscow, the Russian Federation

Abstract. The article discusses the problem of recognising function beginnings in binary files, which is one of the key subtasks of software reverse engineering. The relevance of the research is due to the limitations of traditional deterministic methods based on heuristics, signature analysis, and control flow

graph analysis, as well as the insufficient versatility of existing neural network solutions, which are primarily focused on x86 and x86-64 architectures. The aim of the work is to develop and experimentally evaluate a machine learning model capable of effectively recognising function starts in binary files compiled for alternative machine architectures, taking into account the applied specifics of reverse engineering tasks. The basic approach proposed is to use a recurrent neural network that processes byte sequences of a binary file. A comparative analysis of existing neural network models for function recognition was conducted, and their advantages and limitations were identified, which made it possible to justify the choice of a simple and reproducible RNN architecture. The study examined in detail the impact of key model hyperparameters, including the length of the input sequence, the number of neurons in the recurrent layer, and the weights of the loss function, on the quality of recognition. The experiments were performed on binary files of the ESP32 microcontroller with Xtensa Little Endian architecture and STM32WBA6 microcontroller of Cortex-M33 core with ARMv8-M architecture using both standard and random alignment, which made it possible to evaluate the model's resistance to changes in the structure of binary data. The results show that the length of the input sequence is the most significant hyperparameter, while the influence of the loss function weights is secondary. It has been established that the model does not generalise between different types of alignments, which requires preliminary analysis of the binary file before application. Based on the developed model, an extension for the IDA Pro disassembler has been implemented, demonstrating the practical applicability of the proposed approach in real reverse engineering tasks.

Keywords: reverse engineering, function recognition, binary file, function prologue, recurrent neural network, IDA Pro.

For citation: Shaykhanov A.S. Recognizing function prologues in binary files with recurrent neural networks. *Modeling, Optimization and Information Technology.* 2026;14(5). (In Russ.). URL: https://moitvivt.ru/ru/journal/article?id=2273 DOI: 10.26102/2310-6018/2026.56.5.008

Введение

Реверс-инжиниринг программного обеспечения – процесс исследования файлов этого программного обеспечения с целью восстановления принципов работы для выполнения различных задач, например, поиска недекларированных возможностей. Реверс-инжиниринг почти всегда включает в себя анализ исполняемого кода внутри бинарных файлов, представленного в виде машинных инструкций. Одной из подзадач анализа кода является распознавание начал функций внутри бинарного файла.

На сегодняшний день реверс инженеры используют различные инструменты, автоматизирующие процесс распознавания начал функций. Этими инструментами могут быть простые скрипты, написанные исследователем самостоятельно, бесплатные утилиты, разработанные сообществом, или специализированное коммерческое программное обеспечение, решающее сразу несколько задач обратной разработки помимо распознавания функций. Однако большая часть этих инструментов не применяет нейронные сети, а использует детерминированные методы, основанные на эвристиках, сигнатурном анализе, построении графа потока управления. В то же время существующие инструменты, основанные на нейронных сетях, имеют множество недостатков, делающими очень сложным постоянное использование этих инструментов для решения широкого круга задач.

Цель данного исследования – частичная автоматизация решения задачи распознавания начал функций внутри бинарного файла посредством разработки прикладного инструмента реверс-инжиниринга. Данный инструмент основан на модели нейронной сети. Важной особенностью исследования также является то, что при разработке этого инструмента обязательно должна учитываться прикладная специфика задач обратной разработки.

Обзор существующих исследований. На данный момент существуют научные исследования, решающие задачи реверс-инжиниринга посредством машинного обучения. К таким задачам можно отнести:

– Выделение в бинарном файле секций кода и данных [1, 2];

– Формирование корректного дизассемблерного представления бинарного кода [3];

– Восстановление имен функций в бинарный файлах, не содержащих отладочную информацию [4, 5];

– Распознавание границ функций внутри бинарного файла [6, 7];

– Восстановление типов данных и аргументов функций внутри бинарного файла [8];

– Восстановление имен переменных в декомпилированном представлении функций бинарного файла [9];

– Поиск функций, семантических схожих между собой, в нескольких бинарных файлах [10, 11];

– Сравнение разных версий бинарного файла (бинарный диффинг) [12];

– Создание семантических векторных представлений инструкций ассемблера [13];

– Создание больших наборов с собранными бинарными файлами для обучения моделей [14, 15].

Предложенные решения демонстрируют хорошие показатели оценивающих метрик в сравнении с традиционными методами реверс-инжиниринга. Конкретно для задачи распознавания начал функций этими традиционными методами являются различные эвристики, сигнатурный анализ, построение графа потока управления и другие. Часть этих методов реализована в популярных инструментах реверс-инжиниринга, так как IDA Pro и Ghidra [16, 17].

Далее описаны существующие модели нейронных сетей, распознающих начала функций внутри бинарных файлов, а также выделены преимущества и недостатки этих моделей.

RNN. В 2015 году была представлена модель, использующую двунаправленную рекуррентную нейронную сеть [18]. На вход модели подавались последовательности фиксированной длины в 1000 байт. Байты последовательность конвертировались в one-hot вектора, после чего передавались на вход рекуррентному слою. Модель имела один двунаправленный рекуррентный слой из 16 нейронов. Выход рекуррентного слоя конкатенировался для прямого и обратного направлений и подавался в softmax для создания распределения вероятностей соответствия байта началу (концу) функции.

Авторы модели утверждают, что они использовали две независимые одинаковые модели для решения двух независимых задач по поиску начала и конца функции. Далее результат работы моделей объединялся посредством простой эвристики для получения точных границ функции. Многие авторы моделей, представленных после 2015 года, использовали показатели метрик качества работы RNN для сравнения.

Преимущества модели:

– Высокая скорость обучения модели, обусловленная простотой работы модели;

– Высокая скорость работы модели согласно оценкам авторов статьи «Padding Matters»;

– Высокие показатели метрик для Executable and Linkable Format (ELF) и Portable Executable (PE) форматов файлов, собранных под архитектуры x86 и x86–64 с разными уровнями оптимизации;

– Простота реализации предполагает возможность воспроизвести архитектуру модели, а далее обучить модель для нужной машинной архитектуры.

Недостатки модели:

– Отсутствие программной реализации модели от авторов статьи в открытом доступе;

– Значительное снижение показателей качества модели для x86 и x86-64 при заполнении выравниваний случайными байтами.

Именно эта модель была взята автором данной работы за основу для решения задачи распознавания начала функций.

XDA. В 2020 году была представлена модель, основанная на переносе различных контекстных зависимостей машинного кода для решения задач обратного проектирования, в том числе и для задачи распознавания начала функций [19].

Работа модели делится на две части:

– Предобучение модели посредством маскированного языкового моделирования (MLM) для выявления контекстных зависимостей машинного кода. Задача MLM – обучить модель предсказывать случайно замаскированные байты исходя из окружающего контекста. По словам авторов XDA, это не только повышает уровень понимания моделью машинного кода, но и повышает устойчивость модели к различным компиляторам и уровням оптимизации;

– Обучение модели для распознавания функций. Авторы использовали слои внутреннего внимания для вычисления потока информации между каждой парой байтов. Это необходимо для установления зависимостей между удаленными байтами тела функции, что повышает качество восстановления границ функций.

Преимущества модели:

– Наличие исходного кода модели и уже предобученных весов для некоторых архитектур в открытом доступе, что дает возможность настроить или предварительно обучить модель для нужной машинной архитектуры;

– Возможность использования предобученной модели для решения других задач обратной разработки;

– Высокие показатели метрик для ELF и PE форматов файлов, собранных под архитектуры x86 и x86–64 с разными уровнями оптимизации;

Недостатки модели:

– Низкая скорость обучения из-за сложной архитектуры;

– Низкая скорость работы согласно оценкам авторов DeepDi и Padding Matters;

– Значительное снижение показателей качества модели для x86 и x86–64 при заполнении выравниваний случайными байтами.

DeepDi. В 2022 году была представлена модель, работающая не с байтами, а с машинными инструкциями. Для первичной разметки инструкций в DeepDi строится супермножество всевозможных инструкций. Далее на основе этого супермножества строится граф потока инструкций. Графовая сверточная сеть анализирует этот граф для определения реальных инструкций и их связей между собой [20].

Поиска начала функций DeepDi использует:

– Набор эвристик, применяемых к размеченным инструкциям для определения кандидатов на начало функции. По словам автором, данный подход не только снижает количество ложных срабатываний, но и существенно сокращает входное количество байтов, подаваемое на вход нейронной сети;

– Нейронную сеть со слоем встраивания, рекуррентным слоем и двухслойным персептроном с классификатором.

Преимущества модели:

– Наличие модели, собранной в динамическую библиотеку с программным интерфейсом для доступа к функциям восстановления границ;

– Самая высокая скорость работы среди всех моделей согласно оценкам авторов статьи «Padding Matters»;

– Самые высокие показатели метрик для ELF и PE форматов файлов, собранных под архитектуры x86 и x86–64 с разными уровнями оптимизации согласно оценкам авторов статьи «Padding Matters»;

– Наименьшее снижение показателей качества модели для x86 и x86–64 при заполнении выравниваний случайными байтами согласно оценкам авторов статьи «Padding Matters».

Недостатки модели:

– Отсутствие программной реализации модели. Это обусловлено тем, что разработанный DeepDi принадлежит коммерческой компании DeepBits;

– Ограниченный набор архитектур: x86 и x86–64.

Выводы о существующих моделях. RNN, XDA, DeepDi были обучены на больших наборах обучающих данных. По словам авторов моделей, каждая из них в свое время демонстрировала хорошие показатели оценивающих метрик. Более того, все они в каком-то виде имеют прикладные реализации, которые можно использовать для решения собственных задач.

Еще одной важной отличительной особенностью предыдущих исследований является то, что задачу распознавания функций исследователи рассматривали как «дополнительную». То есть авторами работ были предприняты попытки создать универсальный инструмент бинарного анализа, который мог бы не только восстановить границы функций, но и разметить ассемблерные инструкции, классифицировать вредоносный бинарный файл. Вероятно, это также стало причиной усложнения архитектур самих моделей с течением времени.

Тем не менее, все эти модели имеют недостатки. Основной недостаток – обучение и оценка на бинарных файлах, собранных под ограниченный список архитектур: x86 и x86–64. Вероятно, это вызвано наличием уже готовых емких наборов данных для обучения. Эти наборы включают в себя PE и ELF файлы, собранные под Windows и Linux разными компиляторами и с разными уровнями оптимизациями машинного кода.

Таким образом, задачу распознавания функций необходимо рассмотреть для других машинных архитектур и инструментов сборки. При построении модели помимо ее метрик качества необходимо также учитывать особенности прикладных задач по распознаванию функций. В данном случае, имеется в виду ограниченное время исследователя, применяющего модель, и малый набор обучающих данных.

Материалы и методы

Постановка задачи и оценка качества решения. Входные данные B^F представляют собой последовательность байтов B_0, B_1 … B_{F-1} некоторого бинарного файла размера F, где $B_i \in Z_{256}$ – байт бинарного файла со смещением i.

Выходные данные O^F представляют собой последовательность признаков начала функции O_0, O_1 … $O_{F-1,}$ где $O_i \in Z_2$. При этом $O_i = 1 \Rightarrow$ байт B_i был распознан как байт начала функции, в противном случае байт B_i не обладает признаком начала функции. Количество ненулевых элементов в $\{O_0, O_1 \dots O_{F-1}\}$ – общее количество распознанных функций внутри бинарного файла.

Таким образом, рассматриваемая задача распознавания начал функций представляет собой задачу бинарной классификации. Традиционными метриками оценки качества бинарной классификации являются: Accuracy, Precision, Recall и F1 метрика. Формулы метрик приведены ниже.

$$Accuracy = \frac{\mathrm{TP+TN}}{\mathrm{TP+TN+FP+FN}}, \quad (1)$$

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP+FP}}, \quad (2)$$

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP+FN}}, \quad (3)$$

$$F1 = \frac{2\times\mathrm{Precision}\times\mathrm{Recall}}{\mathrm{Precision+Recall}}. \quad (4)$$

В формулах используются следующие обозначения:

– TP (True Positive) – количество случаев, когда модель верно распознала байт начала функции;

– TN (True Negative) – количество случаев, когда модель верно распознала байт, не являющийся байтом начала функции;

– FP (False Positive) – ошибки 1 рода. Количество случаев, когда модель распознала байт, не являющийся байтом начала функции, как байт начала функции;

– FN (False Negative) – ошибки 2 рода. Количество случаев, когда модель распознала байт начала функции, как байт, не являющийся началом функции.

При выборе метрики оценки качества учитывалось то, что любой бинарный файл помимо секций с кодом функций будет содержать в себе секции с данными: строки, таблицы с адресами и так далее. Также очевидно то, что в коде исполняемых функций всегда будет гораздо меньше байт, обладающих признаком «начала функции», чем байт, не являющихся началом функции. Все это в совокупности вносит сильный дисбаланс в обучающую и тестирующую выборки. Данная проблема отражается в следующем:

Метрика Accuracy, представляющая собой отношение количества верно предсказанных ответов к общему количеству предсказанных ответов, не может быть использована для оценки качества работы модели. Тривиальный классификатор, определяющий любой байт последовательности как байт, не обладающий признаком начала функции, будет демонстрировать высокие показатели данной метрики, что не является правильным;

Недопустимо как уменьшение числа ошибок 1 рода за счет существенного увеличения ошибок 2 рода, так и обратное. Для универсальности оценки качества работы модели необходимо учитывать оба вида ошибок, поэтому отдельно взятый Precision (Recall) не может использоваться в качестве универсальной метрики.

Исходя из вышеперечисленного, для оценки качества работы модели используется f1 метрика.

В предыдущих работах исследователи ставили перед собой разные задачи, смежные с восстановлением начала функций:

– Восстановление конца функции, а далее объединение начала и конца для восстановления границ всей функции с помощью каких-то дополнительных эвристик;

– Восстановление инструкций ассемблера.

Перечисленные задачи не будут рассматриваться в данной работе из-за того, что встроенный функционал IDA Pro способен дизассемблировать инструкции и распознавать тело функции посредством анализа потока управления, если правильно предоставить ему начало функции.

Набор данных для обучения и оценки. Исходный набор данных состоял из двух поднаборов – последовательностей байтов файлов программ микроконтроллера ESP32 архитектуры Xtensa Little Endian и последовательностей байтов файлов программ микроконтроллера STM32WBA6 с ядром Cortex-M33 архитектуры ARMv8-M. Каждый

из поднаборов использовался для обучения и тестирования отдельной модели, что позволило получить независимые результаты для двух разных машинных архитектур.

Исходный код файлов программ микроконтроллера ESP32 разрабатывался на языке Rust с использованием C/C++ библиотек. Сборка файла проекта осуществлялась с использованием фреймворка сборки Rust проектов от Espressif Systems.

Исходный код файлов программ микроконтроллера STM32WBA6 разрабатывался на языках C/C++. Сборка файла проекта осуществлялась с использованием фреймворка сборки проектов от STMicroelectronics.

В предыдущих исследованиях авторы ограничили обучающую и тестирующую выборки. Они использовали секции с кодом, мотивируя это необходимостью минимизации дисбаланса классов. В данном исследовании, как было указано ранее, для обучения и тестирования используются все секции бинарного файла, включающие в себя и данные, и код, и служебные заголовки. Это было сделано исходя из следующей гипотезы: «Обучающая и тестирующая выборки должны максимально соответствовать реальным данным, на которых впоследствии будет применяться модель».

Все секции исходного бинарного файла считываются и разбиваются на последовательности фиксированной длины. Длина последовательности является гиперпараметром. Далее последовательности случайно перемешиваются между собой с фиксированным параметром рандомизации для воспроизводимости, после чего разбиваются на обучающую и тестирующую выборки. Для разбиения используется стандартное отношение 80 % к 20 %.

Необходимо отметить, что идея применения моделей нейронных сетей XDA и RNN для распознавания начал функций в Rust бинарных файлах уже была реализована в работе «RustBound: Function Boundary Detection over Rust Stripped Binaries». Авторы продемонстрировали, что от конфигураций сборки Rust проектов сильно зависят байты прологов и эпилогов функций в собираемом бинарном файле. Поэтому для формирования обучающего набора данных авторы собирали бинарные файлы с разными уровнями оптимизации. Данная работа имеет недостаток оригинальных работ XDA и RNN – обучение модели для архитектуры x86–64. При этом в своей работе авторы ограничились экспериментами с разными подходами к обучению моделей, не проделав никаких экспериментов по изменению гиперпараметров архитектур оригинальных моделей [21].

Выбор модели нейронной сети. Поскольку авторы RNN модели не предоставили ее исходный код, то реализация модели осталась за авторами данной статьи. Исходный код написан на python3 с использованием библиотеки для машинного обучения tensorflow.

За основу архитектуры нейронной сети была взята RNN модель. Основные причины:

– Универсальность модели: минимальное количество дополнительных эвристик, зависящих от конкретной машинной архитектуры. Модель принимает на вход последовательности байт, а не инструкции, напрямую зависящие от конкретной архитектуры.

– Модели со сложной архитектурой требуют большего набора данных для обучения. Простые модели, наоборот, не требовательны к большим наборам обучающих данных. В связи с ограничениями количества входных данных, связанными с прикладными особенностями решаемой задачи, было принято решение рассматривать простые архитектуры нейронных сетей.

– Высокая скорость обучения, обусловленная простотой архитектуры модели. Высокая скорость обучения позволяет проверить большее количество гипотез с

изменением гиперпараметров модели для поиска их оптимальных значений. Скорость обучения также важна в случае, если независимый исследователь захочет переобучить модель со своими значениями гиперпараметров.

– Высокая скорость работы модели. Анализ байтов исследуемого бинарного файла посредством разработанного расширения для IDA Pro должен быстро выполняться, поскольку время исследователя на анализ одного бинарного файла может быть сильно ограничено. В статье «Padding Matters» авторы измерили время работы моделей RNN, XDA и DeepDi, а также инструментов реверс-инжиниринга Ghidra и IDA Pro на файлах динамической библиотеки Chromium и файлах вредоносной программы-вымогателя Conti. Результаты измерений показали, что время работы DeepDi составило секунды, время работы RNN – минуты, а XDA, Ghidra и IDA Pro – часы. Поскольку использование модели DeepDi невозможно на бинарных файлах, собранных не под архитектуру x86, то RNN остается быстрейшим вариантом для распознавания начала функций из доступных.

Несмотря на все достоинства модели RNN, она нуждается в некоторых изменениях перед исследованием гиперпараметров.

В работе «Explaining deep learning based security applications» авторы предлагают модификации модели RNN от 2015 года. Основное изменение – уменьшение количества нейронов выходного слоя и изменение правила выбора класса [22]:

Основные причины:

– В оригинальной версии модели выходной слой имел 2 нейрона, которые выдавали вероятности того, что байт является или не является началом функции. Функция выбора максимального значения из двух вероятностей определяла класс байта;

– В адаптированной версии модели выходной слой имеет всего 1 нейрон, выдающий одну вероятность. Далее эта вероятность сравнивается с пороговым значением для определения класса.

Реализация с одним выходным нейроном имеет следующие преимущества:

– Задача бинарной классификации требует всего одну вероятность. Принадлежность байта к классам 1 и 0 – противоположные события;

– Адаптивное пороговое значение позволяет балансировать метрики качества, по сути, представляя собой еще один гиперпараметр.

Пороговое значение принимается равным 0,5. Авторы работы утверждают, что такие простые изменения позволили им повысить значения показателей метрик качества.

Формальное описание модели. В начале раздела было описано, что входными данными является последовательность байт B^F бинарного файла размера F, где $B_i \in Z_{256}$. Эта последовательность разбивается на подпоследовательности длины L, а далее каждый байт B_j подпоследовательности посредством one-hot преобразования приводится к вектору R^{256} действительных чисел, в котором элемент с индексом, равным значению байта B_j, является единичным, а все остальные – нулевыми:

$$One_{hot}: Z_{256}^{L} \to R^{L\times 256}. \quad (5)$$

One-hot преобразование необходимо, поскольку в наборах инструкций микроконтроллеров байты с близкими значениями могут обладать совершенно различными семантическими свойствами.

После one-hot преобразования подпоследовательности передаются в двунаправленный рекуррентный слой размерности N. Под двунаправленностью в данном случае понимается использование двух независимых рекуррентных слоев, один из которых получает one-hot вектора в прямом порядке, в второй – в обратном. Выходы

двух этих слоев объединяются для передачи в следующий слой. Формально рекуррентный слой можно определить так:

$$Recurrent\_layer: R^{L\times 256} \cdot R^{N} \to R^{L\times 2N}. \quad (6)$$

При этом для каждого j-го входного вектора $one\_hot_j \in R^{256}$, $j\in[0, L-1]$, подаваемого на вход рекуррентному слою прямого порядка справедливо соотношение:

$$Forward\_layer\left(one_hot_j\right) = state_{j-1} =$$

$$= Relu(W_{state,one\_hot}\ one\_hot_j + W_{state,state}\ state_{j-1} + W_{state}), \quad (7)$$

где $state_{j-1}$, $state_{j-1} \in R^N$ векторы внутреннего состояния рекуррентного слоя для j−1 и j входных one-hot векторов соответственно; $W_{state,one\_hot} \in R^{Nx256}$, $W_{state,state} \in R^{NxN}$, $W_{state} \in R^N$ матрицы весов рекуррентного слоя; $W_{state,one\_hot}\cdot one\_hot_j$ – вклад текущего байта подпоследовательности; $W_{state,state}\cdot state_{j-1}$ – вклад предыдущих байтов подпоследовательности (вклад внутреннего состояния рекуррентного слоя); Relu – функция активации в рекуррентном слое.

Приведенное выше соотношение (7) аналогично для слоя обратного порядка с поправкой на очередность подаваемых векторов.

Выход с рекуррентного слоя передается в слой активации, а после полученный вектор вероятностей обрабатывается решающим правилом для получения подпоследовательности признаков начала функции:

$$Activation\_layer: R^{L\times 2N} \to Z_2^L. \quad (8)$$

– Sigmoid – используемая функция активации в слое;

– Наличие признака начала функции для i-го байта подпоследовательности определяется из сравнения полученной вероятности для i-го байта с пороговым значением 0,5.

Подпоследовательности признаков объединяются в последовательность O^F, являющуюся выходом решения задачи распознавания функции в бинарных файлах.

Исследуемые параметры. В представленной работе исследуются гиперпараметры модели, которые могут зависеть либо от архитектуры, под которую собран бинарный файл, либо от особенностей и структуры самого бинарного файла. Список параметров:

– Длина входной последовательности байт. Исходный файл размером в 10000 байт можно разделить большим количеством способов. Например, на 10 последовательностей длиной 1000 или 100 последовательностей с длиной в 100 байт. В исследовании от 2015 года этому не уделяется необходимое внимание. Авторы принимают длину последовательности равной 1000 байт и, в отличие от количества нейронов в рекуррентном слое, не приводят никакой информации об исследованиях. Оптимальное значение параметра длины входной последовательности, скорее всего, зависит от длин самих функций, их прологов, эпилогов и выравниваний.

– Количество нейронов в рекуррентном слое. Исходная модель содержит в рекуррентном слое 16 байт. Несмотря на то, что между количеством нейронов и памятью рекуррентного слоя нет явной зависимости, количество нейронов неявным образом влияет на способность слоя запоминать и обрабатывать входные последовательности.

– Веса функции потерь. В данном разделе уже упоминалась проблема дисбаланса классов при описании показателей метрик качества. Отношение количества байт начала функции к количеству байт не начала функции, возможно, должно быть связано с оптимальным значением весов классов функции потерь.

Все перечисленные выше параметры необходимо также исследовать для случайных байтов выравнивания, как это было предложено в работе «Padding Matters» [23].

Результаты

Далее в тексте представлены графики метрик качества для исследуемых параметров модели. В каждом эксперименте изменялся только один параметр, если о другом не написано явно. По умолчанию, значения параметров устанавливались либо рекомендованными значениями от авторов RNN, либо некоторыми значениями, эмпирически подобранными авторами данной работы. Параметры эмпирически подбирались, если авторы модели RNN явно их не описывали. Значения гиперпараметров модели:

– Длина one-hot вектора – 256. Единица в ячейке с номером, равным значению байта, остальное – нули.
– Длина входной последовательности – 1000 байт.
– Оптимизатор – rmsprop.
– Значение весов в бинарной кросс-энтропии – 0,9 для 1 класса и 0,1 для 0 класса.
– Количество нейронов в рекуррентном слое – 16.
– Архитектура рекуррентного слоя – SimpleRNN.
– Функция активации в рекуррентном слое – relu.
– Функция активации в нейроне выходного слоя – sigmoid.
– Порог определения принадлежности к классу – 0,5.
– Количество эпох обучения – 30.

Все графики приведены для тестирующей выборки.

Обучение со стандартным выравниванием

Количество нейронов в рекуррентном слое. На Рисунках 1 и 2 изображены графики метрики f1 для класса 1 в зависимости от эпохи обучения для ESP32 и STM32WBA6 соответственно. Изменяемый гиперпараметр модели – количество нейронов в рекуррентном слое.

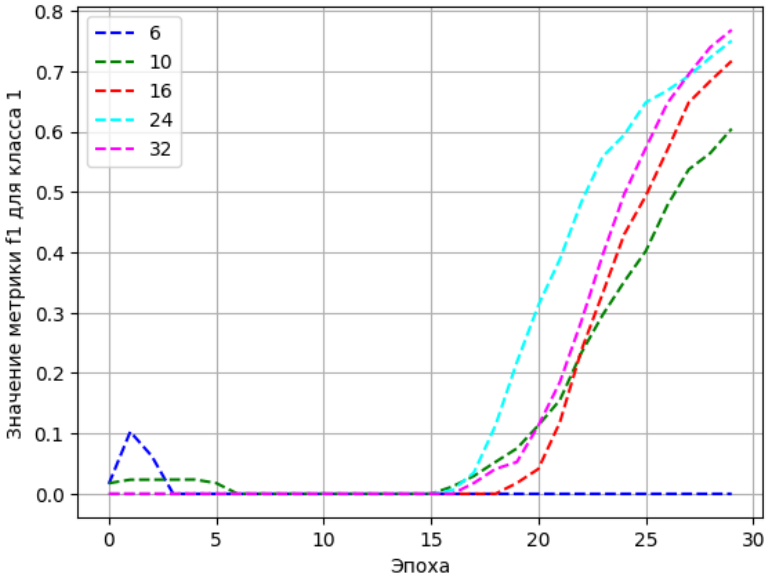

Рисунок 1 – График метрики f1 для ESP32 при изменении количества нейронов в рекуррентном слое

Figure 1 – F1 metric graph for ESP32 as the number of neurons in the recurrent layer changes

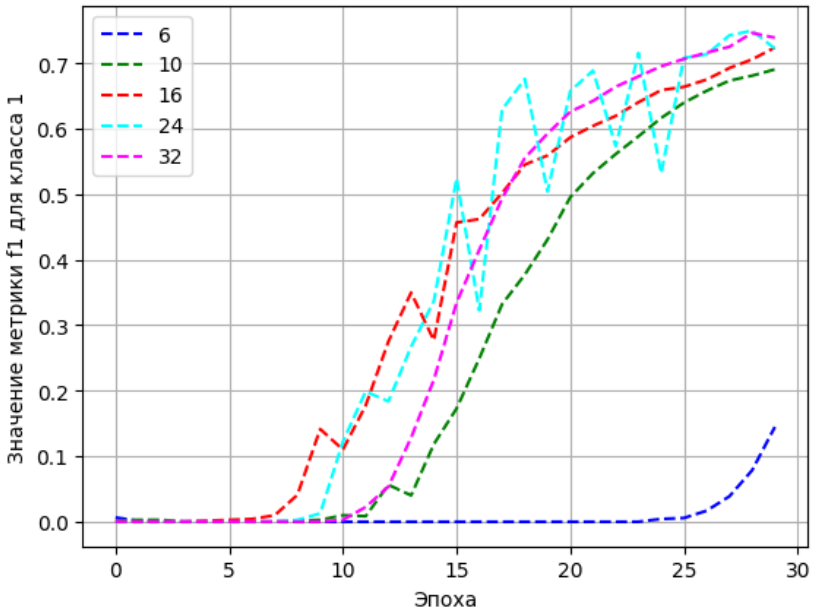

Рисунок 2 – График метрики f1 для STM32WBA6 при изменении количества нейронов в рекуррентном слое

Figure 2 – F1 metric graph for STM32WBA6 as the number of neurons in the recurrent layer changes

Уменьшение количества нейронов существенно ухудшает качество работы модели, что особенно заметно при критическом значении в 6 нейронов в слое.

Увеличение количества нейронов в рекуррентном слое незначительно улучшает качество работы модели. На графике для ESP32 значение метрики f1 для класса 1 с оригинальным значением гиперпараметра в 16 нейронов составило 0,72, а для 32 нейронов это значение 0,77. В то же время на графике для STM32WBA6 увеличение числа нейронов с 16 до 32 позволило повысить значение f1 всего лишь с 0,72 до 0,74. В обоих случаях графики имеют тенденцию к сходимости, что ограничивает возможность в существенном увеличении качества работы модели посредством увеличения числа нейронов.

Исходя из этого, можно сделать вывод о том, что увеличение количества нейронов в рекуррентном слое позволяет улучшить качество работы модели со стандартным выравниванием при фиксированных значениях остальных гиперпараметров, однако у этого улучшения есть верхний предел.

Длина входной последовательности. На Рисунках 3 и 4 изображены графики метрики f1 для класса 1 в зависимости от эпохи обучения для ESP32 и STM32WBA6 соответственно. Изменяемый гиперпараметр модели – длина входной последовательности.

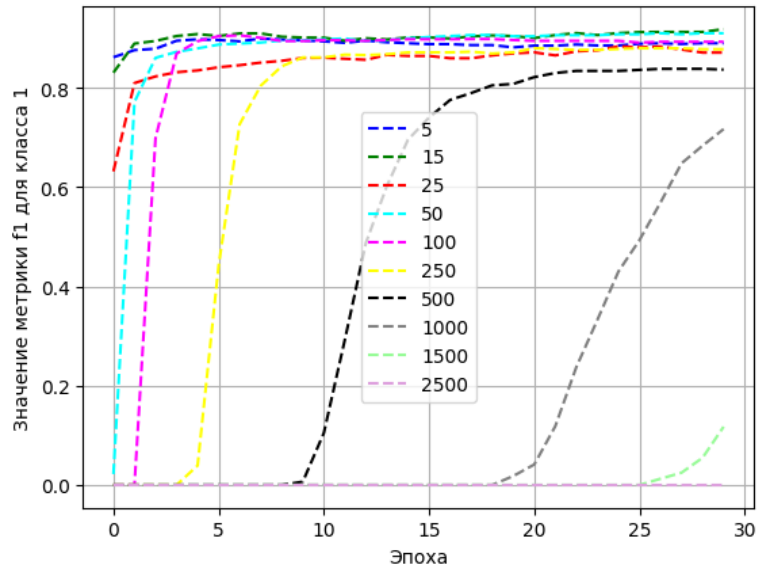

Рисунок 3 – График метрики f1 для ESP32 при изменении длины входной последовательности
Figure 3 – F1 metric graph for the ESP32 as the input sequence length changes

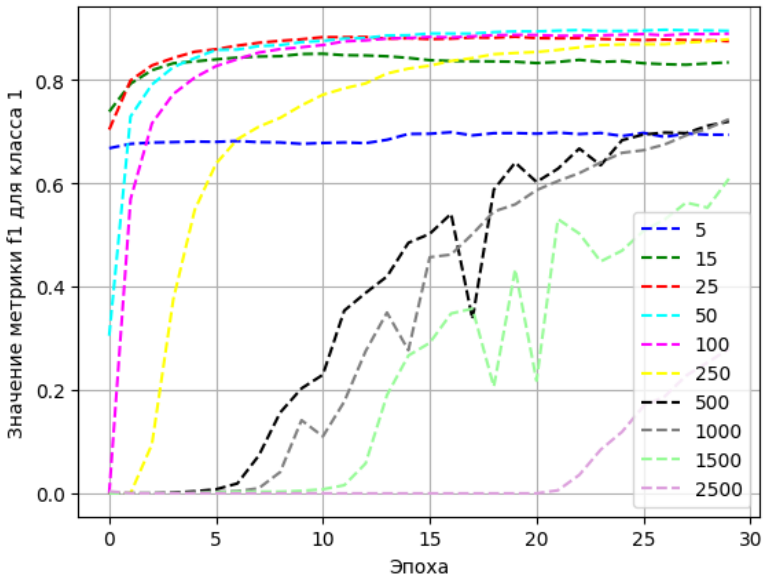

Рисунок 4 – График метрики f1 для STM32WBA6 при изменении длины входной последовательности
Figure 4 – F1 metric graph for STM32WBA6 as the input sequence length changes

Увеличение длин входных последовательностей приводит к избытку контекста у нейронной сети. Для распознавания начала функций методами машинного обучения нет необходимости в анализе полного кода тел функций. Вместо этого необходимо выделять важные признаки в потоке байтов: прологи, эпилоги и выравнивания. Длинные последовательности затрудняют выделение этих признаков. На графиках проблема избытка контекста проявляется в увеличении числа эпох, необходимого для сходимости обучения модели. Метрика f1 у длинных входных последовательностей после 30 эпох значительно ниже, чем у средних последовательностей. Аналогичное справедливо для средних и коротких последовательностей, но по достижении меньшего числа эпох.

С другой стороны, чрезмерное уменьшение длины входной последовательности может приводить к нехватке контекста у нейронной сети. На графике для STM32WBA6

это заметно для длин последовательностей в 15 и особенно 5 байт. На Рисунке 5 отражены типичные эпилог функции X и пролог функции X+1 для STM32WBA6. Если посчитать байты инструкций: эпилог (6–8+ байт), пролог (6–8+ байт), а также возможное выравнивание функций по адресам, кратным четырем (1–3 байта), то получится, что длины последовательности меньше 15 байт может быть недостаточно для распознавания начала функции. Из-за разделения на слишком короткие последовательности контекст может чаще размазываться по нескольким входным последовательностям. В данном случае имеется в виду то, что эпилог функции X с выравниванием может попасть в i-ю последовательность, а пролог функции X+1 попадет в i+1 последовательность. Тогда при определении класса у байта начала функции X+1 нейросеть на входе не будет иметь эпилог и выравнивание.

```
.text:0803195C 83 20                    MOVS    R0, #0x83
.text:0803195E E0 75                    STRB    R0, [R4,#0x17]
.text:08031960 00 20                    MOVS    R0, #0
.text:08031962 B0 BD                    POP     {R4,R5,R7,PC}
.text:08031962                  ; End of function SMP_MI_Send_Pairing_Random
.text:08031962
.text:08031964
.text:08031964                  ; =============== S U B R O U T I N E =======================================
.text:08031964
.text:08031964
.text:08031964                  SMP_LP_Compute_Confirm_Value            ; CODE XREF: SMP_Process_Rx_Packet+182↑p
.text:08031964                                                          ; SMP_Process_Rx_Packet+2EA↑p ...
.text:08031964
.text:08031964                  var_38          = -0x38
.text:08031964                  var_31          = -0x31
.text:08031964                  var_2A          = -0x2A
.text:08031964                  var_23          = -0x23
.text:08031964
.text:08031964 2D E9 F0 43              PUSH.W  {R4-R9,LR}
.text:08031968 8B B0                    SUB     SP, SP, #0x2C
.text:0803196A 6F 46                    MOV     R7, SP
.text:0803196C 89 46                    MOV     R9, R1
.text:0803196E 90 46                    MOV     R8, R2
```

Рисунок 5 – Эпилог и пролог функций для STM32WBA6
Figure 5 – Function epilogue and prologue for STM32WBA6

Описанная выше закономерность для малых входных последовательностей не проявляется на графиках для ESP32, поскольку, в отличие от STM32WBA6, подавляющее большинство функций файла программы микроконтроллера ESP32 начинается с трехбайтовой инструкции выделения памяти на стеке. Тем не менее, для других машинных архитектур существенное уменьшение длины входной последовательности может быть критичным. В частности, для архитектур с длинными инструкциями.

Исходя из всех суждений, приведенных выше, можно сделать вывод о том, что оптимальная длина входной последовательности для решения поставленной задачи распознавания – это некоторое среднее значение. Этому среднему значению для обеих машинных архитектур в случае стандартного выравнивания соответствует длина 50.

Вес класса 1 в функции потерь бинарной кросс-энтропии. Графики показателей метрик для весовых коэффициентов функции потерь из списка наложились друг на друга. При обучении брались следующие весовые коэффициенты для класса 1: 0,5; 0,8; 0,95; 0,99; 0,998. Изменение данного гиперпараметра не изменяет показатели метрик в принципе в случае стандартного выравнивания.

Совпадение графиков можно объяснить слишком большим дисбалансом классов в обучающей и тестирующей выборках. Для исходных данных это соотношение примерно равно 1 к 400.

Обучение со случайным выравниванием

Случайное выравнивание означает, что в исходных последовательностях обучающей и тестирующей выборок все нулевые байты выравнивания заменяются на случайные байты из 0, 255.

В работе «Black-box Attacks Against Neural Binary Function Detection» приводятся примеры преднамеренных и непреднамеренных атак на модели нейронных сетей, распознающих границы функций внутри бинарного файла. В контексте преднамеренных атак авторы в том числе упоминают использование необычных настроек стандартных систем сборки, которые на практике часто используются разработчиками вредоносного программного обеспечения. Одной из атак, повлиявших на качество работы моделей, является замена байтов заполнения на байты исполняемых инструкций. Идею подобных атак развивают авторы статьи «Padding Matters». По словам авторов, на данный момент уже встречаются обфускаторы бинарного кода, изменяющие байты выравниваний. Тем не менее, при проверке работы RNN модели на x86 и x86–64 бинарных файлах со случайным выравниванием метрики качества для обеих архитектур сильно упали. Это говорит о том, что модель сильно полагалась на выравнивания при распознавании [24].

В рамках данной работы предлагается рассматривать случайные выравнивания именно как способ проверки качества обработки моделью прологов функций в отсутствие стандартных выравниваний.

Количество нейронов в рекуррентном слое. На Рисунках 6 и 7 изображены графики метрики f1 для класса 1 в зависимости от эпохи обучения для ESP32 и STM32WBA6 соответственно. Изменяемый гиперпараметр модели – количество нейронов в рекуррентном слое. При этом в обучающей и тестирующей выборках стандартные нулевые выравнивания были заменены на случайные значения.

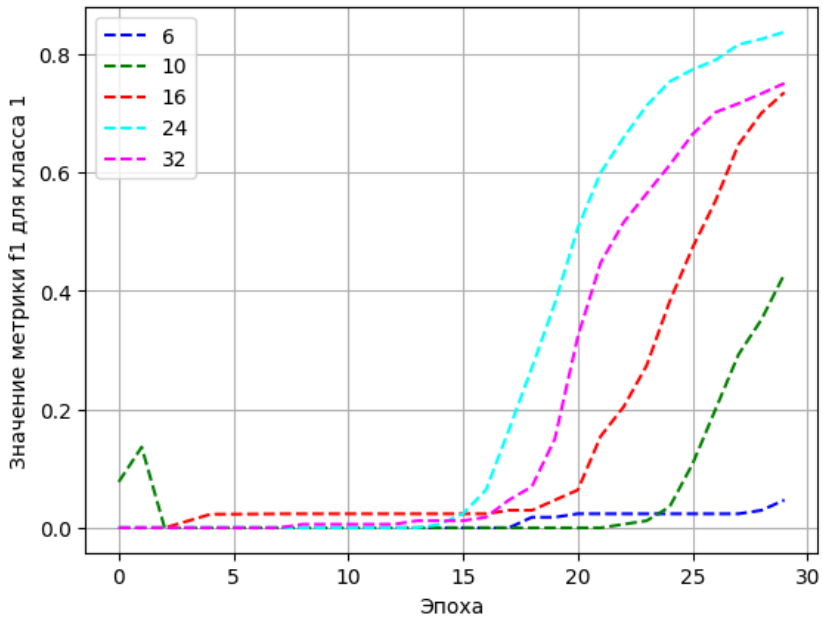

Рисунок 6 – График метрики f1 для случайного выравнивания и ESP32 при изменении количества нейронов в рекуррентном слое

Figure 6 – F1 metric graph for random alignment and ESP32 as the number of neurons in the recurrent layer changes

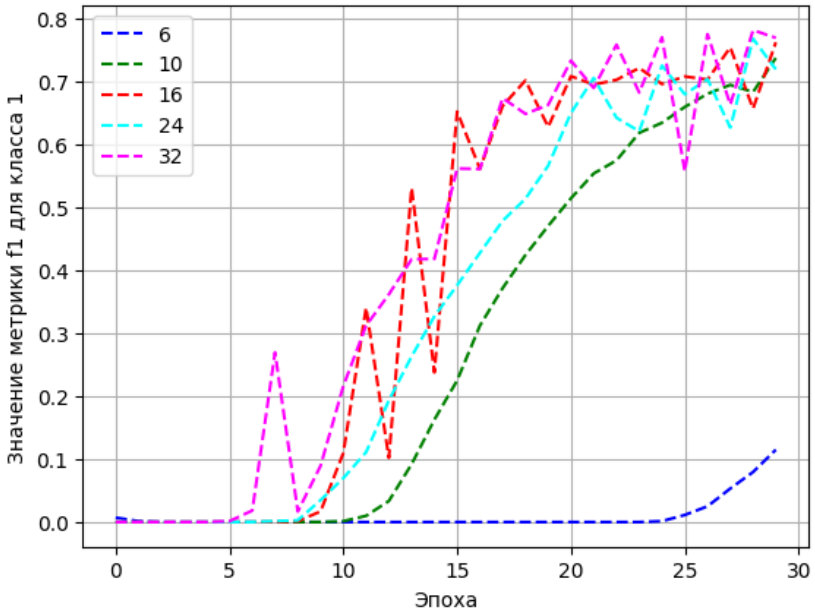

Рисунок 7 – График метрики f1 для случайного выравнивания и STM32WBA6 при изменении количества нейронов в рекуррентном слое
Figure 7 – F1 metric graph for random alignment and STM32WBA6 as the number of neurons in the recurrent layer changes

Поведение графиков метрики качества f1 для класса 1 у разного количества нейронов в рекуррентном слое почти аналогично графикам модели со стандартным выравниванием за исключением единственной аномалии в случае для ESP32 с 24 нейронами в рекуррентном слое.

В ходе проведения эксперимента возникло предположение об обобщаемости задачи распознавания начал функций в бинарных файлах с различными видами выравниваний. Под обобщаемостью задачи подразумевается возможность работы с оптимальными значениями метрик качества рассматриваемой нейронной сети, обученной со случайным заполнением, на данных со случайным или со стандартным заполнением.

Для полноты исследования также стоит рассмотреть обратную ситуацию – стандартное заполнение в обучающей выборке и случайное или стандартное заполнение в тестирующей выборках. Таким образом, необходимо рассмотреть четыре возможных случая:

– стандартное выравнивание и в обучающей, и в тестирующей выборках;

– стандартное выравнивание в обучающей и случайное выравнивание в тестирующей;

– случайное выравнивание в обучающей и стандартное заполнение в тестирующей;

– случайное выравнивание и в обучающей, и в тестирующей выборках.

Не рассматриваются случаи с одновременным использованием стандартного и случайного выравниваний в выборках, исходя из предположения о том, что в одним или нескольких схожих бинарных файлах разработчик будет использовать одинаковые инструменты сборки и обфускации.

На Рисунках 8 и 9 изображены графики метрики f1 для класса 1 для четырех случаев для ESP32 и STM32WBA6 соответственно.

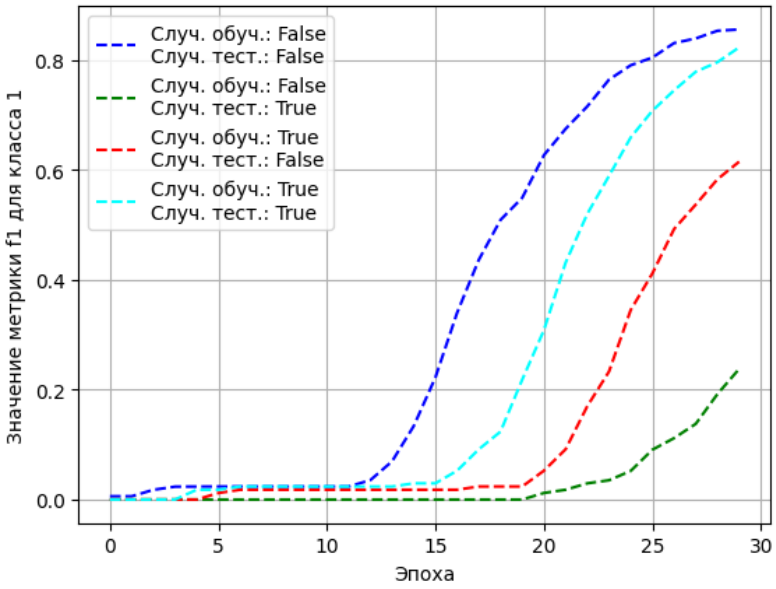

Рисунок 8 – График метрики f1 для ESP32 и комбинаций случайного/стандартного выравнивания в обучающей и тестирующей выборках
Figure 8 – F1 metric graph for ESP32 and random/standard alignment combinations in training and testing sets

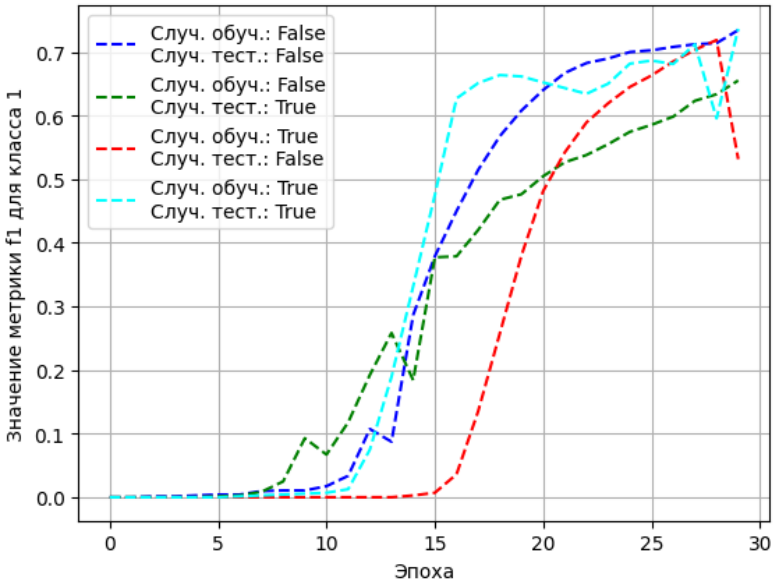

Рисунок 9 – График метрики f1 для STM32WBA6 и комбинаций случайного/стандартного выравнивания в обучающей и тестирующей выборках
Figure 9 – F1 metric graph for STM32WBA6 and random/standard alignment combinations in training and testing sets

Исходя из полученных графиков можно сделать вывод о необобщаемости задачи распознавания функций для модели со случайным выравниванием. Особенно это заметно для графика метрики f1 класса 1 для ESP32. Модель, обученная на случайном выравнивании, показывает низкие показатели оценивающих метрик при работе на стандартном выравниванием. Аналогичный вывод делается и для модели, обученной на стандартном выравнивании.

Таким образом, исследователю необходимо выбирать одну из двух моделей для распознавания функций в новом бинарном файле. Выравнивание данных в обучающей

выборке должно совпадать с выравниванием в рассматриваемом бинарном файле. В противном случае, может сильно ухудшиться качество распознавания.

Длина входной последовательности. На Рисунках 10 и 11 изображены графики метрики f1 для класса 1 в зависимости от эпохи обучения для ESP32 и STM32WBA6 соответственно. Изменяемый гиперпараметр модели – длина входной последовательности. При этом в обучающей и тестирующей выборках стандартные нулевые выравнивания были заменены на случайные значения.

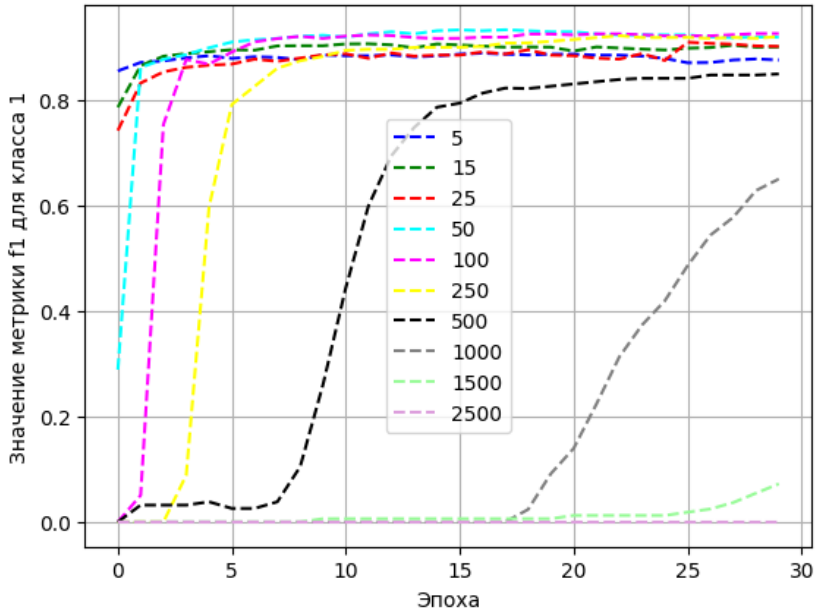

Рисунок 10 – График метрики f1 для случайного выравнивания и ESP32 при изменении длины входной последовательности

Figure 10 – F1 metric graph for random alignment and ESP32 as the input sequence length changes

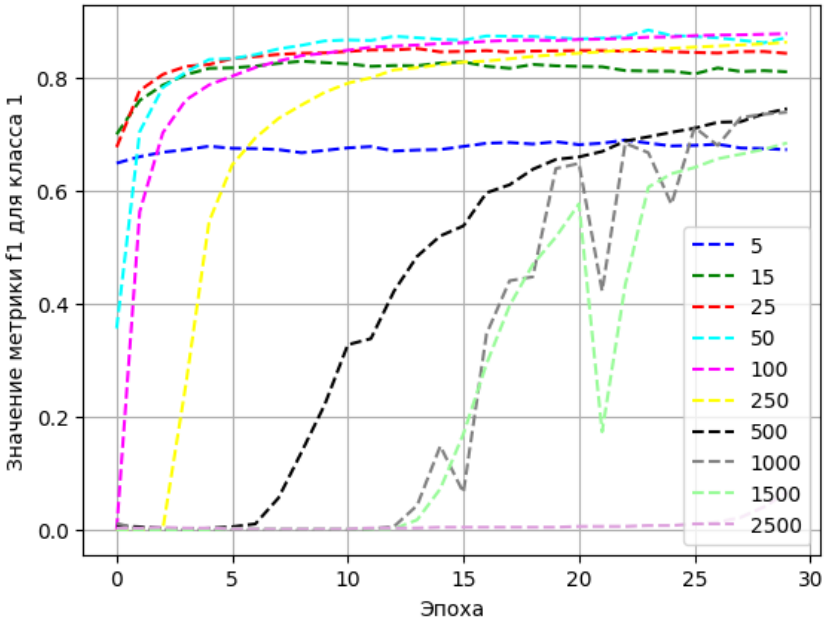

Рисунок 11 – График метрики f1 для случайного выравнивания и STM32WBA6 при изменении длины входной последовательности

Figure 11 – F1 metric graph for random alignment and STM32WBA6 as the input sequence length changes

Поведение графиков метрики качества f1 для класса 1 для разных длин входной последовательности почти аналогично графикам модели со стандартным выравниванием. Оптимальная длина входной последовательности все также соответствует среднему значению длины. Для случайного выравнивания это длина 100 байт для обеих машинных архитектур.

Вес класса 1 в функции потерь бинарной кросс-энтропии. Графики показателей метрик для весовых коэффициентов функции потерь из списка 0,8; 0,95; 0,99; 0,998 наложились друг на друга. Изменение данного гиперпараметра не изменяет показатели метрик и в случае случайного выравнивания.

Обсуждение

Ранее в работе были сформулированы предположения о гиперпараметрах модели, изменение которых способно повлиять на показатели метрик качества. По результатам обучения моделей были сделаны следующие выводы об этих параметрах.

– Количество нейронов в рекуррентном слое – параметр средней значимости. Увеличение количества нейронов может дополнительно незначительно улучшить качество работы модели, однако при увеличении более чем в 2 раза показатели метрик сходятся. Это справедливо для обоих видов выравниваний.

– Длина входной последовательности байт – наиболее значимый гиперпараметр. Разделение потока байт программы на длинные входные последовательности приводит к избытку бесполезного контекста. В результате увеличивается число эпох, необходимое для сходимости обучения модели. В то же время разделение на малые последовательности может привести к нехватке полезного контекста в последовательностях байтов, обладающих свойством начала функции. Под полезным контекстом в данном случае подразумеваются выравнивания, прологи и эпилоги, а под бесполезным – тело функции без пролога и эпилога, строки и другие константы. Оптимальным значением гиперпараметра длины входной последовательности для достижения наилучших значений показателей метрик качества будет некоторое среднее значение. Для ESP32 и STM32WBA6 это значение составило 50 для стандартного выравнивания и 100 для случайного.

– Веса функции потерь – незначимый гиперпараметр.

Значения оценивающих метрик для стандартного и случайного выравниваний, а также соответствующие им значения гиперпараметров исходной и улучшенной моделей приведены в Таблицах 1 и 2.

Таблица 1 – Значения оценивающих метрик и соответствующие им значения гиперпараметров исходной RNN модели
Table 1 – Evaluation metrics and hyperparameters values for original RNN model

Микро-контроллер	Выравни-вание	Длина последова-тельности	Кол-во нейронов	Веса функции потерь	F1 для класса 1	Взвешен-ное F1
ESP32	Стандартное	1000	16	–	0,717	0,998
	Случайное	1000	16	–	0,735	0,998
STM32WBA6	Стандартное	1000	16	–	0,724	0,995
	Случайное	1000	16	–	0,739	0,995

Таблица 2 – Значения оценивающих метрик и соответствующие им значения гиперпараметров улучшенной модели
Table 2 – Evaluation metrics and hyperparameters values for improved model

Микро-контроллер	Выравни-вание	Длина последова-тельности	Кол-во нейронов	Веса функции потерь	F1 для класса 1	Взвешен-ное F1
ESP32	Стандартное	50	64	–	0,918	0,999
	Случайное	100	64	–	0,924	0,999
STM32WBA6	Стандартное	50	64	–	0,904	0,998
	Случайное	100	64	–	0,895	0,998

В процессе обучения моделей был сделан вывод о необобщаемости задачи распознавания функций. Модель, обученная на случайных выравниваниях, плохо справлялась с распознаванием начал функций со стандартными выравниваниями. Аналогично плохо работала модель, обученная на стандартных выравниваниях, на тестирующей выборке со случайными выравниваниями.

Для выбора используемой модели (для установления типа выравнивания) исследователю необходимо вручную проанализировать фрагмент бинарного файла с исполняемым машинным кодом.

Ошибки распознавания.

Ошибки 1 рода. Типичная ошибка 1 рода распознавания начала функции приведена ниже на Рисунке 12.

```
:000E54C0 B1                            DCB 0xB1
:000E54C1 1A                            DCB 0x1A
:000E54C2 00                            DCB    0
:000E54C3 00                            DCB    0
:000E54C4
:000E54C4                   ; =============== S U B R O U T I N E ===============
:000E54C4
:000E54C4
:000E54C4                   sub_E54C4
:000E54C4 F8 B5                         PUSH            {R3-R7,LR}
:000E54C6 03 08                         LSRS            R3, R0, #0x20 ; ' '
:000E54C8 04 00                         MOVS            R4, R0
:000E54CA 00 00                         MOVS            R0, R0
:000E54CC 00 00                         MOVS            R0, R0
```

Рисунок 12 – Ошибка 1 рода распознавания начала функции для STM32WBA6
Figure 12 – False positive prologue recognition for STM32WBA6

Ошибка возникает в служебных отладочных секциях файла программы микроконтроллера, содержащих адресные, размерные, индексные и флаговые константы. В данном случае в подпоследовательности байтов «00 00 F8 B5 03 08 04 00» распознаются признаки наличия функции по адресу 0xE54C5:

– Байты «00 00» ошибочно определяются как выравнивание функций по адресам, кратным 4.

– Байты «F8 B5» определяются как инструкция PUSH {R3–R7, LR} загрузки значений регистров в стек из набора Thumb-2 архитектуры ARMv8-M. При этом младший бит байта «B5» определяет загрузку регистра LR адреса возврата в стек, что свойственно прологам функций.

– Последующие байты определяются моделью как типовые инструкции работы с данными, переданными в качестве параметров функции.

В качестве дополнительного эксперимента были поочередно заменены байты «00 00» на байты случайной инструкции Thumb-2 и инвертирован младший бит байта «B5», после чего было еще раз запущено распознавание. В обоих случаях модель определила байт по адресу 0xE54C5 как байт, не обладающий признаком начала функции, то есть модель отработала верно.

Ошибки 2 рода. Типичная ошибка 2 рода распознавания начала функции приведена ниже на Рисунке 13.

```
:0004258C                 ; =============== S U B R O U T I N E =======================================
:0004258C
:0004258C
:0004258C
:0004258C                 sub_4258C                               ; CODE XREF: sub_3FEE8+A↑p
:0004258C                                                         ; sub_42666+4↓j ...
:0004258C 80 B5                           PUSH            {R7,LR}
:0004258E 06 48                           LDR             R0, =0x2000E4A8
:00042590 01 88                           LDRH            R1, [R0]
:00042592 21 B1                           CBZ             R1, loc_4259E
:00042594 04 30                           ADDS            R0, #4
:00042596 BD E8 80 40                     POP.W           {R7,LR}
:0004259A 00 F0 09 B8                     B.W             loc_425B0
:0004259E                 ; ---------------------------------------------------------------------------
:0004259E
:0004259E                 loc_4259E                               ; CODE XREF: sub_4258C+6↑j
:0004259E 03 48                           LDR             R0, =0x808B353
:000425A0 55 21                           MOVS            R1, #0x55 ; 'U'
:000425A2 CA F7 F7 FF                     BL              sub_D594
:000425A2                 ; ---------------------------------------------------------------------------
:000425A6 00                              DCB    0
:000425A7 BF                              DCB 0xBF
:000425A8 A8 E4 00 20     off_425A8       DCD 0x2000E4A8          ; DATA XREF: sub_4258C+2↑r
:000425AC 53 B3 08 08     dword_425AC     DCD 0x808B353           ; DATA XREF: sub_4258C:loc_4259E↑r
:000425B0                 ; ---------------------------------------------------------------------------
:000425B0
:000425B0                 loc_425B0                               ; CODE XREF: sub_4258C+E↑j
:000425B0 01 68                           LDR             R1, [R0]
:000425B2 48 F2 4E 32                     MOVW            R2, #0x834E
:000425B6 A1 FB 02 12                     UMULL.W         R1, R2, R1, R2
:000425BA 02 EB 51 01                     ADD.W           R1, R2, R1,LSR#1
:000425BE 21 F0 00 42                     BIC.W           R2, R1, #0x80000000
:000425C2 00 29                           CMP             R1, #0
:000425C4 48 BF                           IT MI
:000425C6 51 1C                           ADDMI           R1, R2, #1
:000425C8 01 60                           STR             R1, [R0]
:000425CA 08 46                           MOV             R0, R1
:000425CC 70 47                           BX              LR
:000425CC                 ; End of function sub_4258C
```

Рисунок 13 – Ошибка 2 рода распознавания начала функции для STM32WBA6
Figure 13 – False negative prologue recognition for STM32WBA6

В данном случае модель не смогла распознать функцию по адресу 0x425B0, а автоанализ IDA Pro посчитал нераспознанный блок кода продолжением распознанной ранее функции по адресу 0x4258C. Причиной возникновения ошибки является отсутствие явных типовых признаков наличия функции в последовательности байт:

– Вместо выравнивания функции компилятор разместил 4-х байтовые константы «A8E40020» и «53B30808», используемые инструкциями LDR загрузки данных из памяти в регистры. Эти константы ошибочно воспринимаются моделью как последовательность инструкций обработки данных в регистрах и условного перехода из Thumb-2.

– Система сборки скомпилировала нетипичные эпилог функции 0x4258C, заканчивающийся на байте по адресу 0x425A5 включительно, и пролог функции 0x425B0. Вероятно, сделано это было в целях оптимизации. Однако нетипичность пролога и эпилога усложняют распознавание функции.

Расширение для IDA Pro. Репозиторий проекта содержит в себе 2 функциональные части*:

* Tinkerrer. *RNN-Function-Finder.* GitHub. URL: https://github.com/Tinkerrer/RNN-Function-Finder (дата обращения: 12.02.2026).

– Консольная утилита, разработанная на Python3 с использованием библиотеки машинного обучения tensorflow. Данная утилита содержит в себе программную реализацию модели. Назначение утилиты – обучение модели и сохранение обученных весов для последующего использования непосредственно с самим расширением внутри IDA Pro.

– Расширение для одной из последних версии IDA Pro 9.1. Расширение разработано на Python3 с использованием IDA Python фреймворка от компании Hexrays.

Заключение

В настоящей статье было представлено решение одной из ключевых подзадач реверс-инжиниринга – задачи распознавания начал функций в бинарном файле. Предлагаемое авторами решение частично автоматизирует распознавание начал функций, используя модель нейронной сети с двунаправленным рекуррентным слоем.

В частности, были представлены:

– Описание модели нейронной сети. Модель представляет собой оптимизированную RNN модель.

– Результаты исследований отдельных гиперпараметров модели. Был выделен наиболее значимый параметр, определяющий наилучшие значения метрик качества. Также были представлены оптимальные значения гиперпараметров для бинарных файлов, собранных под микропроцессоры ESP32 и STM32WBA6 архитектур Xtensa Little Endian и ARMv8-M соответственно.

– Описание и исходный код разработанного расширения для дизассемблера IDA Pro, одного из самых используемых инструментов обратной разработки.

Реверс-инжиниринг бинарных файлов редких машинных архитектур иногда подразумевает отсутствие разнообразия бинарных файлов для сравнительного анализа и отсутствие возможности динамического анализа. Ключевой особенностью предлагаемого авторами решения является то, что оно учитывает описанную выше прикладную специфику задач реверс-инжиниринга. В данном случае речь идет о малой выборке обучающих данных, необходимой для обучения модели нейронной сети. Автор предлагает два сценария использования результатов данной работы:

– Автоматизированный перенос разметки функций из одного разобранного бинарного файла в другой, если перед исследователем стоит задача анализа нескольких схожих по структуре бинарных файлов. Данными для обучения в этом случае станет уже ранее вручную разобранный исследователем бинарный файл.

– Автоматизированная разметка анализируемого бинарного файла посредством сборки собственного бинарного файла и обучения модели на нем. Исходный код проектов для сборки может быть взят из открытых репозиториев. Важным условием является определение окружения и инструментов сборки анализируемого бинарного файла. Репозиторий расширения для IDA Pro содержит в себе веса моделей для ESP32 и STM32WBA6, собранных посредством стандартных фреймворков разработки от компаний производителей этих микроконтроллеров.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Wartell R., Zhou Y., Hamlen K.W., Kantarcioglu M., Thuraisingham Bh. Differentiating code from data in x86 binaries. In: *Machine Learning and Knowledge Discovery in Databases: Proceedings: Part III: European Conference (ECML PKDD 2010), 05–09 September 2011, Athens, Greece.* Berlin, Heidelberg: Springer; 2011. P. 522–536. https://doi.org/10.1007/978-3-642-23808-6_34

2. Benkraouda H., Diwan N., Wang G. You Can't Judge a Binary by Its Header: Data-Code Separation for Non-Standard ARM Binaries Using Pseudo Labels. In: *2025 IEEE Symposium on Security and Privacy (SP), 12–15 May 2025, San Francisco, CA, USA*. IEEE; 2025. P. 3727–3745. https://doi.org/10.1109/SP61157.2025.00036
3. Qin S., Yang F., Wang H., et al. *Tady: A Neural Disassembler without Structural Constraint Violations.* arXiv. URL: https://arxiv.org/pdf/2506.13323 [Accessed 28th January 2026].
4. David Y., Alon U., Yahav E. Neural Reverse Engineering of Stripped Binaries using Augmented Control Flow Graphs. *Proceedings of the ACM on Programming Languages.* 2020;4. https://doi.org/10.1145/3428293
5. Jiang L., Jin X., Lin Zh. Beyond Classification: Inferring Function Names in Stripped Binaries via Domain Adapted LLMs. In: *32nd Annual Network and Distributed System Security Symposium (NDSS 2025), 24–28 February 2025, San Diego, California, USA*. The Internet Society; 2025. https://doi.org/10.14722/ndss.2025.240797
6. Bao T., Burket J., Woo M., Turner R., Brumley D. BYTEWEIGHT: Learning to Recognize Functions in Binary Code. In: *23rd USENIX Security Symposium, 20–22 August 2014, San Diego, CA, USA.* USENIX Association; 2014. P. 845–860.
7. He J., Li Sh., Wang X., et al. Neural-FEBI: Accurate Function Identification in Ethereum Virtual Machine Bytecode. *Journal of Systems and Software.* 2023;199. https://doi.org/10.1016/j.jss.2023.111627
8. Pei K., Guan J., Broughton M., et al. StateFormer: Fine-Grained Type Recovery from Binaries using Generative State Modeling. In: *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 23–28 August 2021, Athens, Greece.* New York: ACM; 2021. P. 690–702. https://doi.org/10.1145/3468264.3468607
9. Nitin V., Saieva A., Ray B., Kaiser G. DIRECT: A Transformer-based Model for Decompiled Identifier Renaming. In: *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021), 01–06 August 2021, Virtual Event.* Association for Computational Linguistics; 2021. P. 48–57. https://doi.org/10.18653/v1/2021.nlp4prog-1.6
10. Wang H., Qu W., Katz G., et al. jTrans: Jump-Aware Transformer for Binary Code Similarity Detection. In: *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, 18–22 July 2022, Virtual Event.* New York: ACM; 2022. P. 1–13. https://doi.org/10.1145/3533767.3534367
11. Yu Z., Cao R., Tang Q., et al. Order Matters: Semantic-Aware Neural Networks for Binary Code Similarity Detection. *Proceedings of the AAAI Conference on Artificial Intelligence.* 2020;34(01):1145–1152. https://doi.org/10.1609/aaai.v34i01.5466
12. Duan Y., Li X., Wang J., Yin H. DeepBinDiff: Learning Program-Wide Code Representations for Binary Diffing. In: *27th Annual Network and Distributed System Security Symposium (NDSS 2020), 23–26 February 2020, San Diego, California, USA.* The Internet Society; 2020. https://doi.org/10.14722/ndss.2020.24311
13. Li X., Qu Y., Yin H. PalmTree: Learning an Assembly Language Model for Instruction Embedding. In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, 15–19 November 2021, Virtual Event.* New York: ACM; 2021. P. 3236–3251. https://doi.org/10.1145/3460120.3484587
14. Gao Z., Wang H., Wang Y., Zhang Ch. Virtual Compiler Is All You Need For Assembly Code Search. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics: Volume 1: Long Papers, 11–16 August 2024, Bangkok, Thailand.* Association for Computational Linguistics; 2024. P. 3040–3051. https://doi.org/10.18653/v1/2024.acl-long.167

15. Liu Ch., Saul R., Sun Y., et al. ASSEMBLAGE: Automatic Binary Dataset Construction for Machine Learning. In: *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024 (NeurIPS 2024), 10–15 December 2024, Vancouver, BC, Canada.* 2024. https://openreview.net/pdf?id=dsK5EmmomU
16. Andriesse D., Slowinska A., Bos H. Compiler-agnostic function detection in binaries. In: *2017 IEEE European Symposium on Security and Privacy, 26–28 April 2017, Paris, France.* IEEE; 2017. P. 177–189. https://doi.org/10.1109/EuroSP.2017.11
17. Flores-Montoya A., Schulte E.M. Datalog disassembly. In: *29th USENIX Security Symposium (USENIX Security 2020), 12–14 August 2020.* USENIX Association; 2020. P. 1075–1092. https://www.usenix.org/system/files/sec20-flores-montoya.pdf
18. Shin E.Ch.R., Song D., Moazzezi R. Recognizing Functions in Binaries with Neural Networks. In: *24th USENIX Security Symposium (USENIX Security 15), 12–14 August 2015, Washington, D.C., USA.* USENIX Association; 2015. P. 611–626.
19. Pei K., Guan J., Williams-King D., Yang J., Jana S. XDA: Accurate, robust disassembly with transfer learning. In: *28th Annual Network and Distributed System Security Symposium (NDSS 2021), 21–25 February 2021, Virtual Event.* The Internet Society; 2021. https://doi.org/10.14722/ndss.2021.23112
20. Yu Sh., Qu Y., Hu X., Yin H. DeepDi: Learning a relational graph convolutional network model on instructions for fast and accurate disassembly. In: *31st USENIX Security Symposium (USENIX Security 2022), 10–12 August 2022, Boston, MA, USA.* USENIX Association; 2022. P. 2709–2725.
21. Evans R., Hawkins W., Wang B. RustBound: Function Boundary Detection over Rust Stripped Binaries. In: *Security and Privacy in Cyber-Physical Systems and Smart Vehicles: Second EAI International Conference (SmartSP 2024), 07–08 November 2024, New Orleans, LA, USA*. Cham: Springer; 2025. P. 237–256. https://doi.org/10.1007/978-3-031-93354-7_11
22. Guo W., Mu D., Xu J., et al. LEMNA: Explaining Deep Learning based Security Applications. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS 2018), 15–19 October 2018, Toronto, ON, Canada.* New York: ACM; 2018. P. 364–379. https://doi.org/10.1145/3243734.3243792
23. Springer R., Schmitz A., Leinweber A., Urban T., Dietrich Ch. *Padding Matters – Exploring Function Detection in PE Files.* arXiv. URL: https://arxiv.org/abs/2504.21520 [Accessed 9th February 2026].
24. Bundt J., Davinroy M., Agadakos I., Oprea A., Robertson W.K. Black-box Attacks Against Neural Binary Function Detection. In: *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2023), 16–18 October 2023, Hong Kong, China.* New York: ACM; 2023. https://doi.org/10.1145/3607199.3607200

ИНФОРМАЦИЯ ОБ АВТОРАХ / INFORMATION ABOUT THE AUTHORS

Шайханов Артем Серикович, студент кафедры «Информационная безопасность» Московский государственный технический университет имени Н.Э. Баумана, Москва, Российская Федерация.
e-mail: artem.shaykhanov@gmail.com

Artem S. Shaykhanov, Student at the Department of Information Security, Bauman Moscow State Technical University, Moscow, the Russian Federation.

Статья поступила в редакцию 18.03.2026; одобрена после рецензирования 06.05.2026; принята к публикации 13.05.2026.

The article was submitted 18.03.2026; approved after reviewing 06.05.2026; accepted for publication 13.05.2026.