«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Направление	09.03.04 - Программная инженерия
Профиль	Разработка программно-информационных систем
Факультет	КТИ
Кафедра	МО ЭВМ

К защите допустить

Зав. кафедрой А.А. Лисс

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: Разработка модели искусственного интеллекта для прогнозирования поведения программ в ОС

Студент _____ П.А. Бодунов
подпись

Руководитель *доцент, к.т.н* _____ А.А. Лисс
подпись

Консультант _____ А.В. Гаврилов
подпись

Санкт-Петербург

2024

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю
Зав. кафедрой МО ЭВМ
_____ А.А. Лисс
«___»_____20___ г.

Студент Бодунов П.А. Группа 0303

Тема работы: Разработка модели искусственного интеллекта для прогнозирования поведения программ в ОС

Место выполнения ВКР: СПБГЭТУ «ЛЭТИ», кафедра МО ЭВМ

Исходные данные (технические требования):

Необходимо разработать модель искусственного интеллекта для предсказания следующего системного вызова для конкретных программы.

Содержание ВКР:

Введение, Обзор предметной области, Формулировка требований к решению и постановка задачи, Проектирование модели ИИ, Реализация модели, Заключение

Перечень отчетных материалов: пояснительная записка, иллюстративный материал

Дополнительные разделы: безопасность жизнедеятельности

Дата выдачи задания «___»_____20___ г.

Дата представления ВКР к защите «___»_____20___ г.

Студент _____ П.А. Бодунов

Руководитель _____ А.А. Лисс
доцент, к.т.н

Консультант _____ А.В. Гаврилов

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю
Зав. кафедрой МО ЭВМ
_____ А.А. Лисс
«___»_____20___ г.

Студент Бодунов П.А. Группа 0303

Тема работы: Разработка модели искусственного интеллекта для прогнозирования поведения программ в ОС

№ п/п	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	17.11 – 30.01
2	Анализ требований и проектирование приложения	30.01 – 01.02
3	Разработка приложения	01.02 – 15.04
5	Оформление пояснительной записки	15.04 – 07.05
6	Оформление иллюстративного материала	07.05 – 12.05

Студент _____ П.А. Бодунов

Руководитель _____ А.А. Лисс
доцент, к.т.н

Консультант _____ А.В. Гаврилов

РЕФЕРАТ

Пояснительная записка 51 стр., 17 рис., 2 табл., 26 ист.

НЕЙРОННЫЕ СЕТИ, ТРАССИРОВКА, СИСТЕМНЫЕ ВЫЗОВЫ, ПРЕДСКАЗАНИЕ СИСТЕМНЫХ ВЫЗОВОВ.

Объектом исследования является модель прогнозирования системных вызовов.

Предметом исследования является точность и время предсказания системных вызовов.

Цель работы: разработка модели искусственного интеллекта для прогнозирования действий программ в реальном времени для улучшения производительности операционных систем.

В данной выпускной квалификационной работе изучаются подходы к предсказанию поведения программ с целью выявления наиболее эффективного метода для прогнозирования системных вызовов. Знание поведения программ полезно при разработке более совершенных методов планирования процессов в ОС. Проведено исследование различных аналогов для модели прогнозирования. Были рассмотрены предиктор последнего значения, предиктор на основе таблиц, унифицированный метод, предиктор на основе дерева решений, а также предсказание на основе Sequence-to-Sequence модели. Нейронная сеть, состоящая из слоев: Word Embedding, Bidirectional LSTM и линейного, является более эффективным методом прогнозирования по сравнению с другими методами, поскольку он способен учитывать и использовать информацию как о предыдущих, так и о последующих временных шагах. Это позволяет модели более полно и точно анализировать зависимости в данных и делать более точные прогнозы. Исследование данного метода показало, что он недостаточно хорош за счет своей долгой работы, поэтому было принято решение использовать нейросеть со слоями: Word Embedding и линейным. Данный метод показал допустимое время прогнозирования для использования в ОС.

ABSTRACT

In this final qualifying work, approaches to predicting program behavior are studied in order to identify the most effective method for predicting system calls. Knowledge of program behavior is useful in developing more advanced methods for planning processes in the OS. A study of various analogues for the forecasting model has been conducted. The predictor of the last value, a predictor based on tables, a unified method, a predictor based on a decision tree, as well as prediction based on a Sequence-to-Sequence model were considered. A neural network consisting of layers: Word Embedding, Bidirectional LSTM and linear is a more effective forecasting method compared to other methods, since it is able to take into account and use information about both previous and subsequent time steps. This allows the model to more fully and accurately analyze the dependencies in the data and make more accurate predictions. The study of this method showed that it is not good enough due to its long work, so it was decided to use a neural network with layers: Word Embedding and linear. This method showed the acceptable prediction time for use in the OS.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Прикладные программы – это программы, предназначенные для выполнения определенных задач и взаимодействия с пользователями.

Операционная система (ОС) – программное обеспечение, которое управляет аппаратными ресурсами компьютера и обеспечивает работу других прикладных программ.

Процесс в Linux представляет собой запущенную программу, которая выполняется в операционной системе.

Трассировка системных вызовов – это процесс отслеживания и записи всех системных вызовов, которые выполняются программой во время ее работы.

Искусственный интеллект (ИИ) – методика, позволяющая имитировать человеческий интеллект. Это набор методов, позволяющий компьютерным программам решать сложные задачи, учиться и самосовершенствоваться.

Искусственная нейронная сеть (нейронная сеть, нейросеть) – математическая или компьютерная модель, созданная на основе функционирования биологических нейронных сетей в организме человека. Она состоит из набора связанных между собой формальных нейронов и способна обрабатывать входные данные, генерируя соответствующий выходной сигнал.

Нейронный слой представляет собой совокупность нейронов, которые выполняют определенные функции при обработке информации. Каждый слой нейронной сети обычно состоит из нескольких нейронов, которые работают вместе для обработки и передачи данных от одного слоя к другому.

Токенизация – это процесс разделения текста на токены.

Токен – это единица обработки текста для анализа и обработки языка.

Математическая лингвистика – это область науки, которая изучает методы математики и статистики для анализа текстов, генерации и понимания языка, распознавания речи, и других задач, связанных с лингвистикой.

Natural language processing (NLP) – область математической лингвистики и искусственного интеллекта, которая занимается обработкой и анализом человеческого языка.

Word embedding – это процесс преобразования текстовой информации в числовой формат (числовые векторы в многомерном пространстве), который помогает компьютеру более эффективно понимать и обрабатывать естественный язык.

Рекуррентные нейронные сети (RNN)– это вид нейронных сетей, которые могут обрабатывать последовательности данных переменной длины за счет своей циклической структуры. Они способны запоминать предыдущие состояния и применять эту информацию для принятия решений на основе текущего входа.

Датасет (выборка) – набор данных.

Период времени, в течение которого нейронная сеть обучается на конкретных данных, обновляя свои веса и повышая точность, называется эпохой.

ЯП – язык программирования.

ВВЕДЕНИЕ

Современные операционные системы сталкиваются с растущим объемом данных и сложностью работы приложений, что может привести к снижению производительности и увеличению времени отклика системы. Предсказание поведения программ позволяет оптимизировать производительность операционной системы и приложений, путем уменьшения загрузки CPU, улучшения использования памяти и уменьшения времени ожидания при выполнении системных вызовов [1], [2], [5].

Целью работы является разработка модели искусственного интеллекта, которая может прогнозировать действия программ в реальном времени, способствуя улучшению производительности операционных систем.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- Реализовать трассировку списка процессов, чтобы получать список последовательных событий (системных вызовов).
- Обработать список системных вызовов и разделить полученные данные на обучающую, валидационную и тестовую выборки, тем самым подготовив данные для обучения модели ИИ.
- Выбрать подходящую модель искусственного интеллекта для прогнозирования системных вызовов.
- Провести обучение модели на обучающем наборе данных.
- Оценить качество модели ИИ на тестовой выборке.

Модель прогнозирования системных вызовов в ОС Linux является объектом исследования.

Предметом исследования является точность и время предсказания системных вызовов.

Разработка модели искусственного интеллекта для прогнозирования поведения программ в операционных системах имеет практическую значимость,

поскольку она может найти применение в различных областях, таких как кибербезопасность, оптимизация работы программ и улучшение эффективности IT-инфраструктуры.

1 Обзор предметной области

1.1 Роль системных вызовов в ОС

Операционная система Linux одна из наиболее широко используемых операционных систем [7], относится к семейству Unix-подобных операционных систем с открытым исходным текстом программы, основанных на ядре Linux. Система состоит из следующих основных компонентов:

1. Ядро Linux - основа операционной системы, управляющая работой аппаратного обеспечения, обеспечивающая основные функции операционной системы и отвечающая за выполнение прикладных программ.

2. Командная оболочка (shell) - интерфейс командной строки, с помощью которого пользователи взаимодействуют с операционной системой.

3. Системные утилиты — это набор базовых программ, который обеспечивает выполнение основных функций операционной системы (управление файлами, сетевое взаимодействие, администрирование системы и другое).

4. Графическое окружение - набор программ и библиотек, обеспечивающих графический интерфейс пользователя (GUI) для удобства работы с операционной системой.

5. Драйверы устройств - программное обеспечение, обеспечивающее взаимодействие операционной системы с аппаратным обеспечением компьютера.

Для обеспечения стабильной и эффективной работы операционной системы Linux эти компоненты работают вместе.

Операционная система обладает двумя режимами работы: пользовательский режим и режим ядра.

Пользовательский режим - это режим работы, в котором выполняются пользовательские процессы или пользовательские программы. В этом режиме процессы имеют ограниченный доступ к ресурсам компьютера и не имеют прямого доступа к аппаратному обеспечению. Пользовательский режим обеспечивает изоляцию процессов друг от друга и обеспечивает безопасность системы.

Режим ядра - это режим работы, в котором выполняется только ядро операционной системы. В данном режиме ядро операционной системы имеет полный контроль над ресурсами компьютера и осуществляет их управление. Ядро операционной системы управляет обработкой прерываний, переключением контекста процессов, управлением памятью и другими низкоуровневыми операциями.

Пользовательский режим взаимодействует с режимом ядра для обеспечения работы операционной системы. Ядро операционной системы предоставляет интерфейс для взаимодействия между пользовательскими программами и аппаратным обеспечением компьютера, обеспечивает безопасность и стабильность работы системы.

Пространство пользователя может взаимодействовать с пространством ядра посредством системных вызовов.

Системный вызов – это функция операционной системы, которая позволяет приложению или пользовательскому процессу взаимодействовать с ядром операционной системы и получать доступ к различным ресурсам и функциям системы. Системные вызовы обеспечивают интерфейс для работы с аппаратным обеспечением компьютера через операционную систему.

Состояние процесса в конкретный момент времени называется «контекстом».

Переключение контекста(context switch) – это процесс, при котором операционная система переключает выполнение процессора с выполнения одного процесса на другой. При переключение контекста выполнение текущего процесса останавливается и его контекст сохраняется, чтобы когда процесс будет возобновлен, он продолжил работу с того же момента, на котором был прерван.

Стоимость контекстного переключения (context switch) зависит от множества факторов и считается затратной операцией [8], так как требует сохранения и загрузки состояния процесса, включая значения регистров, указатели

стека, таблицы страниц памяти и другие данные. Уменьшение частоты переключения контекста является ключевым аспектом повышения производительности системы.

Предсказание следующего системного вызова способствует уменьшению частоты переключения контекста и повышению производительности программы. Когда предсказание делается точно, операционная система может заранее загружать данные в память и подготавливать необходимые ресурсы, что позволяет уменьшить время ожидания выполнения операций и ускорить работу программы.

1.2 Существующие модели предсказания

1.2.1 Предиктор последнего значения

Предиктор последнего значения [1] предсказывает, что значение текущего экземпляра инструкции будет таким же, как значение, полученное предыдущим экземпляром. Этот предиктор применяет функцию идентификации к ранее наблюдаемому значению и использует его в качестве следующего прогноза. Этот метод довольно простой и полезен, когда в данных наблюдается устойчивая тенденция и незначительная изменчивость.

1.2.2 Предиктор на основе таблиц

Программы при выполнении демонстрируют различную степень периодичности, что указывает на то, что модели поведения повторяются с течением времени. Предикторы на основе таблиц [1] разработаны на использовании этой повторяемости. Они используют историю прошлого поведения в качестве индекса в таблице прогнозирования. Предсказание, сохраненное в таблице является значением, которое соответствовало истории поведения в прошлом.

При выполнении новой рабочей нагрузки предикторы проходят начальную фазу обучения, чтобы заполнить таблицу прогнозов историческими шаблонами. Для получения прогнозов на этапе обучения может использоваться

схема Предиктор последнего значения. Предикторы адаптируются к изменениям в изученном поведении, обновляя записи в таблице, но необходимо соблюдать осторожность, чтобы избежать обновления таблицы ложными значениями из-за шумовых пиков. В дополнение к прогнозируемому значению, в каждой записи таблицы хранится последнее фактическое значение, которое наблюдалось в истории. После получения нового фактического значения, таблица обновляется путем установки на наиболее частое из трех значений: прогнозируемого и последнего значения, сохраненные в таблице, и нового фактического значения.

1.2.3 Унифицированный метод прогнозирования (UPM)

UPM – это универсальный метод прогнозирования, который не зависит от системы и компьютерных показателей [2]. В основе метода лежит параметрическая модель, основанная на статистической базе.

Применение UPM состоит из нескольких этапов.

Первым шагом для применения UPM является выбор параметрической модели (например, авторегрессионная, марковская). Наиболее часто применяется авторегрессионная модель, которая требуют очень мало информации о статистике входных данных и имеет простую реализацию.

Авторегрессионная модель зависит только от предыдущих выходных данных системы и позволяет находить коэффициенты линейных предикторов, которые предсказывают текущее значение на основе прошлых образцов, причем предполагается, что параметры модели постоянны на протяжении всего периода анализа.

Вторым шагом является определение целевой функции (например, функция среднеквадратичной ошибки, функция накопленной квадратичной ошибки, вероятность). Целевая функция, как правило, является функцией параметров модели и прошлых наблюдаемых выборок.

Наиболее часто используемой целевой функцией является функция среднеквадратичной ошибки, которая имеет множество преимуществ, в том числе следующее:

- Минимизация целевой функции приводит к линейной системе уравнений, которую легко решить без значительных затрат. Любой другой выбор целевой функции может привести к нелинейной системе уравнений, которая может быть решена с помощью численной оптимизации. Решение нелинейной системы уравнений может быть дорогостоящим без гарантий сходимости.
- Целевая функция предпочитает множество небольших ошибок нескольким большим, которые линейно складываются в одно и то же значение, что обычно желательно для минимизации влияния превышений в прогнозировании.
- Функция среднеквадратичной ошибки монотонно растет со стандартным отклонением, поэтому минимизация квадрата ошибки также сводит к минимуму другие статистические показатели, которые обычно используются.

Третьим шагом является оценка параметров модели путем минимизации выбранной целевой функции. Она может быть минимизирована либо аналитически, либо с помощью численной оптимизации. Форма решения определяется выбранной целевой функцией и типом модели.

1.2.4 Предиктор на основе дерева решений

Предиктор на основе дерева решений используются методы машинного обучения (дерево решений) построенное при помощи алгоритма C4.5. Для построения датасета используется набор определенных статических и динамических атрибутов процессов во время их выполнения. В работе [3] были проведены исследования по выявлению наиболее важных статических и динамических характеристик процессов, которые могут наилучшим образом помочь в

прогнозировании времени загрузки процессора и минимизировать время выполнения процесса. Результатом работы стал набор атрибутов для построения датасета, представленный в таблице 1, который был получен с помощью команд readelf и size.

Таблица 1 – Набор атрибутов для построения датасета для предиктора на основе дерева решений

Атрибут	Его ранг	Обозначение
InputSize	1	Значение размера входных данных, зависящее от типа программы
ProgramName	2	Название программы и номинальный атрибут
BSS	3	Информация о размере неинициализированных данных
RoData	4	Размер данных (байт), доступный только для чтения, который обычно используется для создания недоступного для записи сегмента в образе процесса
Text	5	Текст или исполняемые инструкции
InputType	6	Тип входного номинального значения

Обучение осуществляется путем анализа статических и динамических атрибутов процессов, представленных в таблице.

1.2.5 Предсказание на основе Sequence-to-Sequence модели

В своей архитектуре Sequence-to-Sequence модель [4] использует рекуррентные нейронные сети (RNN) для обработки последовательностей входных и выходных данных. В этой работе для улучшения производительности базовых классификаторов обнаружения вторжений авторы объединили предсказанную последовательность с вызванной последовательностью, чтобы сформировать расширенные входные данные. Как показано на рис. 1, предсказан-

ная последовательность предоставляет классификаторам дополнительную информацию для улучшения возможности определения характера последовательности системных вызовов.

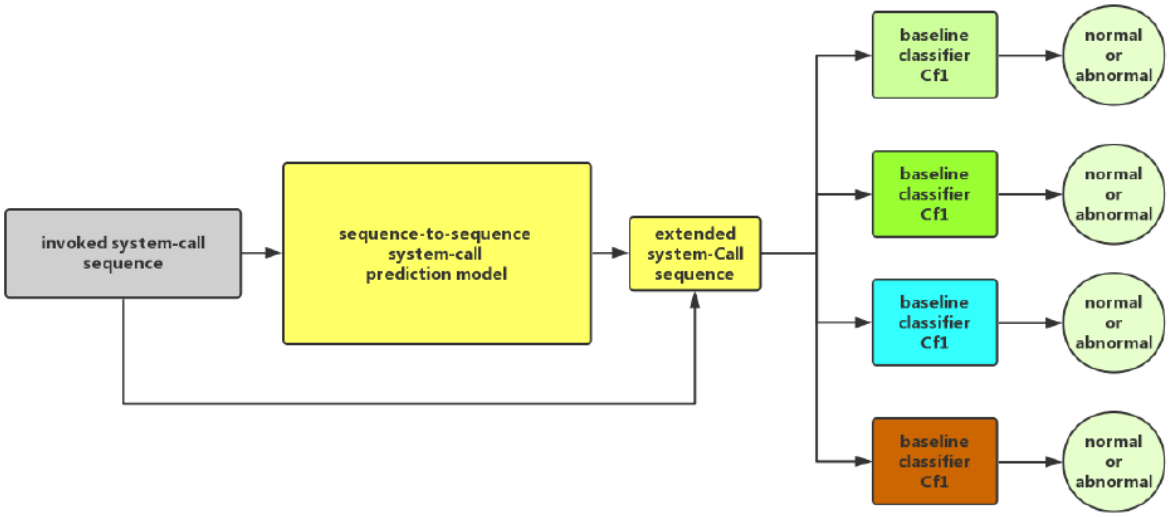

Рисунок 1 – Схема работы Sequence-to-Sequence модели

Авторы рассматривают трассировки системных вызовов, создаваемые во время работы программы, как набор языковых предложений, которые процесс использует для взаимодействия с операционной системой. Для генерации семантически обоснованных последовательностей системных вызовов внедряется структура sequence-to-sequence языковой модели RNN с «механизмом внимания» (attention) (кодер выполняет операцию прямого распространения). Механизм attention позволяет подчеркнуть влияние некоторых определенных системных вызовов, которые лучше отражают намерения программы, на конкретное декодирование системных вызовов. Таким образом данный метод использует информацию о входных данных, которые наиболее важны при декодировании каждого системного вызова.

Архитектура модели sequence-to-sequence с attention показана на рис. 2.

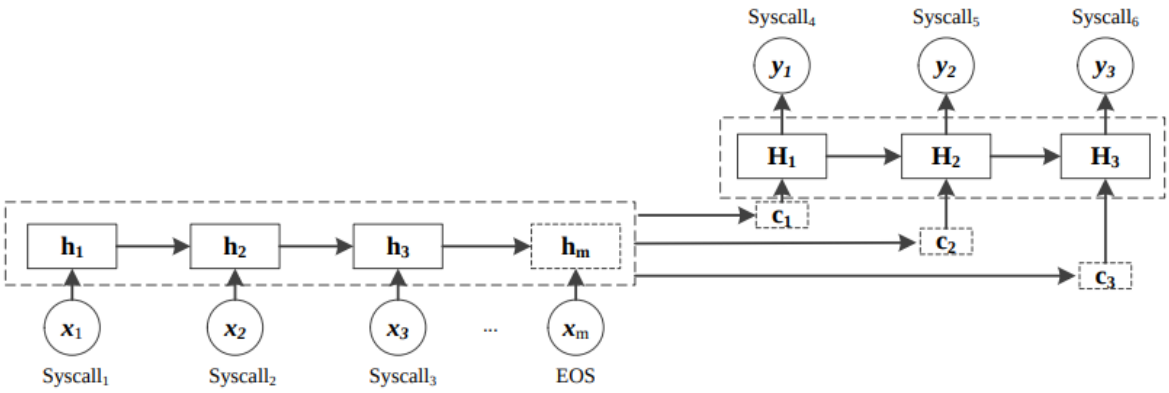

Рисунок 2 – Схема работы Sequence-to-Sequence модели с attention

Где h_i – i-ое скрытое состояние. $i \in (1,2,\dots,m)$, а c_i обозначает i-й информационный вектор контекста, который вычисляется с помощью алгоритма attention.

Предложенная прогностическая модель предсказывает, какие системные вызовы будут выполнены впоследствии, что позволит наблюдать состояние системы и прогнозировать атаки.

1.3 Критерии сравнения аналогов

1.3.1 Точность прогнозирования

Точность прогнозирования системных вызовов зависит от нескольких факторов, включая используемые алгоритмы прогнозирования, объем доступных данных и качество обучающей выборки. Чтобы оценить точность прогнозирования, данные разделяют на обучающий и тестовый наборы. Одним из распространенных методов оценки точности прогнозирования является процент правильно предсказанных системных вызовов относительно общего числа вызовов.

1.3.2 Лицензирование ОС

Лицензирование операционной системы — это процесс получения разрешения на использование операционной системы. Существуют два основных типа лицензий: проприетарные и открытые. Проприетарные лицензии обычно

требуют оплаты и предоставляют пользователю ограниченные права на использование программного обеспечения. Открытые лицензии, такие как лицензии GNU General Public License (GPL), предоставляют пользователю более широкие права, включая право на изменение и распространение программного обеспечения.

Сравнительная характеристика аналогов представлена в таблице 2:

Таблица 2 – Сравнение аналогов

Критерий/Аналог	Точность	Лицензирование ОС
Предиктор последнего значения	88%	Коммерческое
Предиктор на основе таблиц	92%	Коммерческое
Унифицированный метод прогнозирования	94%	Коммерческое
Предиктор на основе дерева решений	94%	Коммерческое/Свободное
Предсказание на основе Sequence-to-Sequence модели	93%	Коммерческое/Свободное

1.3.3 Выводы по итогам сравнений

Предиктор последнего значения обладает наименьшей точностью среди остальных методов. Предиктор на основе таблиц, унифицированный метод, предиктор на основе дерева решений, предсказание на основе Sequence-to-Sequence модели обладают практически одинаковой точность. Предиктор последнего значения, предиктор на основе таблиц, унифицированный метод прогнозирования используются в архитектуре IBM Power и ОС AIX, которая является коммерческой проприетарной операционной системой, разработанной компанией IBM для серверов, имеет высокую производительность, надежность и поддержку различных аппаратных компонентов. Предиктор на основе дерева решений, предсказание на основе Sequence-to-Sequence модели используются в архитектуре x86 и ОС Linux, которая является свободно распространяемой операционной системой с открытым исходным текстом программы и

может быть использована на различных типах компьютеров или серверов, имеет большое сообщество разработчиков и пользователей, что приводит к быстрому внедрению новых технологий и исправлению ошибок.

2 Формулировка требований к решению и постановка задачи

2.1 Постановка задачи

Необходимо собрать набор данных системных вызовов, разработать модель искусственного интеллекта для прогнозирования следующего системного вызова по пяти вызовам и оценить время предсказания.

2.2 Требования к решению

На основе сравнения аналогов решение должно удовлетворять следующим условиям:

1. Повышенной точностью прогнозирования.
2. Обучение модели на датасете системных вызовов, содержащих информацию о контексте, включая указание на используемое приложение и выполняемое действие.
3. Обеспечение более точной работы обученной нейросети за счет большого количества тестов.

2.3 Обоснование требований к решению

1. Модель ИИ должна обладать повышенной точностью прогнозирования, не меньше, чем на существующих решениях, чтобы успешно предсказывать следующие системные вызовы.
2. Обучение модели должно происходить на данных системных вызовов, содержащих информацию о контексте, включая указание на используемое приложение и выполняемое действие для более точного предсказания.
3. Модель должна быть обучена на большом количестве тестов для охвата каких-нибудь уникальных тестов для улучшения точности предсказания системных вызовов.

2.4 Обоснование выбора метода решения

Для выполнения условия повышенной точности выбрана нейронная сеть со слоями: Word Embedding, Bidirectional LSTM и линейного, из-за способности работать с последовательными данных и извлекать сложные нелинейные зависимости между входными и выходными данными. Word Embedding слой позволяет преобразовать текстовую информацию в векторное представление, что значительно упрощает работу с данными в формате текста. Bidirectional LSTM слой обеспечивает обработку входной последовательной информации в обоих направлениях, что позволяет лучше анализировать контекст и взаимосвязи между различными словами в предложениях. Линейный слой в конце нейронной сети позволяет сделать финальные предсказания на основе извлеченных признаков и связей между данными.

3 Проектирование модели ИИ

3.1 Подготовка данных

3.1.1 Сценарий собранных данных

Многие люди в наше время ежедневно пользуются компьютером, выполняя различные задачи - от работы и обучения до развлечений и общения. Именно поэтому для обучения нейросетей необходимо иметь данные о системных вызовах, которые часто вызываются во время использования компьютера пользователем.

Чтобы улучшить точность прогнозирования, следует предсказывать название следующего системного вызова на основе анализа нескольких предыдущих системных вызовов.

Для более точного прогнозирования системных вызовов данные должны обладать достаточно большим контекстом и собраны из различных действий пользователя в компьютере (сценариев использования). Контекстом системных вызовов является: название процесса, который вызывает системный вызов, название системного вызова, его аргументы и его результат выполнения.

Наиболее популярные действия пользователя использующего ОС Linux:

- Использование интернет браузеров: посещение сайтов, чтение новостей, просмотр видео и т. д.
- Использование социальных сетей: чаты, просмотр ленты новостей.
- Открытие и использование различных приложений.
- Использование терминала

3.1.2 Токенизация

Разделение текста на токены является одним из первых и важных этапов при обработке текстов и для многих методов анализа естественного языка.

В процессе токенизации слов текст разделяется на отдельные слова, символы или фразы в зависимости от конкретной задачи. Токенизация слов может включать в себя удаление знаков препинания, приведение всех слов к нижнему регистру, удаление стоп-слов.

Последовательность из системных вызовов объединяется в предложение, которое следует обработать для нейронной сети, так же необходимо обработать результат нейронной сети (название системного вызова).

В данной задаче токенами будут считаться:

- Название процесса
- Название системного вызова
- Полное перечисление аргументов
- Результат системного вызова

Пример разбиения данных на токены представлен на рис. 3:

Рисунок 3 – Пример разбиения данных на токены

3.1.3 Word embedding

Для успешного обучения нейронной сети необходимо преобразовать токены в вектора чисел, это можно осуществить при помощи word embedding. Вектора получившиеся после word embedding обладают следующими свойствами [22]:

- Семантическая близость. Слова с похожим значением имеют близкие векторы в пространстве, что позволяет измерять семантическую близость между словами.
- Арифметические операции над векторами слов позволяют находить аналогии.

Данные свойства могут быть полезны из-за существования системных вызовов, которые выполняют задачу с похожим смыслом, но при этом имеют разные названия.

Пример работы word embedding представлен на рис. 4, где в левой части изображены токены, а в правой – полученные вектора:

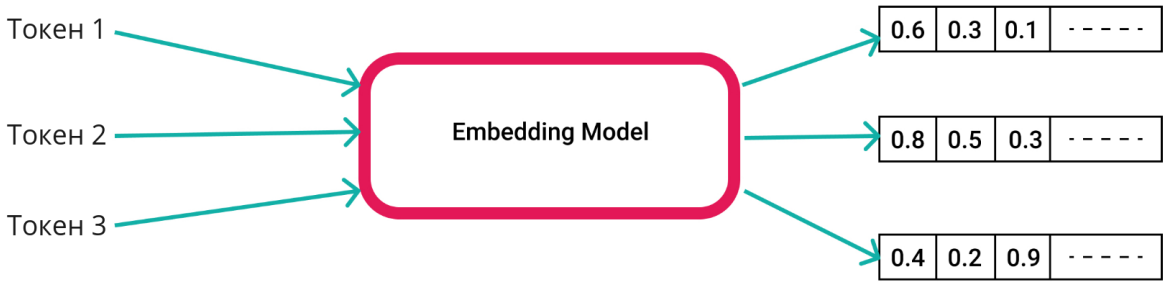

Рисунок 4 – Пример работы word embedding

3.2 Архитектура и принцип работы Нейронной сети

3.2.1 LSTM

LSTM (Long Short-Term Memory) [9], [10], [11], [12] – это разновидность архитектуры рекуррентных нейронных сетей, которая способна запоминать информацию на длительные промежутки времени. LSTM состоит из специальных блоков памяти, которые позволяют сети запоминать и забывать информацию в зависимости от контекста.

Затухания градиентов – это явление, которое происходит при обучении нейронной сети, когда значения градиентов становятся очень маленькими по мере распространения через слои сети, что приводит к незначительным обновлениям весов, и обучение сети замедляется или останавливается.

Архитектура LSTM представлена на рис. 5.

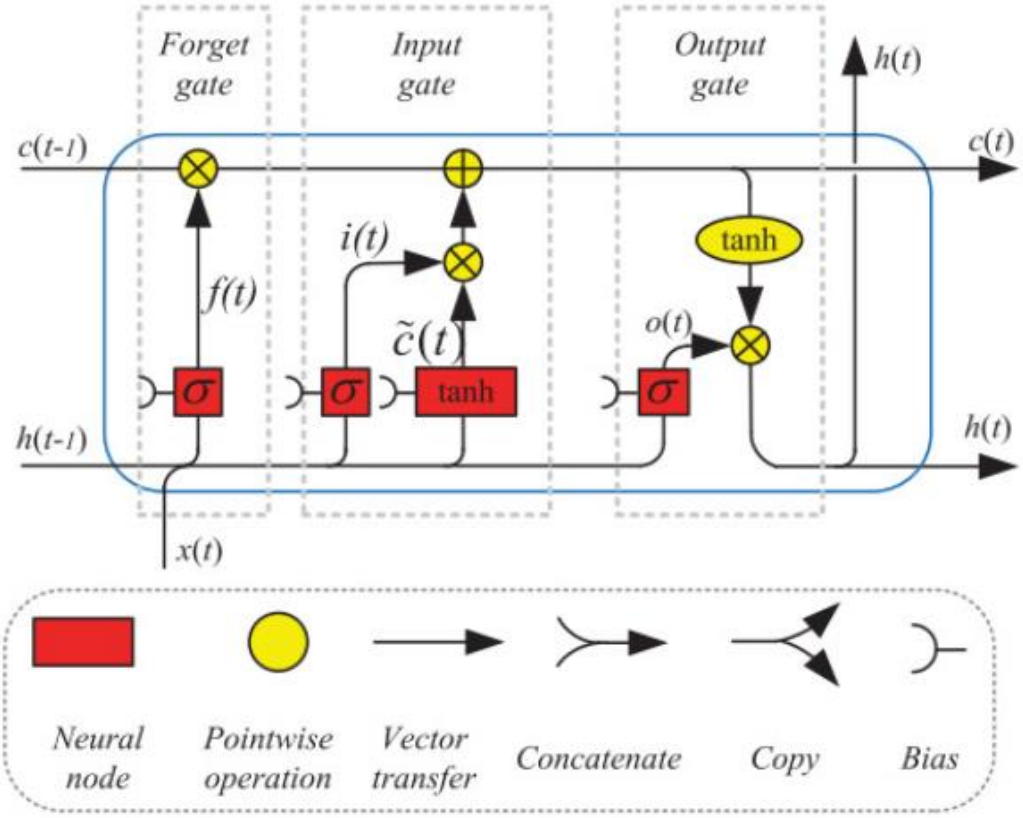

Рисунок 5 – Архитектура LSTM

Neural node – это обученный слой нейронной сети, pointwise operation – поточечная операция, vector transfer – векторный перенос (перемещение вектора с выхода одного узла на вход другого); concatenate – объединение; copy – копирование (информация копируется и отправляется в различные узлы сети), bias – добавление смещения.

В блоке памяти LSTM существует несколько основных «вентилей»:

1. Forget gate определяет, какую информацию можно забыть из предыдущей ячейки памяти. Помогает модели избегать проблемы затухания градиентов и сохранять только актуальную информацию.
2. Input gate решает, какую информацию следует добавить в ячейку памяти на основе нового входного сигнала. Позволяет модели обновлять свою память с учетом новых данных.
3. Output gate определяет, какую информацию из ячейки памяти следует передать на выход.

Сеть с двунаправленной LSTM (Bidirectional LSTM Network) [13], [14] – тип рекуррентной нейронной сети, который использует LSTM блоки и обрабатывает входные данные как в прямом, так и в обратном направлении. Этот подход позволяет учитывать как предыдущий, так и последующий контекст для более точного прогнозирования выходных данных.

Структура Bidirectional LSTM представлена на рис. 6.

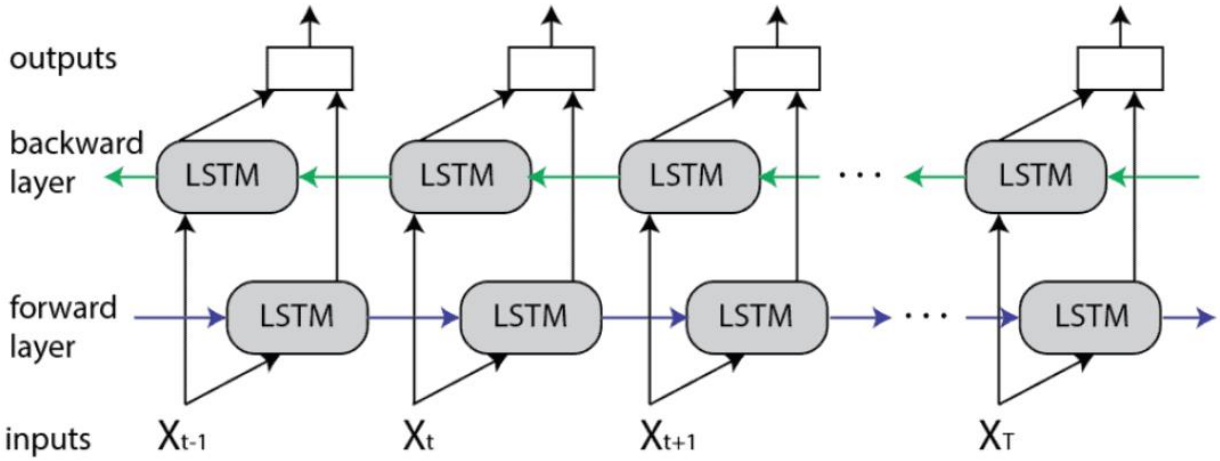

Рисунок 6 – Структура Bidirectional LSTM

Forward layer (прямой слой) работает в прямом направлении, начиная с первого элемента последовательности и последовательно обрабатывает следующие элементы до последнего.

Backward layer (обратный слой) работает в обратном направлении, начиная с последнего элемента последовательности и последовательно обрабатывает предыдущие элементы до первого.

Данный метод позволяет модели лучше прогнозировать следующий элемент на основе предыдущего и будущего контекста.

LSTM слои широко применяются для анализа и прогнозирования последовательных данных в различных областях науки и техники благодаря их способности эффективно работать с долгосрочными зависимостями в данных [15].

3.2.2 Линейный слой

Линейный слой нейросети представляет собой слой, в котором каждый нейрон соединен с каждым нейроном предыдущего слоя, и каждая связь между нейронами имеет свой вес, который определяет влияние сигнала на следующий нейрон.

Линейный слой математически можно представить следующим образом (см. рис. 7), где y – вектор выходных данных, x – вектор входных данных, W – матрица весов (wji – вес от j-го нейрона к i-ому нейрону), b – вектор сдвига [16].

$$\begin{pmatrix} y_1 \\ \vdots \\ y_K \end{pmatrix} = \begin{pmatrix} w_{10} & w_{11} & \dots & w_{1D} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K0} & w_{K1} & \dots & w_{KD} \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_D \end{pmatrix},$$

$$\mathbf{y} = \mathbf{W}\dot{\mathbf{x}},$$

Рисунок 7 – Математическое представление линейного слоя

Линейный слой нейронной сети представлен на рис. 8:

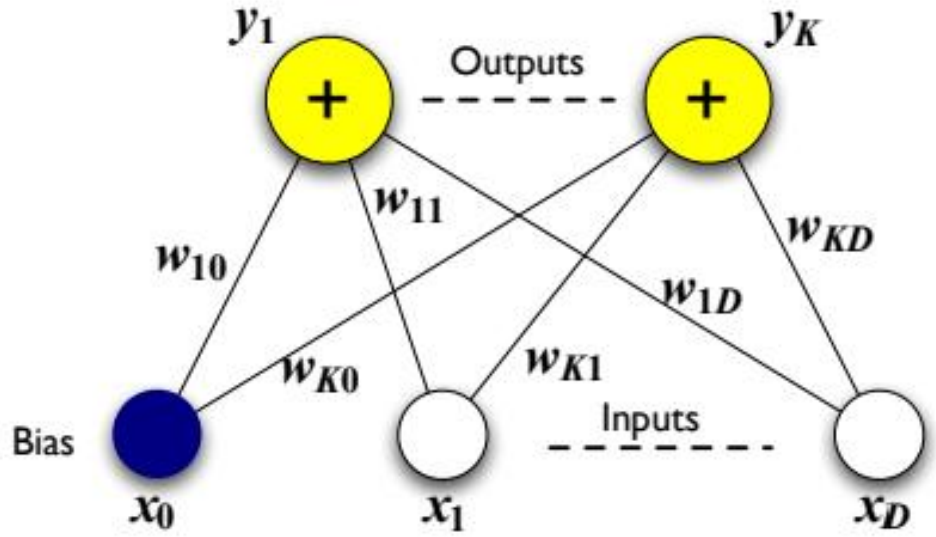

Рисунок 8 – Линейный слой нейронной сети

3.2.3 Спроектированная архитектура нейросети

Текстовые данные системных вызовов токенизируются, токенами являются: название процесса, название системного вызова, полное перечисление аргументов, результат системного вызова. Далее токены передаются на вход нейронной сети.

Итоговый вид спроектированной нейронной сети состоит из следующих слоев: слой word embedding, Bidirectional LSTM слой и линейный слой.

Для преобразования текстовых данных, содержащих информацию о системных вызовах, в векторное представление был использован слой word embedding. Такой подход позволяет присвоить каждому уникальному слову или символу числовой вектор, что улучшает работу нейронной сети и повышает ее эффективность. Векторы токенов подаются на вход Bidirectional LSTM слою.

Слой Bidirectional LSTM представляет собой двунаправленную рекуррентную нейронную сеть с долгой краткосрочной памятью, которая способна анализировать последовательность слов как в прямом, так и в обратном порядке. Это позволяет модели учитывать как предшествующую, так и последующую информацию при обработке данных. LSTM слой способен запоминать предыдущие состояния и принимать решения на основе этой информации.

Линейный слой был добавлен в конце нейронной сети для финального прогноза поведении программ в операционной системе. Выходов у линейного слоя будет столько, сколько названий системных вызовов встретилось в данных.

Схема спроектированной нейронной сети представлена на рис. 9:

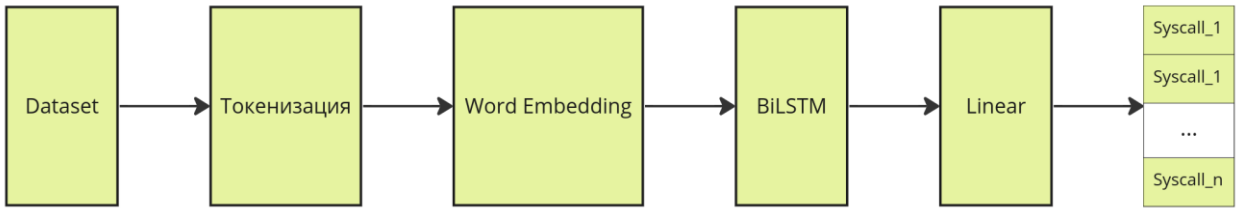

Рисунок 9 – Схема спроектированной нейронной сети

Таким образом, токенизация данных и комбинация слоев word embedding, LSTM и линейного слоя в нейронной сети позволяют эффективно обрабатывать данные и прогнозировать следующие системные вызовы в операционной системе на основе анализа текстовой информации.

4 Реализация модели ИИ

4.1 Датасет

4.1.1 Сбор датасета

Набор данных формируется с помощью утилиты трассировки: strace. Strace - один из самых популярных инструментов для трассировки системных вызовов. Он позволяет отслеживать все системные вызовы, производимые программой, и выводить подробную информацию о них, такую как тип вызова, аргументы, возвращаемое значение и время выполнения. [17]

Была разработана программа, которая устанавливает идентификаторы запущенных процессов в операционной системе, после чего запускает трассировку для каждого процесса с целью отслеживания системных вызовов. Трассировка выполняется с помощью утилиты strace -ff -p pid, где pid – это уникальный идентификатор запущенного процесса, ключ -ff для каждого дочернего процесса создает отдельный файл с именем «имя_файла.pid», -p трассирует процесс с заданным pid. Таким образом, команда strace -ff -p будет отслеживать системные вызовы и сигналы нескольких процессов с указанными pid. Для того, чтобы в конечном датасете присутствовало название команды, которые запускали выполнение системных вызовов, название команды добавляется в названия созданных файлов, т. к. в названии команды присутствуют запрещенные символы для названия файла, то было использовано регулярное выражение, которое берет название команды до пробела, тем самым заменяя запрещенные символы.

Для прогнозирования будущих системных вызовов было решено использовать данные о 5 предыдущих вызовах. Для составления выборки было решено исключить файлы размером менее 1 килобайта, а также исключить файлы с одинаковыми вызовами, так как они не представляют сильной смысловой ценности. Была разработана программа, которая анализирует все файлы трассировки и отбирает 200 процессов для создания датасета, выбирая их по уникальности названия команды и размеру файла.

Учитывая большой размер и наличие дублирующих данных в полученном датасете, было принято решение выделить 500000 случайно выбранных элементов, оставив из них только уникальные значения. В итоговом датасете получилось 366032 элементов.

4.1.2 Dataset и dataloader

Классы Dataset и DataLoader в PyTorch используются для упрощения и ускорения загрузки и обработки данных во время обучения моделей машинного обучения [18].

Класс Dataset – класс, который предоставляет доступ к набору данных. Он позволяет загружать и обрабатывать данные.

Класс DataLoader используется для создания итератора, который позволяет эффективно загружать данные из созданного объекта класса Dataset во время обучения модели. DataLoader позволяет работать с большими объемами данных и разделять данные на батчи (небольшие подмножества данных из исходного датасета).

Для обработки уже подготовленного датасета были реализованны функции:

1. *tokenizer(syscalls)* – функция на вход получает строку *syscalls,* состоящую из системных вызовов, и токенизирует ее, возвращает список токенов.
2. *add_tokens_to_index(index, tokens_counter)* – преобразование токенов *tokens_counter*, отсортированных по встречаемости, в индексы и добавление токена с индексом в словарь *index*, возвращает словарь *index*.

Был создан класс *SyscallDataset*, наследуемый от класса Dataset. В конструкторе данного класса происходит обработка данных токенизация (функцией *tokenizer*) и преобразования токенов в коды (функция add_tokens_to_index). Так же были переопределены методы *__len__()*, который возвращает количество элементов в наборе данных, и метод *__getitem__(ix)*,

который возвращает пару (*X, y*), где *X* – тензор индексов токенов, которые были получены путем токенизации предложения, а *y* – это индекс названия системного вызова.

Также были созданы экземпляры класса Dataloader - *train_dataloader*, *val_dataloader*, *test_dataloader*, которые представляют собой данные, разделенные на обучающую, валидационную и тестовую выборки соответственно. Далее на этих объектах нейронная сеть будет обучаться, настраивать свои гиперпараметры и оценивать итоговое качество модели.

4.2 Реализация нейронной сети и ее обучение

Был реализован класс нейронной сети *SyscallBiLSTMNetwork* со следующими слоями:

- Word Embedding, где размерность embedding равна 4.
- Bidirectional LSTM, где используется 1 слой и размер скрытого состояния равен 16.
- Линейный, на вход которого подается 32 выхода за счет двунаправленности слоя LSTM, и возвращает количество выходных значений, равное количеству системных вызовов в наборе данных.

Данные подаются на вход в нейронную сеть батчами по 1024 элемента.

Архитектура получившейся нейронной сети представлена на рис. 10:

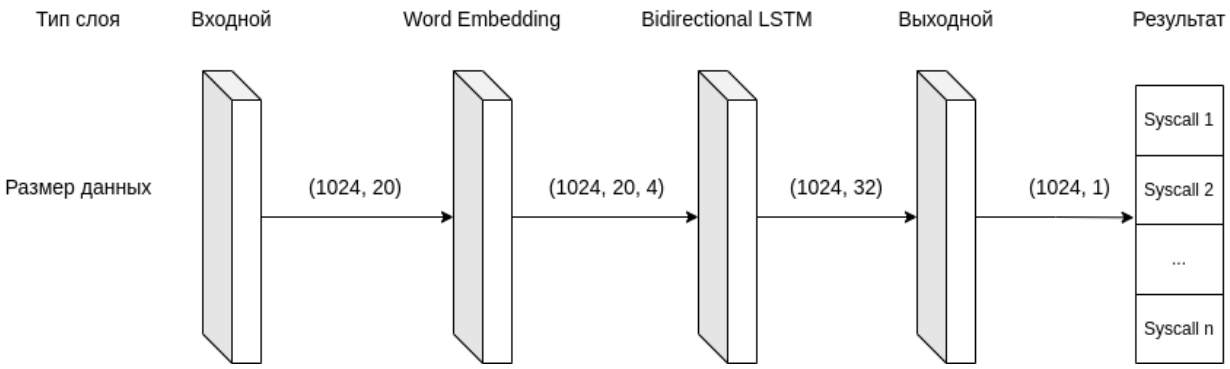

Рисунок 10 – Архитектура получившейся нейронной сети

Так же была реализована функция *compute_loss_accuracy_perfomance(loss_f, net, dataloader)*, которая вычисляет

loss, точность и среднее время, которое потребовалось для предсказания последующего вызова, функции потерь *loss_f*, нейросети *net* на данных *dataloader*.

В качестве функции потерь была выбрана кросс энтропия (Cross-entropy), т. к. она одна из самых популярных и часто используемых функций потерь для решения задачи многоклассовой классификации [19], [20], [21]. В качестве алгоритма настройки параметров нейронной сети был выбран стохастический метод оптимизации Adam. Модель обучалась на 50 эпохах на датасете разделенном на батчи размером 1024 элементов.

Результаты обучения представлены на рис. 11, рис. 12, рис. 13:

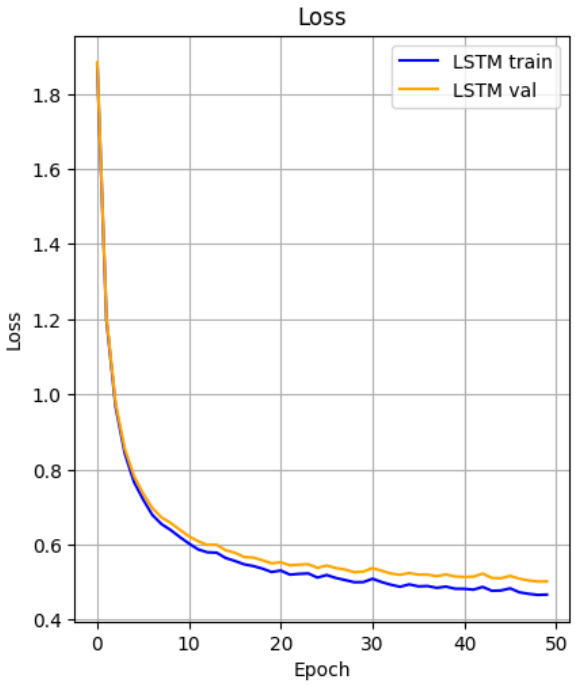

Рисунок 11 – График значений функции потерь от номера эпохи спроектированной нейронной сети

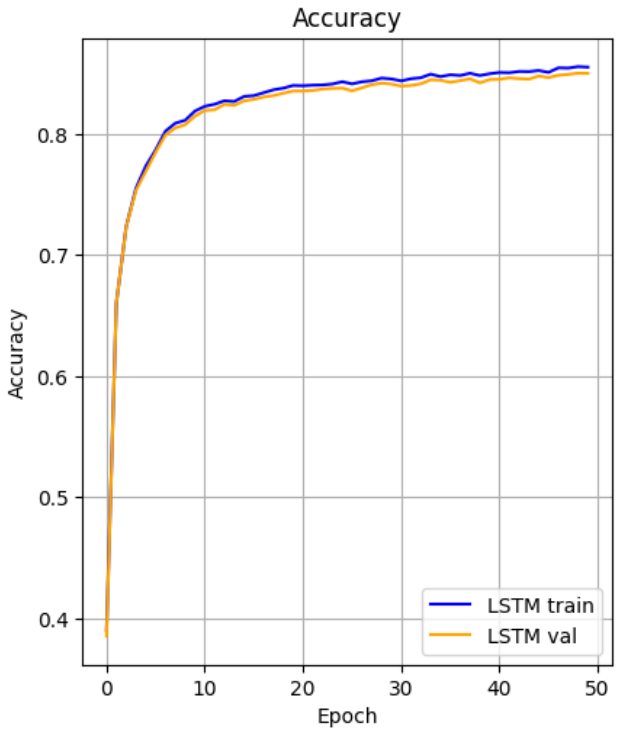

Рисунок 12 – График точности от номера эпохи спроектированной нейронной сети

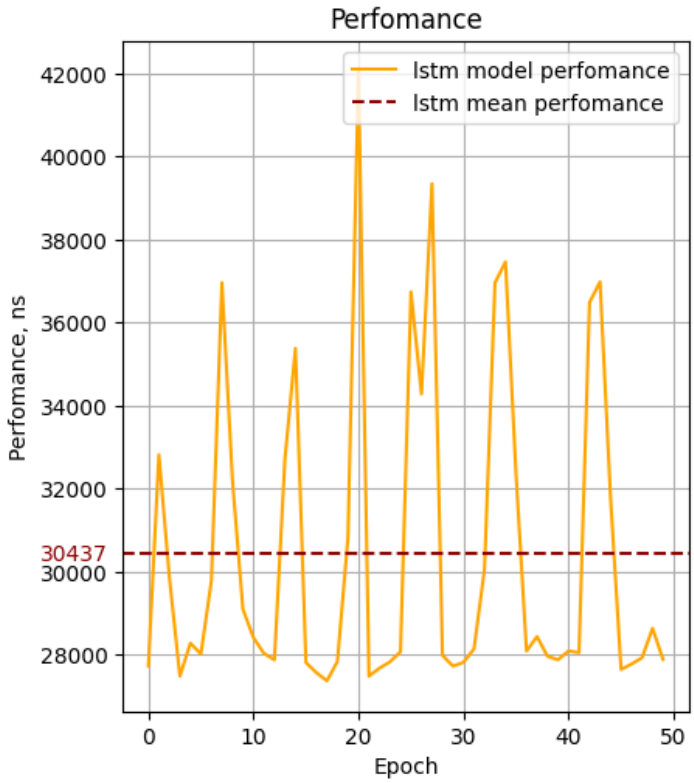

Рисунок 13 – График производительности от номера эпохи спроектированной нейронной сети

На тестовой выборке точность равна 84%, а среднее время предсказания равно 30437. Данная модель обладает достаточно хорошей точностью (больше 80%), но использование сложной модели, включающей слой word embedding, Bidirectional LSTM и линейный слой, для прогнозирования системных вызовов неэффективно из-за высокой вычислительной нагрузки. Время, требуемое на предсказание такой моделью (около 30 микросекунд), не соответствует скорости выполнения задач операционных систем. Чтобы быть эффективными в ОС, модели должны быть более простыми и быстрыми, способными быстро и эффективно предсказывать системные вызовы. Поэтому было принято решение использовать слой word embedding и линейный слой для предсказания.

Итоговая нейронная сеть состоит из следующих слоев:

- Word Embedding, где размерность токена равна 4.
- Линейный, на вход которого подается 5 системных вызова по 4 слова, и каждое слово является вектором размерностью 4, таким образом на вход подается 80 значений, а выход остается неизменным, он равен количеству системных вызовов в наборе данных.

Архитектура итоговой нейронной сети представлена на рис. 14:

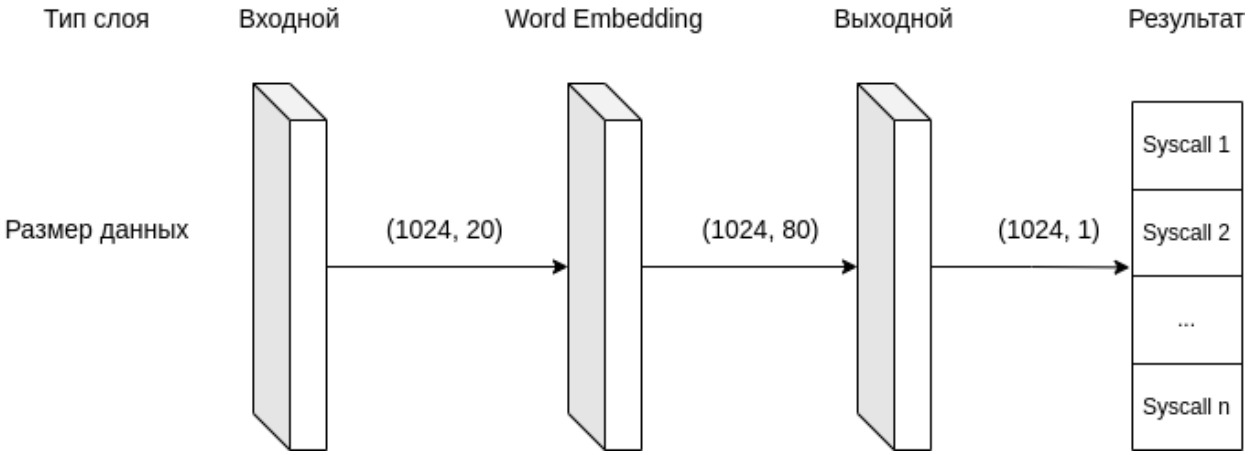

Рисунок 14 – Архитектура итоговой нейронной сети

Графики loss, точности и производительности двух нейросетей предсавлены на рис. 15, рис. 16, рис. 17:

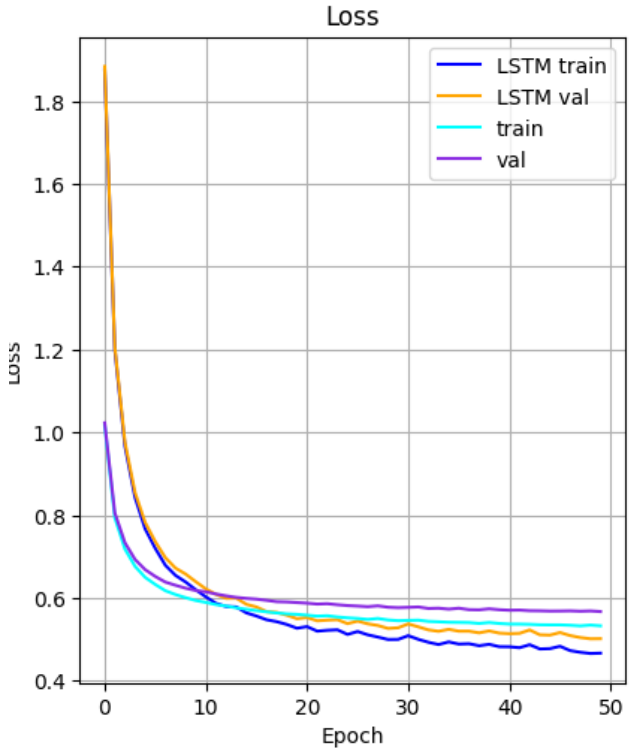

Рисунок 15 – График значений функции потерь от номера эпохи спроектированной нейронной сети

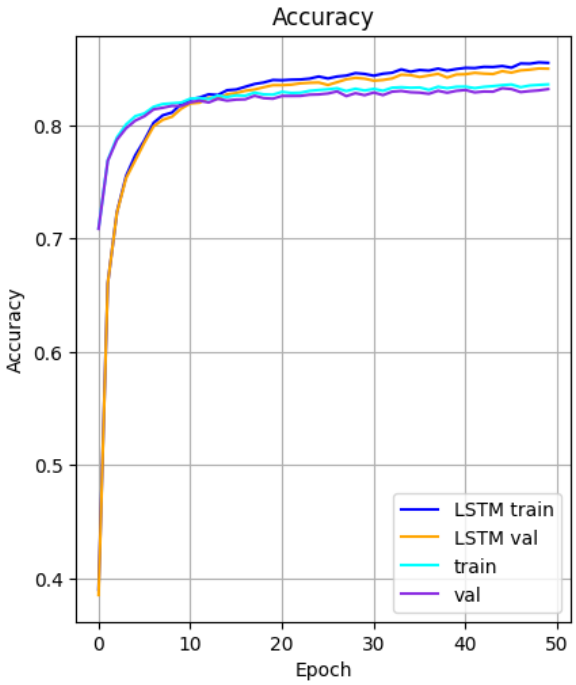

Рисунок 16 – График точности от номера эпохи спроектированной нейронной сети

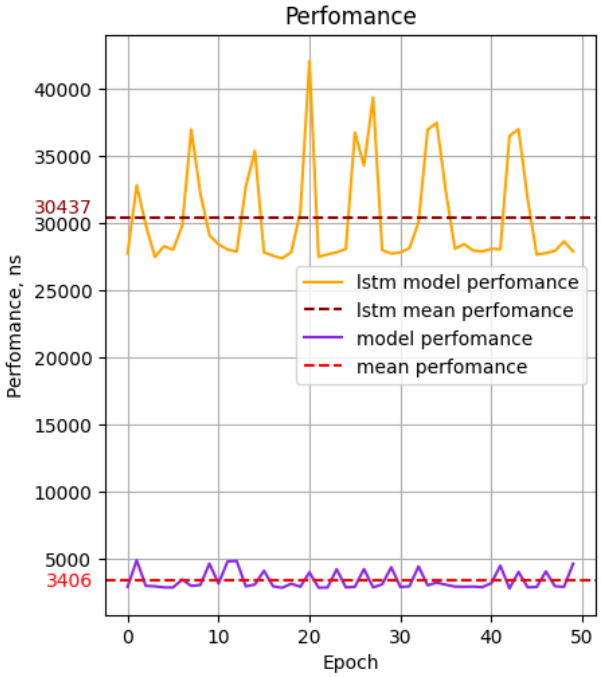

Рисунок 17 – График производительности от номера эпохи спроектированной нейронной сети

Исходя из полученных графиков можно заметить, что у итоговой модели немного ниже точность по сравнению с нейронной сетью с Bidirectional LSTM, но зато время для предсказания уменьшилось примерно в 10 раз.

4.3 Используемые технологии

Для создания и обучения нейронных сетей доступны различные инструменты и библиотеки, и одним из наиболее популярных языков программирования для этой цели является Python [6]. Python предлагает широкий спектр библиотек, таких как TensorFlow, PyTorch и Keras, которые предоставляют необходимые инструменты и функциональность для разработки и обучения нейронных сетей. Благодаря этому разнообразию средств Python является предпочтительным языком программирования для разработки и обучения нейронных сетей.

Для разработки нейронных сетей было решено использовать библиотеку PyTorch из-за её простого и гибкого интерфейса для создания и обучения сложных моделей. PyTorch обладает хорошей производительностью благодаря поддержке вычислений на GPU и различных алгоритмов оптимизации,

что делает её отличным выбором для исследований в области машинного обучения. Кроме того, наличие обширного сообщества пользователей обеспечивает доступ к дополнительным библиотекам и инструментам для улучшения работы с нейронными сетями [24].

Была выбрана онлайн среда разработки Google colab по следующим причинам:

- Цена: Google Colab предоставляет бесплатные вычислительные ресурсы.
- Гибкость: Запуск текста программы, не требуя установки дополнительного ПО, на ЯП Python прямо в браузере. Это делает его удобным для использования на любом устройстве.
- Облачные вычисления: Google colab работает на серверах Google, что позволяет выполнять вычисления на мощных вычислительных машинах.

Strace является широко используемым и активно поддерживаемым сообществом разработчиков инструментом для трассировки системных вызовов в операционной системе Linux. Он предоставляет простой и удобный интерфейс для выполнения трассировки, а также широкий набор функций и опций, которые позволяют пользователю настраивать и управлять процессом трассировки в соответствии с его потребностями.

5 БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

С появлением новых технологий и развитием науки все большее значение приобретает вопрос обеспечения безопасности людей при выполнении ими своих профессиональных обязанностей. Именно поэтому была создана и постоянно совершенствуется наука, изучающая проблемы безопасности труда и жизнедеятельности человека.

Безопасность жизнедеятельности (БЖД) - это набор мероприятий, направленных на защиту человека в его обычной среде обитания, поддержание его здоровья и разработку методов защиты от вредных и опасных факторов. Целью БЖД является минимизация рисков возникновения травм, несчастных случаев и болезней на рабочем месте или в любой другой сфере человеческой деятельности.

В современное время компьютерная техника широко используется в различных сферах человеческой деятельности. При работе с ЭВМ человек подвергается воздействию опасных и вредных производственных факторов [23], таких как электромагнитные поля, инфракрасное и ионизирующее излучения, шум, вибрация, статическое электричество. Работа за компьютером может быть связана с серьезным умственным и эмоциональным стрессом, а также с высокой нагрузкой на органы зрения. Это требует особого внимания к вопросам здоровья и безопасности при работе с электронной вычислительной машиной.

Для устранения негативных последствий работы с компьютером необходимо соблюдать правильный режим труда и отдыха. В противном случае сотрудники могут столкнуться с проблемами зрения, головными болями, раздражительностью, нарушениями сна, усталостью и болезненными ощущениями в глазах, пояснице, шее и руках [23].

5.1 Требования к производственным помещениям

Согласно СанПиН 2.2.2/2.4.1340-03 [25], для помещения, где используется вычислительная техника, необходимо соблюдать следующие требования:

- Освещение должно соответствовать нормативной документации, а окна предпочтительно должны быть направлены на север и северо-восток.
- Оконные проемы должны иметь регулируемые устройства, такие как жалюзи или занавеси.
- Площадь на одно рабочее место пользователей ПЭВМ проводящего за компьютером более четырех часов в день, если компьютер снабжен жидкокристаллическим или плазменным монитором, должна быть не менее 4,5 кв. м.
- Внутренняя отделка помещений с компьютерами должна выполняться с использованием материалов с диффузным отражением и определенным коэффициентом отражения для потолка, стен и пола.
- расстояние между рабочими столами с видеомониторами, должно быть не менее 2,0 м.

Оптимальными параметрами микроклимата в помещении с компьютерами считаются:

- температура воздуха – от 19 до 21°.
- относительная влажность – от 62 до 55%.
- скорость движения воздуха – не более 0,1 м/с.

5.2 Требования к рабочему месту сотрудника

В производственных помещениях при выполнении основных или вспомогательных работ с использованием ЭВМ, согласно СанПиН 2.2.2/2.4.1340-03 [25] должны выполняться следующие требования:

- Уровни шума на рабочих местах не должны превышать предельно допустимых значений.
- Рабочие столы должны быть размещены так, чтобы мониторы были обращены боковой стороной к окнам, чтобы естественный свет освещал их слева.

- Освещенность на рабочем месте должна составлять от 300 до 500 лк, без бликов на экране. Освещенность экрана не должна превышать 300 лк.
- Коэффициент пульсации не должен превышать 5%.
- Высота рабочей поверхности стола для взрослых пользователей должна регулироваться в пределах 680-800 мм; при отсутствии такой возможности высота рабочей поверхности стола должна составлять 725 мм.
- Минимальный размер рабочей поверхности стола для ПЭВМ составляет ширину 800 мм. и глубину 800 мм.
- Ширину и глубину поверхности сиденья не менее 400 мм.
- Поверхность сиденья с закругленным передним краем.
- Регулировку высоты поверхности сиденья в пределах 400-550 мм и углам наклона вперед до 15° и назад до 5°.
- Угол наклона спинки сиденья в вертикальной плоскости в пределах ±30°.
- Сиденья должны быть оборудованы стационарными или съемными подлокотниками длиной не менее 250 мм и шириной - 50-70 мм.
- Клавиатуру следует располагать на поверхности стола на расстоянии 100-300 мм от края, обращенного к пользователю.
- Экран видеомонитора должен находиться от глаз пользователя на расстоянии 600-700 мм, но не ближе 500 мм с учетом размеров алфавитно-цифровых знаков и символов.

5.3 Эргономика интерактивной системы

Эргономика - это наука, изучающая взаимодействие человека с окружающей средой с целью создания комфортных и удобных условий для работы и жизни. Эргономика занимается проектированием рабочих мест, оборудова-

ния, предметов быта и компьютерных программ для обеспечения безопасности и эффективности труда работника, учитывая физические и психические особенности человеческого организма.

Интерфейс пользователя IDE оказывает влияние на психофизиологические аспекты, скорость и эффективность работы. Согласно ГОСТ Р ИСО 9241-110-2016 [26] IDE PyCharm удовлетворяет семи основным принципам организации диалога пользователь-система:

1. IDE PyCharm соответствует принципу приемлемости организации диалога для выполнения производственного задания, поскольку в своем составе имеет большое количество инструментов и возможностей для эффективной работы разработчиков. Среди наиболее распространенных инструментов можно выделить следующие:

• автоматическое дополнение текста программы, которое позволяет ускорить набор и уменьшить количество опечаток;

• рефакторинг текста программы, которое позволяет улучшать структуру и читаемость текста программы:

• отладка текста программы, которая помогает выявлять и исправлять ошибки;

• интеграция с системами контроля версий, такими как Git, SVN и Mercurial, которая облегчает совместную работу над проектом.

2. IDE PyCharm соответствует принципу информативности, поскольку предоставляет разработчикам необходимую информацию о тексте программы, предупреждения и ошибки. Кроме того, IDE PyCharm предоставляет возможность настройки интерфейса и горячих клавиш, что позволяет пользователям адаптировать среду разработки под свои потребности и предпочтения, а также имеет систему подсказок к каждому элементу интерфейса и систему уведомлений разработчика.

3. IDE PyCharm соответствует принципу соответствия ожиданиям пользователей, поскольку предоставляет гибко настраиваемый интерфейс, возможность интеграции с другими инструментами и сервисами, такими как системы

контроля версий, базы данных и серверы приложений, а также имеет обширную документацию и активное сообщество пользователей, которое помогает решать возникающие вопросы и проблемы.

4. IDE PyCharm соответствует принципу пригодности для обучения, так как предоставляет доступ к большому количеству учебных ресурсов, включая видео-уроки, статьи и документацию, которые помогают обучающимся улучшить свои навыки программирования на Python. Кроме того, IDE PyCharm предоставляет возможность запуска и отладки текста программы прямо из среды разработки, что позволяет видеть результаты своей работы немедленно, а также имеет встроенный редактор текста программы с подсветкой синтаксиса, автодополнением и проверкой ошибок, что облегчает написание и отладку текста программы.

5. IDE PyCharm соответствует принципу контролируемости, так как предоставляет разработчикам инструменты для мониторинга и управления процессом разработки. Среди наиболее часто используемых инструментов можно выделить встроенную систему контроля версий, возможность настройки параметров выполнения текста программы, которая позволяет пользователям контролировать поведение программы во время выполнения. Кроме того, IDE PyCharm имеет мощные средства отладки, которые помогают пользователям находить и исправлять ошибки в тексте программы, а также хранит историю выполненных команд и имеет возможность отменить или вернуть последнее выполненное действие.

6. IDE PyCharm соответствует принципу устойчивости к ошибкам, так как предоставляет разработчикам инструменты для обнаружения и исправления ошибок в тексте программы. К таким инструментам относится встроенный статический анализатор текста программы, который может выявить потенциальные проблемы в тексте программы во время его написания, возможность запуска проверки текста программы, которая помогает выявить ошибки. Кроме того, среда разработки имеет мощные средства отладки, которые помогают разработчикам быстро диагностировать и исправлять ошибки в тексте

программы, а также имеет подсветку некорректных участков текста программы.

7. IDE PyCharm соответствует принципу адаптируемости к индивидуальным особенностям пользователя, так как имеет широкие возможности по настройке и персонализации интерфейса. К таким возможностям относится настройка цветовой схемы и горящих клавиш клавиатуры, выбор шрифтов и размеров текста. Кроме того, предоставляет возможность расширения функционала за счет установки дополнительных плагинов, что позволяет пользователям добавлять новые возможности и инструменты необходимые для работы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была достигнута поставленная цель - разработка модели искусственного интеллекта для прогнозирования действий программ в реальном времени, предназначенная для улучшения производительности операционных систем.

Для достижения поставленной цели были выполнены следующие задачи:

- Трассировка списка процессов. Этот шаг включает в себя получение списка последовательных событий (системных вызовов) с помощью утилиты strace.
- Обработка списка системных вызовов. На этом шаге произведено разбиение данных на токены, полученные данные разделены на обучающую, валидационную и тестовую выборки. Таким образом были подготовлены данные для обучения модели ИИ.
- Обзор существующих моделей искусственного интеллекта для прогнозирования системных вызовов.
- Обучение модели на обучающем наборе данных.
- Оценка качества модели ИИ на тестовой выборке.

В данной работе был произведен обзор моделей искусственного интеллекта для прогнозирования последующего системного вызова в операционной системе Linux. Для проведения обзора было рассмотрено пять моделей: предиктор последнего значения, предиктор на основе таблиц, унифицированный метод, предиктор на основе дерева решений, а также предсказание на основе Sequence-to-Sequence модели. В ходе анализа выбранных моделей нейронная сеть, состоящая из слоев Word Embedding, Bidirectional LSTM и линейного, является более эффективным методом прогнозирования по сравнению с другими методами. Предиктор последнего значения и предиктор на основе таблиц менее эффективны, так как они не учитывают дополнительный контекст или сложные взаимосвязи между словами. Унифицированный метод и предиктор

на основе дерева решений также могут иметь ограничения в обработке последовательных данных и определении сложных зависимостей. Sequence-to-Sequence модель может быть менее эффективной, чем нейронная сеть с LSTM слоями, т. к. она обычно используется для задач генерации текста или перевода.

Тестирование выбранной нейронной сети со слоями Word Embedding, Bidirectional LSTM и линейного показало, что нейронная сеть имеет недостаточно хорошее время предсказания, то есть время предсказания превышает допустимое время в работе ОС. В связи с этим было принято решение упростить модель, используя только 2 слоя: Word Embedding и линейный. Упрощенная модель показала незначительное снижение точности (1%), но при этом скорость прогнозирования уменьшилась приблизительно в 10 раз, что стало сопостовимым со временнем работы в ОС.

Для дальнейшего улучшения точности и скорости прогнозирования поведения программ в операционных системах, предлагается настраивать параметры нейронной сети, исследовать возможность использования различных видов нейронных сетей, а также вместо прогнозирования названия следующего системного вызова генерировать последовательность системных вызовов вместе с их аргументами и результатами. Кроме того, целесообразно изучить возможность использования разработанной модели для прогнозирования других типов аномалий в операционных системах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Duesterwald E., Cascaval C., Dwarkadas S. Characterizing and predicting program behavior and its variability //2003 12th International Conference on Parallel Architectures and Compilation Techniques. – IEEE, 2003. – С. 220-231.
2. Sarikaya R., Buyuktosunoglu A. A unified prediction method for predicting program behavior //IEEE Transactions on Computers. – 2009. – Т. 59. – №. 2. – С. 272-282.
3. Negi A., Kumar P. K. Applying machine learning techniques to improve linux process scheduling //TENCON 2005-2005 IEEE Region 10 Conference. – IEEE, 2005. – С. 1-6.
4. Lv S. H. et al. Intrusion prediction with system-call sequence-to-sequence model //IEEE Access. – 2018. – Т. 6. – С. 71413-71421.
5. Xu C. et al. Cache contention and application performance prediction for multi-core systems //2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS). – IEEE, 2010. – С. 76-86.
6. Thaker N., Shukla A. Python as multi paradigm programming language //International Journal of Computer Applications. – 2020. – Т. 177. – №. 31. – С. 38-42.
7. https://gs.statcounter.com/os-market-share/desktop/worldwide
8. Tsafrir D. The context-switch overhead inflicted by hardware interrupts (and the enigma of do-nothing loops) //Proceedings of the 2007 workshop on Experimental computer science. – 2007. – С. 4-es.
9. Gers F. A., Schmidhuber J., Cummins F. Learning to forget: Continual prediction with LSTM //Neural computation. – 2000. – Т. 12. – №. 10. – С. 2451-2471.
10. Hochreiter S., Schmidhuber J. LSTM can solve hard long time lag problems //Advances in neural information processing systems. – 1996. – Т. 9.
11. Hochreiter S., Schmidhuber J. Long short-term memory //Neural computation. – 1997. – Т. 9. – №. 8. – С. 1735-1780.

12. Gers F. A., Schraudolph N. N., Schmidhuber J. Learning precise timing with LSTM recurrent networks //Journal of machine learning research. – 2002. – Т. 3. – №. Aug. – С. 115-143.
13. Huang Z., Xu W., Yu K. Bidirectional LSTM-CRF models for sequence tagging //arXiv preprint arXiv:1508.01991. – 2015.
14. Yu Y. et al. A review of recurrent neural networks: LSTM cells and network architectures //Neural computation. – 2019. – Т. 31. – №. 7. – С. 1235-1270.
15. Staudemeyer R. C., Morris E. R. Understanding LSTM--a tutorial into long short-term memory recurrent neural networks //arXiv preprint arXiv:1909.09586. – 2019.
16. Shimodaira H. Single Layer Neural Networks //Learning and Data Note. – 2015.
17. BB S. D. S. P. D., Varshapriya M. P., Ambavkar P. P. P. DEBUGGING ON LINUX.
18. Subramanian V. Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch. – Packt Publishing Ltd, 2018.
19. Sergeevna Bosman A., Engelbrecht A., Helbig M. Visualising Basins of Attraction for the Cross-Entropy and the Squared Error Neural Network Loss Functions //arXiv e-prints. – 2019. – С. ArXiv: 1901.02302.
20. Golik P., Doetsch P., Ney H. Cross-entropy vs. squared error training: a theoretical and experimental comparison //Interspeech. – 2013. – Т. 13. – С. 1756-1760.
21. Glorot X., Bengio Y. Understanding the difficulty of training deep feedforward neural networks //Proceedings of the thirteenth international conference on artificial intelligence and statistics. – JMLR Workshop and Conference Proceedings, 2010. – С. 249-256.
22. Allen C., Hospedales T. Analogies explained: Towards understanding word embeddings //International Conference on Machine Learning. – PMLR, 2019. – С. 223-231.

23. Белехов Александр Николаевич Научно-технический прогресс и безопасность труда // Инновации и инвестиции. 2016. №3. URL: https://cyberleninka.ru/article/n/nauchno-tehnicheskiy-progress-i-bezopasnost-truda (дата обращения: 12.05.2024).
24. Steiner B. et al. Pytorch: An imperative style, high-performance deep learning library. – 2019.
25. СанПиН 2.2.2/2.4.1340-03.
26. ГОСТ Р ИСО 9241-110-2016 [Электронный ресурс]. URL: https://docs.cntd.ru/document/1200141125 (дата обращения: 12.05.2024).