Научная статья
DOI 10.66424/2071-8217-2026-2-8
УДК 004.056

РАСПОЗНАВАНИЕ НАЧАЛ ФУНКЦИЙ В БИНАРНЫХ ФАЙЛАХ С ИСПОЛЬЗОВАНИЕМ РЕКУРРЕНТНЫХ НЕЙРОННЫХ СЕТЕЙ

А. С. Шайханов*

Московский государственный технический университет имени Н. Э. Баумана, Москва, Россия

✉ *artem.shaykhanov@gmail.com

ДЛЯ ЦИТИРОВАНИЯ

Шайханов А. С. Распознавание начал функций в бинарных файлах с использованием рекуррентных нейронных сетей // Проблемы информационной безопасности. Компьютерные системы. 2026. № 2. С. 92–112. DOI: 10.66424/2071-8217-2026-2-8

ПОСТУПИЛА 01.04.2026
ПРИНЯТА 04.05.2026
ОПУБЛИКОВАНА 15.06.2026

Издатель: Санкт-Петербургский политехнический университет Петра Великого

АННОТАЦИЯ

Рассматривается задача распознавания начал функций в бинарных файлах, которая является одной из ключевых подзадач реверс-инжиниринга программного обеспечения. В качестве подхода к решению задачи 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. Получены результаты, на основании которых сделаны выводы о влиянии гиперпараметров модели, соответствующих особенностям машинной архитектуры и форматов бинарных файлов, на качество распознавания модели. 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 также позволило оценить устойчивость модели к изменению структуры бинарных данных. 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, бинарный файл, начало функции, рекуррентная нейронная сеть, IDA Pro

Original article
DOI 10.66424/2071-8217-2026-2-8

RECOGNIZING FUNCTION PROLOGUES IN BINARY FILES WITH RECURRENT NEURAL NETWORKS

A. S. Shaikhanov*

Bauman Moscow State Technical University, Moscow, Russia

✉ *artem.shaykhanov@gmail.com

FOR CITATION

Shaikhanov A. S. Recognizing function prologues in binary files with

ABSTRACT

The article discusses the problem of recognising function prologues in binary files, which is one of the key subtasks of software reverse engi-

recurrent neural networks. *Problems of information security. Computer systems*. 2026. No. 2, pp. 92–112.
DOI: 10.66424/2071-8217-2026-2-8
(In Russian)

RECEIVED 01.04.2026
ACCEPTED 04.05.2026
PUBLICATION 15.06.2026

neering. The proposed approach 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 obtained results allow to make conclusions about the influence of model hyperparameters on the quality of model recognition. These hyperparameters correspond to the features of the machine architecture and binary file formats. The experiments were performed on binary files of the ESP32 microcontroller with Xtensa Little Endian architecture and STM-32WBA6 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. 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, recognition, binary file, function prologue, recurrent neural network, IDA Pro

1. 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.

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, которые усложняют постоянное использование этих инструментов для решения широкого круга задач.

Цель 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.

2. МЕТОДЫ

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 [3–5];
- восстановление имен функций в бинарный файлах, не содержащих отладочную информацию [6–8];

• распознавание границ функций внутри бинарного файла [3, 4, 9–11];

• восстановление типов данных и аргументов функций внутри бинарного файла [12];

• восстановление имен переменных в декомпилированном представлении функций бинарного файла [13];

• поиск функций, семантически схожих между собой, в нескольких бинарных файлах [14, 15];

• сравнение разных версий бинарного файла (бинарный диффинг) [16];

• создание семантических векторных представлений инструкций ассемблера [17];

• создание больших наборов с собранными бинарными файлами для обучения моделей [18, 19].

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 др. Часть этих методов реализована в популярных инструментах реверс-инжиниринга – IDA Pro и Ghidra [20–22].

Опишем существующие модели нейронных сетей, распознающих начала функций внутри бинарных файлов, а также выделим преимущества и недостатки этих моделей.

RNN. В 2015 г. представлена модель, использующая двунаправленную рекуррентную нейронную сеть [10]. На вход модели подавались последовательности фиксированной длины в 1000 байт. Байты последовательности 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 поиска начал и концов функций. Далее результат работы моделей объединялся посредством простой эвристики для получения точных границ функции. Многие авторы моделей, представленных после 2015 г., использовали показатели метрик качества работы RNN для сравнения.

Преимущества модели:

• высокая скорость обучения, обусловленная простотой работы модели;

• высокая скорость работы, 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: отсутствие программной реализации от авторов статьи в открытом доступе; значительное снижение показателей качества модели для x86 и x86–64 при заполнении выравниваний случайными байтами.

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

XDA. В 2020 г. представлена модель, основанная на переносе различных контекстных зависимостей машинного кода для решения задач обратного проектирования, в том числе и для задачи распознавания начал функций [3].

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 повышает уровень понимания моделью машинного кода и 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 удаленным 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 позволяет 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: 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 г. представлена модель, работающая не с байтами, а с машинными инструкциями. Для первичной разметки инструкций в DeepDi строится супермножество всевозможных инструкций. На основе этого супермножества строится граф потока инструкций. Графовая сверточная сеть анализирует этот граф для определения реальных инструкций и их связей между собой [4].

Для поиска начал функций DeepDi использует:

• Набор эвристик, применяемых к размеченным инструкциям для определения кандидатов на начало функции. По словам авторов, 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: отсутствие программной реализации модели, что обусловлено тем, что разработанный DeepDi принадлежит коммерческой компании DeepBits; ограниченный набор архитектур – x86 и x86–64.

Выводы о существующих моделях. RNN, XDA, DeepDi обучены на больших наборах обучающих данных. По словам авторов моделей, каждая из них в свое время демонстрировала хорошие показатели оценивающих метрик. 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 предприняты попытки создать универсальный инструмент бинарного анализа, который мог бы не только восстановить границы функций, но и разметить ассемблерные инструкции, классифицировать вредоносный бинарный файл. 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), что 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 данный.

Постановка задачи и оценка качества решения. Входные данные B^F представляют собой последовательность байтов B_1, B_2, …, B_{F-1} некоторого бинарного файла размера F, где $B_i \in \mathbb{Z}_{256}$ – байт бинарного файла со смещением i.

Выходные данные O^F представляют собой последовательность признаков начала функции O_1, O_2, …, O_{F-1}, где $O_i \in \mathbb{Z}_2$. При этом $O_i=1 \Rightarrow$ байт B_i распознан как байт начала функции, в противном случае байт B_i не обладает признаком «начала функции». Количество ненулевых элементов в $\{O_1, O_2, \ldots, O_{F-1}\}$ – 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:

$$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}};$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}};$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}};$$

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}},$$

где 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 первого рода (количество случаев, когда модель распознала байт, не являющийся байтом начала функции, как байт начала функции); FN (False Negative) – ошибки второго 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 т.д. 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 первого рода за счет существенного увеличения ошибок второго 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 перечисленного, для оценки качества работы модели используется F1 метрика.

В предыдущих работах исследователи ставили перед собой разные задачи, смежные с восстановлением начал функций: восстановление конца функции, 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 для обучения и тестирования используются все секции бинарного файла, включающие в себя и данные, и код, и служебные заголовки. Это 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 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 [23].

Выбор модели нейронной сети. За основу архитектуры нейронной сети 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, 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 начал 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 г. Основное изменение – уменьшение количества нейронов выходного слоя и изменение правила выбора класса [24].

Основные причины:

• В оригинальной версии модели выходной слой имел два 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 один 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. Входными данными является последовательность байт B^F бинарного файла размера F, где $B_i \in \mathbb{Z}_{256}$. Эта последовательность разбивается на подпоследовательности длины L, а далее каждый байт B_i подпоследовательности посредством one-hot преобразования приводится к вектору действительных чисел, в котором элемент с индексом, равным значению байта B_i, является единичным, а все остальные нулевыми:

$$\text{one-hot} : \mathbb{Z}_{256}^{L} \to \mathbb{R}^{L \times 256}.$$

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, а второй – в обратном. Выходы двух этих слоев объединяются для передачи в следующий слой. Формально рекуррентный слой можно определить так:

$$\text{Recurrent layer} : \mathbb{R}^{L \times 256} \times \mathbb{R}^{N} \to \mathbb{R}^{L \times 2N}.$$

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

$$\begin{aligned} &\text{Forward layer}(\text{one-hot}_j) = \text{state}_j = \\ &= \text{Relu}(W_{\text{state,one-hot}} \times \text{one-hot}_j + \\ &+ W_{\text{state,state}} \times \text{state}_{j-1} + W_{\text{state}}), \end{aligned}$$

где $\text{state}_{j-1}, \text{state}_j \in \mathbb{R}^N$ – векторы внутреннего состояния рекуррентного слоя для $j-1$ и j входных one-hot векторов соответственно; $W_{\text{state,one-hot}} \in \mathbb{R}^{N \times 256}$, $W_{\text{state,state}} \in \mathbb{R}^{N \times N}$, $W_{\text{state}} \in \mathbb{R}^{N}$ – матрицы весов рекуррентного слоя; $W_{\text{state,one-hot}} \times \text{one-hot}_j$ – вклад текущего байта подпоследовательности; $W_{\text{state,state}} \times \text{state}_{j-1}$ – вклад предыдущих байтов подпоследовательности (вклад внутреннего состояния рекуррентного слоя); Relu – функция активации в рекуррентном слое.

Приведенное соотношение 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»:

$$\text{Activation layer}: \mathbb{R}^{L\times 2N} \to \mathbb{Z}_2^L,$$

где Sigmoid – используемая функция активации в слое.

Как было упомянуто, 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. Приведем список исследуемых гиперпараметров модели:

• 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 г. 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. Значение параметра длины входной последовательности определяется длинами прологов, эпилогов и выравниваний функций.

• Количество нейронов в рекуррентном слое. Исходная модель содержит в рекуррентном слое 16 нейронов. Количество нейронов неявным образом влияет на способность слоя обрабатывать закономерности между байтами инструкций (данных) входных последовательностей.

• Веса функции потерь. Уже упоминалась проблема дисбаланса классов. Отношение количества байт начала функции к количеству байт, не обладающих признаком «начала функции», связано с оптимальным значением весов классов функции потерь.

В рамках работы исследовалась зависимость между качеством распознавания и только теми гиперпараметрами модели, которые соответствуют особенностям машинной архитектуры или структуре самого бинарного файла. Поэтому в главе с результатами не приведены эксперименты, например, для функции активации в рекуррентном слое (tanh или различные вариации ReLU: Leaky ReLU, PReLU, ELU) или архитектуры рекуррентного слоя (GRU и LSTM). Для этих гиперпараметров не определяется прямая связь с машинной архитектурой, под которую собран бинарный файл, или форматом самого бинарного файла.

Необходимо отметить, что необоснованное усложнение архитектуры рекуррентного слоя увеличивает количество обучаемых параметров нейронной сети, что отражается на времени обучения одной эпохи и количестве эпох, требуемых для сходимости. Данный вывод согласуется с точкой зрения авторов модели RNN. Опираясь на результаты своих экспериментов, авторы делают вывод о том, что применение LSTM и GRU в качестве архитектуры рекуррентного слоя не дает значительного увеличения показателей метрик качества в сравнении с обычным SimpleRNN в задаче распознавания начал функций.

Эксперименты с исследуемыми параметрами необходимо провести для случайных байтов выравнивания, как это предложено в работе «Padding Matters» [25].

3. РЕЗУЛЬТАТЫ И ОБСУЖДЕНИЕ

Представим 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 по умолчанию:

• 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, 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. На рис. 1 изображены графики метрики F1 для класса 1 в зависимости от эпохи обучения для ESP32 и STM32WBA6. Изменяемый гиперпараметр модели – количество нейронов в рекуррентном слое.

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

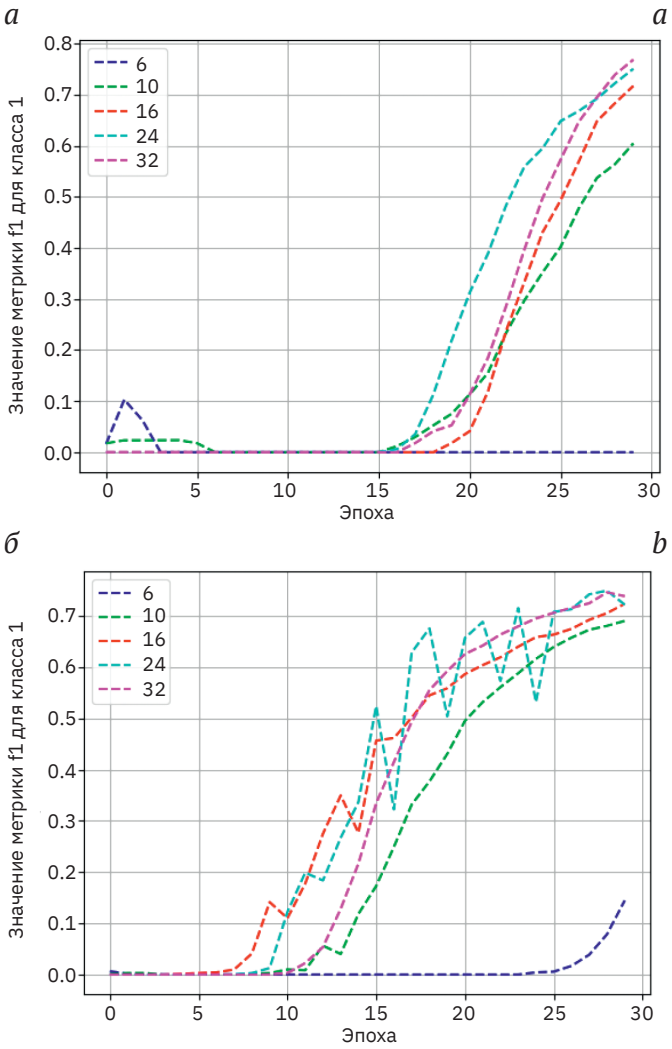

Рис. 1 | Графики метрики F1 класса 1 при изменении количества нейронов в рекуррентном слое для ESP32 (*а*) и STM32WBA6 (*б*)

Fig. 1 | Graphs of Class 1 F1 metrics with changes in the number of neurons in the recurrent layer for ESP32 (*a*) and STM32WBA6 (*b*)

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, 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 рис. 2 изображены графики метрики F1 для класса 1 в зависимости от эпохи обучения для ESP32 и STM32WBA6. Изменяемый гиперпараметр модели – длина входной последовательности.

Увеличение длин входных последовательностей приводит к избытку контекста у нейронной сети. Для распознавания начал 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

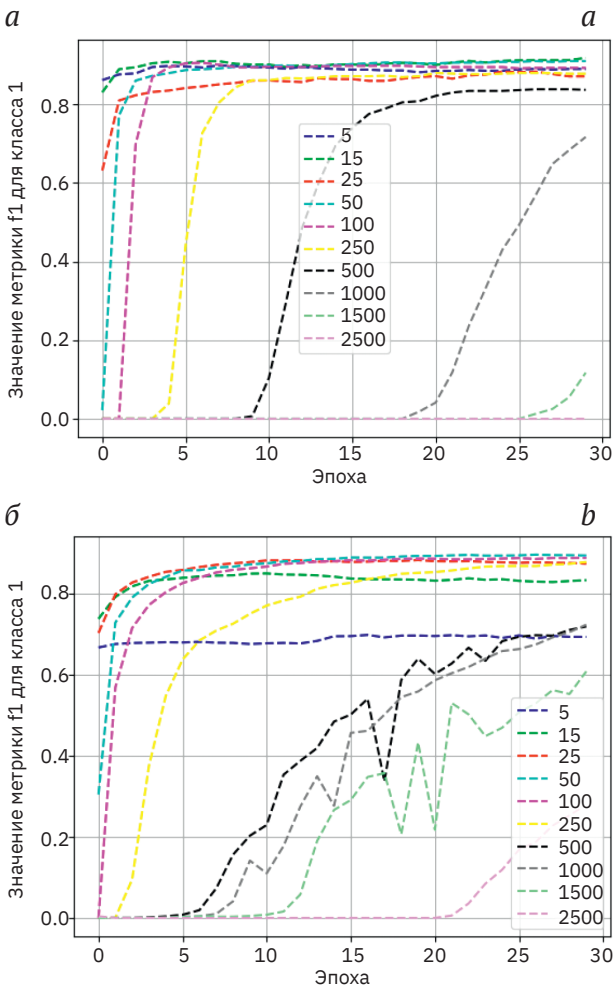

Рис. 2 | Графики метрики F1 класса 1 при изменении длины входной последовательности для ESP32 (*а*) и STM32WBA6 (*б*)

Fig. 2 | Graphs of Class 1 F1 metrics when changing the length of the input sequence for ESP32 (*a*) and STM32WBA6 (*b*)

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 рис. 3 отражены типичные эпилог функции *X* и пролог функции *X*+1 для STM32WBA6. Если посчитать байты инструкций: эпилог (6–8+ байт), пролог (6–8+ байт) и возможное выравнивание функций по адресам, кратным четырем (1–3 байта), то получится, что длины последовательности менее 15 байт недостаточно для распознавания начал функций. 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.

Описанная 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 некоторых машинных архитектур существенное уменьшение длины входной последовательности будет критичным. В частности, для архитектур с длинными инструкциями.

Исходя из всех приведенных суждений, можно сделать вывод, 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 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.

```
.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
```

Рис. 3 | Эпилог и пролог функций для STM32WBA6

Fig. 3 | Epilogue and prologue of functions for STM32WBA6

В работе [26] 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 [26].

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 рис. 4 представлены графики

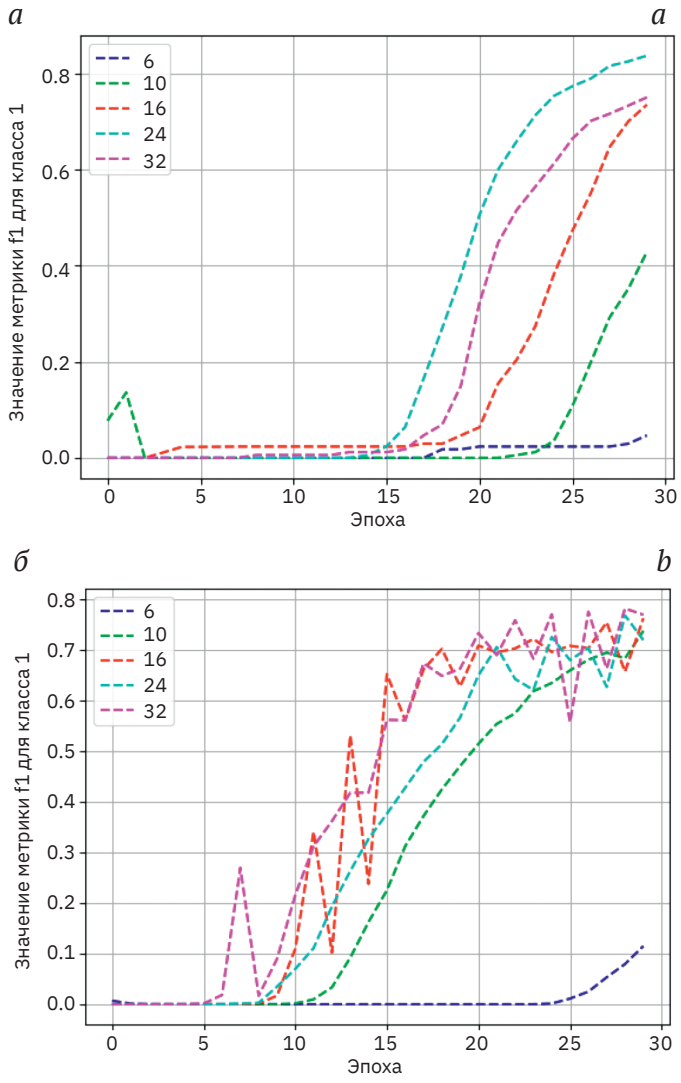

Рис. 4 | График метрики F1 класса 1 для случайного выравнивания при изменении количества нейронов в рекуррентном слое для ESP32 (*а*) и STM32WBA6 (*б*)

Fig. 4 | Graph of Class 1 F1 metric for random alignment when changing the number of neurons in the recurrent layer for ESP32 (*a*) and STM32WBA6 (*b*)

метрики F1 для класса 1 в зависимости от эпохи обучения для ESP32 и STM32WBA6. Изменяемый гиперпараметр модели – количество нейронов в рекуррентном слое. При этом в обучающей и тестирующей выборках стандартные нулевые выравнивания заменены на случайные значения.

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 стандартным заполнением.

Для полноты исследования 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 одном или нескольких схожих бинарных файлах разработчик будет использовать одинаковые инструменты сборки и обфускации.

На рис. 5 показаны графики метрики F1 для класса 1 для четырех случаев для ESP32 и STM32WBA6.

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 выравнивании. 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 рис. 6 изображены графики метрики F1 для класса 1 в зависимости от эпохи обучения для ESP32 и STM32WBA6. Изменяемый гиперпараметр модели – длина входной последовательности. При этом в обучающей и тестирующей выборках стандартные нулевые выравнивания заменены на случайные значения.

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 так же 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.

Ошибки распознавания. *Ошибки первого рода.* Типичная ошибка первого рода

распознавания начала функции приведена на рис. 7.

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 поочередно заменены байты «00 00»

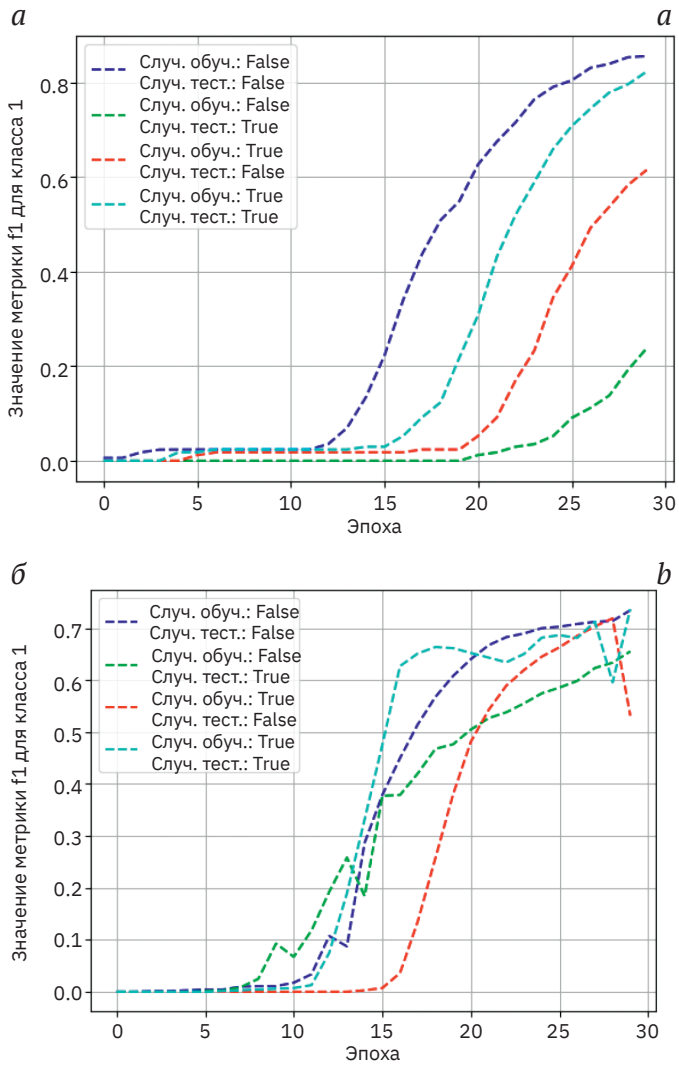

Рис. 5 | График метрики F1 класса 1 для комбинаций случайного и стандартного выравнивания в обучающей и тестирующей выборках для ESP32 (*а*) и STM32WBA6 (*б*)

Fig. 5 | Graph of Class 1 F1 metrics for combinations of random and standard alignment in training and testing samples for ESP32 (*a*) and STM32WBA6 (*b*)

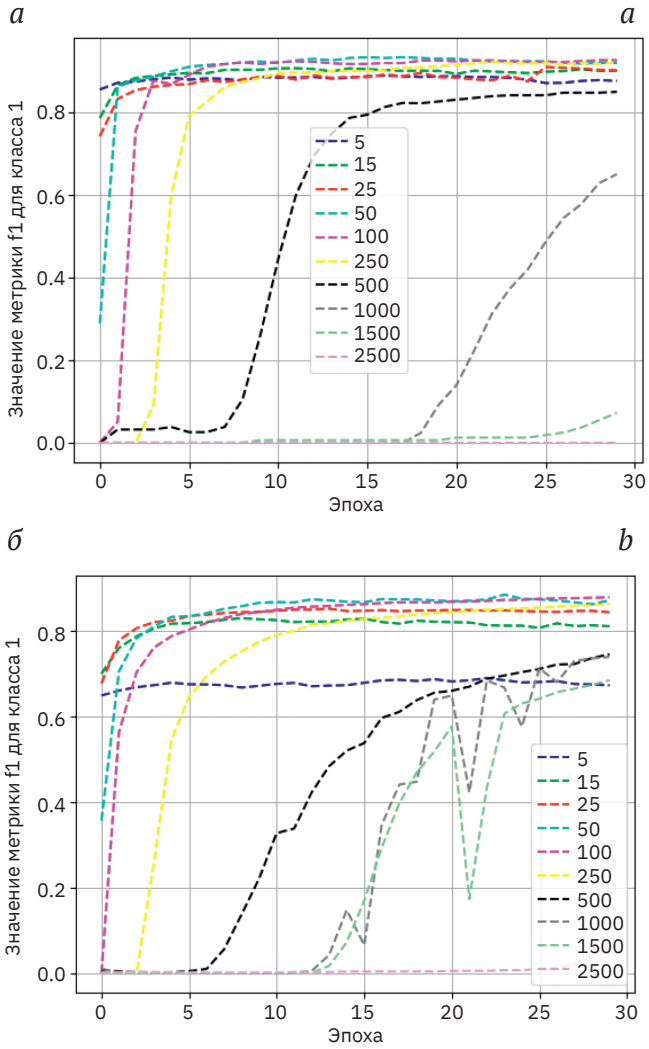

Рис. 6 | График метрики F1 класса 1 для случайного выравнивания при изменении длины входной последовательности для ESP32 (*а*) и STM32WBA6 (*б*)

Fig. 6 | Graph of Class 1 F1 metric for random alignment when changing the length of the input sequence for ESP32 (*a*) and STM32WBA6 (*b*)

```
: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
```

Рис. 7 | Ошибка первого рода распознавания начала функции для STM32WBA6

Fig. 7 | Error of the first kind of recognition of the function start for STM32WBA6

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», т. е. модель отработала верно.

Ошибки второго рода. Типичная ошибка второго рода распознавания начала функции приведена на рис. 8.

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 четырехбайтовые 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 для оптимизации генерируемого кода. Однако нетипичность пролога и эпилога усложняют распознавание функции.

Обсуждение полученных результатов. По результатам обучения моделей сделаны следующие выводы о влиянии гиперпараметров, соответствующих особенностям машинной архитектуры и форматов бинарных файлов, на показатели метрик качества:

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

• Длина входной последовательности байт – наиболее значимый гиперпараметр.

```
: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
```

Рис. 8 | Ошибка второго рода распознавания начала функции для STM32WBA6

Fig. 8 | Error of the second kind of recognition of the beginning of the function for STM32WBA6

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 табл. 1 и 2.

В процессе обучения моделей 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.

Расширение для IDA Pro. Репозиторий проекта содержит в себе две функциональные части [27]:

Таблица 1 | Значения оценивающих метрик и соответствующие им значения гиперпараметров исходной RNN модели

Table 1 | Values of the estimating metrics and their corresponding hyperparameter values of the initial RNN model

Микроконтроллер	Выравнивание	Длина последовательности	Кол-во нейронов	Веса функции потерь	F1 для класса 1	Взвешенное 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 RNN модели

Table 2 | Values of the evaluation metrics and their corresponding hyperparameter values of the improved RNN model

Микроконтроллер	Выравнивание	Длина последовательности	Кол-во нейронов	Веса функции потерь	F1 для класса 1	Взвешенное F1
ESP32	Стандартное	50	32	–	0,904	0,999
	Случайное	100	32	–	0,930	0,999
STM32WBA6	Стандартное	50	32	–	0,897	0,998
	Случайное	100	32	–	0,862	0,997

• 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.

4. ЗАКЛЮЧЕНИЕ

Представлено решение одной из ключевых подзадач реверс-инжиниринга – задачи распознавания начал функций в бинарном файле. Предлагаемое решение автоматизирует распознавание начал функций, используя модель нейронной сети.

В работе представлены:

• Описание модели нейронной сети с двунаправленным рекуррентным слоем. Модель представляет собой оптимизированную RNN модель.

• Результаты экспериментов с гиперпараметрами модели для случайного и стандартного выравниваний функций в бинарных файлах. На основе результатов сделаны выводы о влиянии на качество распознавания гиперпараметров модели (длина входной последовательности, количество нейронов в рекуррентном слое и веса функции потерь), соответствующих особенностям машинной архитектуры или форматов бинарных файлов.

• Сводная таблица оптимальных значений 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 решения является то, что оно учитывает описанную прикладную специфику задач реверс-инжиниринга, проявляющуюся в малой выборке обучающих данных. Автор предлагает два сценария использования результатов исследования:

• 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.

КОНФЛИКТ ИНТЕРЕСОВ / CONFLICT OF INTERESTS

Автор заявляет об отсутствии конфликта интересов / The author declare no conflict of interests.

СПИСОК ИСТОЧНИКОВ

1. **Wartell R., Zhou Y., Hamlen K. W. et al.** Differentiating code from data in x86 binaries // Machine Learning and Knowledge Discovery in Databases. 2011. P. 522–536. DOI: 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 // IEEE Symposium on Security and Privacy (SP). 2025. P. 3727–3745. DOI: 10.1109/SP61157.2025.00036.
3. **Pei K., Guan J., Williams-King D. et al.** XDA: accurate, robust disassembly with transfer learning // Network and Distributed Systems Security (NDSS) Symposium 2021, 21–24 February 2021, San Diego, CA, USA. 2021. DOI: 10.14722/ndss.2021.23112.
4. **Yu S., Qu Y., Hu X., Yin H.** DeepDi: Learning a relational graph convolutional network model on instructions for fast and accurate disassembly // 31st USENIX Security Symposium (USENIX Security 22). 2022. P. 2709–2725.
5. **Siliang Qin1, Fengrui Yang, Hao Wang et al.** Tady: A Neural Disassembler without Structural Constraint Violations. URL: https://arxiv.org/pdf/2506.13323 (дата обращения: 03.12.2025).
6. **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. Vol. 4 (OOPSLA). 28 p. DOI: 10.1145/3428293.
7. **Patrick-Evans J., Cavallaro L., Kinder J.** Probabilistic Naming of Functions in Stripped Binaries // Annual Computer Security Applications Conference (ACSAC 2020), 7–11 December 2020, Austin, USA. 2020. P. 373–385. DOI: 10.1145/3427228.3427265.
8. **Jiang L., Jin X., Lin Z.** Beyond Classification: Inferring Function Names in Stripped Binaries via Domain Adapted LLMs. URL: https://www.ndss-symposium.org/wp-content/uploads/2025-797-paper.pdf (дата обращения: 03.12.2025).
9. **Bao T., Burket J., Woo M., Turner R., Brumley D.** BYTEWEIGHT: Learning to Recognize Functions in Binary Code // 23rd USENIX Security Symposium. 2014. P. 845–860.
10. **Shin E.C.R., Song D., Moazzezi R.** Recognizing Functions in Binaries with Neural Networks // 24th USENIX Security Symposium (USENIX Security 15). 2015. P. 611–626.
11. **He J., Li S., Wang X., Yang J.** Neural-FEBI: Accurate Function Identification in Ethereum Virtual Machine Bytecode // Journal of Systems and Software. 2023. Vol. 199. № 111627. DOI: 10.1016/j.jss.2023.111627.
12. **Pei K., Guan J., Broughton M. et al.** StateFormer: Fine-Grained Type Recovery from Binaries using Generative State Modeling // Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021. P. 690–702. DOI: 10.1145/3468264.3468607.
13. **Nitin V., Saieva A., Ray B., Kaiser G.** A Transformer-based Model for Decompiled Identifier Renaming // Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021), 1–6 August 2021. 2021. P. 48–57.
14. **Wang H., Qu W., Katz G. et al.** jTrans: Jump-Aware Transformer for Binary Code Similarity // Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA). 2022. P. 1–13. DOI: 10.1145/3533767.3534367.
15. **Yu Z., Cao R., Tang Q. et al.** OrderMatters: Semantic-Aware Neural Networks for Binary Code Similarity Detection // Proceedings of the AAAI Conference on Artificial Intelligence. 2020. Vol. 34. P. 1145–1152. DOI: 10.1609/aaai.v34i01.5466.
16. **Duan Y., Li X., Wang J., Yin H.** DeepBinDiff: Learning Program-Wide Code Representations for Binary Diffing // Network and Distributed System Security Symposium (NDSS). 2020. DOI: 10.14722/ndss.2020.24311.
17. **Li X., Qu Y., Yin H.** PalmTree: Learning an Assembly Language Model for Instruction Embedding // ACM SIGSAC Conference on Computer and Communications Security (CCS '21). 2021. P. 3236–3251. DOI: 10.1145/3460120.3484587.
18. **Gao Z., Wang H., Wang Y., Zhang C.** Virtual Compiler Is All You Need For Assembly Code Search // Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024. P. 3040–3051. DOI: 10.18653/v1/2024.acl-long.167.

19. **Liu C., Saul R., Sun Y. et al.** ASSEMBLAGE: Automatic Binary Dataset Construction for Machine Learning. URL: https://openreview.net/pdf?id=dsK5EmmomU (дата обращения: 03.12.2025).
20. **Andriesse D., Slowinska A., Bos H.** Compiler-agnostic function detection in binaries // IEEE European Symposium on Security and Privacy. 2017. P. 251–276.
21. **Александров Я. А., Сафин Л. К., Чернов А. В., Трошина К. Н.** Определение границ подпрограмм при статическом анализе бинарных образов // Вопросы кибербезопасности. 2016. № 1 (14). С. 53–60.
22. **Flores-Montoya A., Schulte E.** Datalog disassembly // 29th USENIX Security Symposium (USENIX Security 20). 2020. P. 1075–1092.
23. **Evans R., Hawkins W., Wang B.** RustBound: Function Boundary Detection over Rust Stripped Binaries // Security and Privacy in Cyber-Physical Systems and Smart Vehicles. 2025. Vol. 622. P. 237–256.
24. **Guo W., Mu D., Xu J. et al.** LEMNA: Explaining Deep Learning based Security Applications // ACM SIGSAC Conference on Computer and Communications Security (CCS '18). 2018. P. 364–379.
25. **Springer R., Schmitz A., Leinweber A. et al.** Padding Matters – Exploring Function Detection in PE Files // arXiv:2504.21520. 2025.
26. **Bundt J., Davinroy M., Agadakos I. et al.** Black-box Attacks Against Neural Binary Function Detection // The 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23). 2023. 16 p. DOI: 10.1145/3607199.3607200.
27. RNN-Function-Finder. URL: https://github.com/Tinkerrer/RNN-Function-Finder (дата обращения: 03.12.2025).

REFERENCES

1. **Wartell R., Zhou Y., Hamlen K. W. et al.** Differentiating code from data in x86 binaries. *Machine Learning and Knowledge Discovery in Databases*. 2011, pp. 522–536. DOI: 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. IEEE Symposium on Security and Privacy (SP). 2025, pp. 3727–3745. DOI: 10.1109/SP61157.2025.00036.
3. **Pei K., Guan J., Williams-King D. et al.** XDA: accurate, robust disassembly with transfer learning. Network and Distributed Systems Security (NDSS) Symposium 2021, 21–24 February 2021, San Diego, CA, USA. 2021. DOI: 10.14722/ndss.2021.23112.
4. **Yu S., Qu Y., Hu X., Yin H.** DeepDi: Learning a relational graph convolutional network model on instructions for fast and accurate disassembly. 31st USENIX Security Symposium (USENIX Security 22). 2022, pp. 2709–2725.
5. **Siliang Qin1, Fengrui Yang, Hao Wang et al.** Tady: A Neural Disassembler without Structural Constraint Violations. URL: https://arxiv.org/pdf/2506.13323 (accessed: 03.12.2025).
6. **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. Vol. 4 (OOPSLA), 28 p. DOI: 10.1145/3428293.
7. **Patrick-Evans J., Cavallaro L., Kinder J.** Probabilistic Naming of Functions in Stripped Binaries. Annual Computer Security Applications Conference (ACSAC 2020), 7–11 December 2020, Austin, USA. 2020, pp. 373–385. DOI: 10.1145/3427228.3427265.
8. **Jiang L., Jin X., Lin Z.** Beyond Classification: Inferring Function Names in Stripped Binaries via Domain Adapted LLMs. URL: https://www.ndss-symposium.org/wp-content/uploads/2025-797-paper.pdf (accessed: 03.12.2025).
9. **Bao T., Burket J., Woo M., Turner R., Brumley D.** BYTEWEIGHT: Learning to Recognize Functions in Binary Code. 23rd USENIX Security Symposium. 2014, pp. 845–860.
10. **Shin E.C.R., Song D., Moazzezi R.** Recognizing Functions in Binaries with Neural Networks. 24th USENIX Security Symposium (USENIX Security 15). 2015, pp. 611–626.
11. **He J., Li S., Wang X., Yang J.** Neural-FEBI: Accurate Function Identification in Ethereum Virtual Machine Bytecode. Journal of Systems and Software. 2023. Vol. 199. No. 111627. DOI: 10.1016/j.jss.2023.111627.

12. **Pei K., Guan J., Broughton M. et al.** StateFormer: Fine-Grained Type Recovery from Binaries using Generative State Modeling. Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021, pp. 690–702. DOI: 10.1145/3468264.3468607.
13. **Nitin V., Saieva A., Ray B., Kaiser G.** A Transformer-based Model for Decompiled Identifier Renaming. Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021), 1–6 August 2021. 2021, pp. 48–57.
14. **Wang H., Qu W., Katz G. et al.** jTrans: Jump-Aware Transformer for Binary Code Similarity. Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA). 2022, pp. 1–13. DOI: 10.1145/3533767.3534367.
15. **Yu Z., Cao R., Tang Q. et al.** OrderMatters: Semantic-Aware Neural Networks for Binary Code Similarity Detection. Proceedings of the AAAI Conference on Artificial Intelligence. 2020. Vol. 34, pp. 1145–1152. DOI: 10.1609/aaai.v34i01.5466.
16. **Duan Y., Li X., Wang J., Yin H.** DeepBinDiff: Learning Program-Wide Code Representations for Binary Diffing. Network and Distributed System Security Symposium (NDSS). 2020. DOI: 10.14722/ndss.2020.24311.
17. **Li X., Qu Y., Yin H.** PalmTree: Learning an Assembly Language Model for Instruction Embedding. ACM SIGSAC Conference on Computer and Communications Security (CCS '21). 2021, pp. 3236–3251. DOI: 10.1145/3460120.3484587.
18. **Gao Z., Wang H., Wang Y., Zhang C.** Virtual Compiler Is All You Need For Assembly Code Search. Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024, pp. 3040–3051. DOI: 10.18653/v1/2024.acl-long.167.
19. **Liu C., Saul R., Sun Y. et al.** ASSEMBLAGE: Automatic Binary Dataset Construction for Machine Learning. URL: https://openreview.net/pdf?id=dsK5EmmomU (accessed: 03.12.2025).
20. **Andriesse D., Slowinska A., Bos H.** Compiler-agnostic function detection in binaries. IEEE European Symposium on Security and Privacy. 2017, pp. 251–276.
21. **Aleksandrov Ya. A., Safin L. K., Chernov A. V., Troshina K. N.** Determining subroutine boundaries in static analysis of binary images. *Voprosy kiberbezopasnosti – Cybersecurity Issues*. 2016. No. 1 (14), pp. 53–60. (In Russian)
22. **Flores-Montoya A., Schulte E.** Datalog disassembly. 29th USENIX Security Symposium (USENIX Security 20). 2020, pp. 1075–1092.
23. **Evans R., Hawkins W., Wang B.** RustBound: Function Boundary Detection over Rust Stripped Binaries. *Security and Privacy in Cyber-Physical Systems and Smart Vehicles*. 2025. Vol. 622, pp. 237–256.
24. **Guo W., Mu D., Xu J. et al.** LEMNA: Explaining Deep Learning based Security Applications. ACM SIGSAC Conference on Computer and Communications Security (CCS '18). 2018, pp. 364–379.
25. **Springer R., Schmitz A., Leinweber A. et al.** Padding Matters – Exploring Function Detection in PE Files. *arXiv:2504.21520*. 2025.
26. **Bundt J., Davinroy M., Agadakos I. et al.** Black-box Attacks Against Neural Binary Function Detection. The 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23). 2023, 16 p. DOI: 10.1145/3607199.3607200.
27. RNN-Function-Finder. URL: https://github.com/Tinkerrer/RNN-Function-Finder (accessed: 03.12.2025).

СВЕДЕНИЯ ОБ АВТОРЕ / INFORMATION ABOUT AUTHOR

ШАЙХАНОВ Артем Серикович – студент, Московский государственный технический университет имени Н. Э. Баумана, Россия, 105005, Москва, ул. 2-я Бауманская, д. 5, стр. 1
E-mail: artem.shaykhanov@gmail.com

SHAIKHANOV Artem S. – Student, Bauman Moscow State Technical University, Russia, 105005, Moscow, 2-ya Baumanskaya str., 5, build. 1