

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

ПРИМЕНЕНИЕ МЕТОДОВ МАШИННОГО
ОБУЧЕНИЯ ДЛЯ ВЫЯВЛЕНИЯ АНОМАЛИЙ
БЕЗОПАСНОСТИ

Студент

 Гречишкина Маргарита Сергеевна
(подпись, дата) 08.06.2026 (ФИО полностью)

Руководитель

 08.06.2026 Афонин А.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

 08.06.2026 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

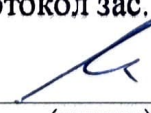
Рецензент

(должность, место работы)

 Севастьянов А. В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 № 14)

Заведующий кафедрой

 М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от
09 июня 2026, № 22)

Секретарь ГЭК

 Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 20 » 09 2024 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Гречишкина Маргарита Сергеевна гр. 24ВВИм1 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы Применение методов машинного обучения для выявления аномалий безопасности

Тема утверждена приказом ПГУ № 920/о от " 20 " 09 2024 г.

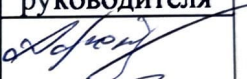
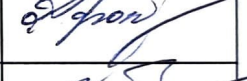
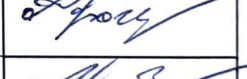
3. Руководитель магистерской диссертации Афонин Александр Юрьевич

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Систематическая оценка и сравнение моделей глубокого обучения для обнаружения аномалий по журналам регистрации, разработка программного модуля для автоматизированного проведения экспериментов и формирование практической методики выбора модели. Наборы данных для исследования: HDF5, BGL, Spirit, Thunderbird.

5. Перечень подлежащих разработке вопросов _____
Методология предобработки журнальных данных, анализ архитектур и особенностей моделей глубокого обучения, обоснование выбора метрик качества, анализ датасетов, разработка программного модуля с унифицированным пайплайном обучения и оценки моделей

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Ознакомление с тематикой исследования.	7%	20.09.2024 – 31.10.2024	
2	Обзор и анализ актуальных публикаций по теме.	10%	01.11.2024 – 30.11.2024	
3	Подготовка и предварительная обработка наборов данных	14%	01.12.2024 – 31.01.2025	
4	Разработка программного модуля: реализация пайплайна парсинга, группировки и векторизации.	14%	01.02.2025 – 31.03.2025	
5	Интеграция моделей глубокого обучения в программный модуль.	14%	01.04.2025 – 31.05.2025	
6	Проведение экспериментов: оценка влияния стратегий отбора данных и методов группировки.	14%	01.06.2025 – 31.07.2025	
7	Исследование влияния дисбаланса классов и зашумлённости на качество обнаружения.	7%	01.08.2025 – 30.09.2025	
8	Обобщение результатов, формирование методики выбора модели	10%	01.10.2025 – 30.11.2025	
9	Оформление пояснительной записки.	5%	01.12.2025 – 28.02.2026	
10	Подготовка к защите.	5%	01.03.2026 – 21.05.2026	

Дата выдачи задания "20" _____ 09 _____ 2024г.

Руководитель магистерской диссертации _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
09.06.2026 (подпись, дата)

РЕФЕРАТ

Магистерская работа содержит 108 листов, 13 рисунков, 18 таблиц, 42 источника, 2 приложения.

ОБНАРУЖЕНИЕ АНОМАЛИЙ, АНАЛИЗ ЛОГОВ, ПАРСИНГ ЛОГОВ, ГЛУБОКОЕ ОБУЧЕНИЕ

Программно-интенсивные системы генерируют журналы регистрации событий для целей диагностики и устранения неисправностей. В последнее время предложено значительное число моделей глубокого обучения для автоматизированного выявления аномалий в работе систем на основе данных журналов. Как правило, подобные модели демонстрируют весьма высокую точность обнаружения: большинство из них достигают значения F-меры выше 0,9. Для глубокого понимания того, насколько далеко мы продвинулись в решении задачи обнаружения аномалий по журналам регистрации, в настоящей работе проводится детальный анализ пяти современных моделей глубокого обучения для выявления системных аномалий на четырёх открытых наборах данных журналов.

Эксперименты охватывают несколько аспектов оценки модели: стратегии отбора обучающих данных, методы группировки данных, распределение классов, зашумлённость данных и способность к раннему обнаружению аномалий. Полученные результаты свидетельствуют о том, что все перечисленные аспекты оказывают существенное влияние на процедуру оценки и что ни одна из исследованных моделей не демонстрирует устойчивой работоспособности во всех условиях.

Задача обнаружения аномалий на основе журналов регистрации по-прежнему не решена. На основе выявленных закономерностей предложены направления дальнейших исследований.

				ПГУ 1.090401.05.001 ПЗ			
	№ докум.	Подпись	Дата	Применение методов машинного обучения для выявления аномалий безопасности Пояснительная записка	Лит.	Лист	Листов
Разраб.	Гречаликина М.С.		08.06.26				
Провер.	Афонин А.Ю.		08.06.26			2	108
Н. Контр	Бычков А.С.		08.06.26		ФВТ, каф. ВТ гр. 24ВВИм1		
Утверд.	Митрохин М.А.		08.06.26				

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ	5
ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	7
ВВЕДЕНИЕ	8
1 Датасеты	10
2 Предобработка данных	13
2.1 Парсинг логов	14
2.2 Группировка логов	15
2.3 Представление логов.....	16
3 Модели глубокого обучения	18
3.1 DeepLog	18
3.2 LogAnomaly.....	20
3.3 PLELog.....	23
3.4 LogRobust	26
3.5 CNN.....	27
4 Программный модуль для анализа производительности ГО	29
4.1 Входные данные	29
4.2 Параметры запуска	30
4.3 Основные компоненты.....	32
4.4 Выходные данные.....	34
5 Результаты и основные выводы	35
5.1 Производительность при различных стратегиях выбора обучающих данных	37
5.2 Производительность при различных методах группировки данных	39
5.3 Производительность при различных распределениях классов.....	42
5.4 Производительность при различной степени зашумлённости данных	45
5.5 Раннее обнаружение аномалий	50
6 Обсуждение.....	52

6.1	Преимущества и недостатки исследованных методов	52
6.2	Направления будущих исследований	54
6.3	Угрозы достоверности анализа	55
7	Методика выбора модели глубокого обучения для обнаружения аномалий при интеграции в комплексную систему безопасности	57
7.1	Аудит данных.....	57
7.2	Согласование с бизнес-требованиями.....	61
7.3	Сводная матрица принятия решений.....	63
7.4	Практический алгоритм выбора.....	64
	ЗАКЛЮЧЕНИЕ.....	67
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	69
	Приложение А Код программы.....	74
	Приложение Б Презентация.....	104

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В данной работе используются следующие термины с соответствующими определениями.

Аномалия (anomaly) – отклонение поведения системы от нормального, выражающееся в нарушении ожидаемых последовательностей событий, нетипичных частотных распределениях или иных паттернах, характерных для сбоя, атаки или нештатного состояния

Векторизация логов (log representation) – преобразование последовательности шаблонов событий лога в числовое векторное представление, пригодное для подачи в модель глубокого обучения. Существуют три основных типа: последовательные, количественные и семантические векторы

Группировка логов (log grouping) – разделение потока записей лога на конечные группы – последовательности логов, – из которых извлекаются признаки для моделей обнаружения аномалий. Второй этап предобработки

Датасет (dataset) – публичный набор логов реальных вычислительных систем, размеченный или снабжённый бинарным признаком аномалии

Дисбаланс классов (class imbalance) – ситуация, при которой доля аномальных последовательностей в датасете значительно меньше доли нормальных (например, 0,49% в Thunderbird). Существенно усложняет задачу обнаружения аномалий: стандартные метрики (точность, полнота, F-мера) могут давать вводящие в заблуждение результаты

Лог (журнальная запись, log) – текстовое сообщение, автоматически порождаемое программной системой и фиксирующее отдельное событие её работы: запуск процесса, сбой компонента, сетевую ошибку, изменение состояния и т. п.

Механизм внимания (attention mechanism) – стратегия, присваивающая различным элементам входной последовательности различные веса важности. Скрытые состояния суммируются с учётом весов внимания, вычисляемых функцией softmax. Позволяет модели акцентировать внимание на информативных событиях и маскировать зашумлённые

Парсинг логов (log parsing) – автоматическое преобразование каждого сырого сообщения лога в пару «шаблон события (постоянная часть) + параметры (переменная часть)». Парсинг – первый этап предобработки

Рекуррентная нейронная сеть (RNN, Recurrent Neural Network) – искусственная нейронная сеть, использующая цикл для передачи выхода предыдущего состояния на текущий вход, что позволяет ей отслеживать историю при формировании предсказаний и работать с последовательными данными произвольной длины

Свёрточная нейронная сеть (CNN, Convolutional Neural Network) – нейронная сеть, извлекающая локальные паттерны входных данных посредством операции свёртки – применения набора обучаемых фильтров к локальным рецептивным полям. Позволяет улавливать не только корреляцию между шаблонами журналов, но и корреляцию внутри семантического представления (эмбединга) шаблонов

Утечка данных (data leakage) – проблема, возникающая при случайном выборе обучающих данных: модель получает доступ к информации о будущих событиях на этапе обучения, что приводит к завышенным показателям точности, не отражающим реальную эффективность в условиях эксплуатации

Шаблон события – постоянная часть сообщения лога, получаемая в результате парсинга. Каждому сообщению лога ставится в соответствие конкретный шаблон события, связанный с переменными параметрами (переменной частью)

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В данной работе используются следующие обозначения и сокращения

AEL – Abstracting Execution Logs

BERT – Bidirectional Encoder Representations from Transformers

BGL – Blue Gene/L

Bi-LSTM – Bidirectional Long Short-Term Memory

BROP – Blind Return-Oriented Programming

dLCE – Distributional Lexical-Contrast Embedding

EC2 – Amazon Elastic Compute Cloud

GloVe – Global Vectors for Word Representation

GRU – Gated Recurrent Unit

HDFS – Hadoop Distributed File System

HDBSCAN – Hierarchical Density-Based Spatial Clustering of Applications with Noise

IDS – Intrusion Detection System

IPLoM – Iterative Partitioning Log Mining

LLNL – Lawrence Livermore National Laboratory

LSTM – Long Short-Term Memory

NLP – Natural Language Processing

PU – Positive-Unlabeled learning

ReLU – Rectified Linear Unit

RNN – Recurrent Neural Network

TF-IDF – Term Frequency – Inverse Document Frequency

URL – Uniform Resource Locator

ВВЕДЕНИЕ

Высокая доступность и надёжность являются обязательными требованиями для крупномасштабных программных систем. Поскольку такие системы предоставляют различные услуги большому числу пользователей, даже незначительная проблема в системе может привести к неудовлетворённости пользователей и даже к существенным финансовым потерям. Обнаружение аномалий, таким образом, важно для обеспечения качества сложных программных систем [1].

Программные системы часто записывают информацию о работе в виде консольных логов. Крупная и сложная система может генерировать огромный объём логов, которые могут быть использованы для диагностики неисправностей [2]. Данные логов обычно представляют собой неструктурированные текстовые сообщения, которые помогают инженерам понять внутреннее состояние системы и облегчают мониторинг, администрирование и устранение неполадок [3]. Сообщения логов могут быть преобразованы в события логов – шаблоны (постоянная часть) сообщений.

На протяжении многих лет было предложено множество подходов на основе данных для автоматического обнаружения системных аномалий путём анализа данных логов [4, 5, 6, 7, 8, 9, 10]. Например, была проведена оценка шести распространённых алгоритмов машинного обучения применительно к задаче обнаружения аномалий по журналам [11]. Методы на основе машинного обучения страдают рядом общих недостатков: негибкость признаков, низкая эффективность и слабая адаптируемость [10]. Для устранения этих ограничений было применено глубокое обучение, давшее обнадеживающие результаты. Было предложено использовать сети с долговременной краткосрочной памятью (LSTM) для моделирования последовательностей нормального выполнения системы [6], а затем выявлять аномалии как нарушения предсказанных паттернов [12]. В целом существующие методы глубокого обучения демонстрируют перспективные результаты на широко используемых наборах данных и претендуют на превосходство над традиционными подходами

машинного обучения: DeepLog [6] и LogAnomaly [13] показывают высокое значение F-меры на наборах HDFS и BGL (свыше 0,9).

Целью данной работы является комплексное исследование методов обнаружения аномалий по журналам регистрации, основанных на глубоком обучении. Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ существующих подходов к обнаружению аномалий по журналам регистрации и обосновать выбор репрезентативных моделей глубокого обучения для исследования,
- исследовать и описать методы предобработки журнальных данных – парсинг, группировку и векторизацию – как факторы, определяющие качество обнаружения аномалий,
- разработать программный модуль, реализующий унифицированный пайплайн обучения и оценки пяти моделей глубокого обучения на четырёх наборах данных с гибкой конфигурацией параметров эксперимента,
- провести систематическую оценку исследуемых моделей при различных стратегиях отбора обучающих данных и методах группировки журналов,
- выявить влияние дисбаланса классов и зашумлённости данных на качество обнаружения аномалий,
- оценить способность исследуемых моделей к раннему обнаружению аномалий в режиме онлайн-детектирования,
- разработать методику выбора модели глубокого обучения для обнаружения аномалий при интеграции в комплексную систему безопасности на основе характеристик данных и операционных приоритетов организации.

1 Датасеты

Как уже было упомянуто, современные вычислительные системы – серверы, кластеры, суперкомпьютеры – непрерывно генерируют журнальные записи (логи). Каждая строка лога фиксирует событие: запуск процесса, сбой компонента, сетевую ошибку, изменение состояния. Публичные датасеты позволяют сравнивать алгоритмы на единой базе и воспроизводить результаты. В данной работе для оценки исследуемых моделей были выбраны четыре публичных датасета [16, 17].

HDFS (Hadoop Distributed File System) содержит консольные логи распределённой файловой системы Hadoop. Датасет получен с более чем 200 узлов Amazon EC2 и сохраняет исходную структуру строк, добавляя бинарный признак аномалии. Логи собирались при различных нагрузках (например, кластеры разного масштаба, разные типы пользовательских запросов, различные уровни рабочей нагрузки). HDFS содержит 11 175 629 сообщений, сгруппированных в последовательности логов по `block_id`. Это делает HDFS удобным для задач классификации: последовательность событий внутри одного блока чётко отражает его жизненный цикл. 16 838 блоков (2,93%) в этом датасете указывают на системные аномалии. Каждая строка содержит следующие параметры:

- `date` – шестизначная метка даты из исходного консольного вывода (формат ГГММДД),
- `time` – шестизначная метка времени (формат ЧЧММСС),
- `pid` – идентификатор процесса, извлечённый из строки лога,
- `level` – уровень логирования (например, INFO),
- `component` – компонент Java/демон, сгенерировавший запись лога,
- `content` – подробное текстовое содержание события,
- `block_id` – идентификаторы блоков, обнаруженные в строке лога, разделённые пробелами (пустое значение при отсутствии),

– anomaly – бинарный индикатор, полученный из меток на уровне блоков (1 – аномальный блок).

BGL (Blue Gene/L) – датасет логов суперкомпьютера, собранный Ливерморской национальной лабораторией (LLNL) [18]. Датасет содержит 4 747 963 сообщения от компонентов суперкомпьютера: ядра ОС, коммуникационной сети, узлов памяти. В отличие от HDFS, каждая строка уже имеет метку – либо «-» (нормально), либо код типа сбоя (например, APPREAD, KERNREC). 348 460 сообщений (7,34%) были помечены как аномальные. В таблице 1 приведены наиболее популярные типы сбоев с примерами.

Таблица 1 – Примеры сообщений разных типов сбоев в BGL

Категория сбоя	Тип сбоя	Количество	Пример сообщения
H	KERNDTLB	152 734	data TLB error interrupt
H	KERNSTOR	63 491	data storage interrupt
S	APPSEV	49 651	ciod: Error reading message prefix after LOGIN MESSAGE on CioStream [...]
S	KERNMNTF	31 531	Lustre mount FAILED : bglio11 : block id : location
S	KERNTERM	23 338	rts: kernel terminated for reason 1004rts: bad message header: [...]
S/I	Другие	27 715	machine check interrupt

Текст в квадратных скобках обозначает опущенную информацию. Категории сообщений различаются в зависимости от системы – в силу особенностей конфигурации, механизмов журналирования и того, что администраторы каждой системы считают важным для отслеживания. Для каждого типа сообщений указано общее количество сообщений, категория сбоя: H (аппаратный сбой), S (программный сбой) и I (неопределённый).

Неопределённые оповещения могут быть вызваны как аппаратными, так и программными причинами, либо их источник неизвестен.

Spirit – один из крупнейших публичных датасетов. Логи описывают работу высокопроизводительного кластера, включая системные вызовы, сетевые события и сообщения планировщика задач суперкомпьютерной системы Spirit в Сандийских национальных лабораториях [18]. Главная особенность Spirit – масштаб. Датасет содержит около 272 миллионов строк. Именно поэтому его используют для проверки масштабируемости алгоритмов: методы, хорошо работающие на HDFS и BGL, могут не справляться с объёмом Spirit. В данном исследовании использовались первые 5 миллионов строк оригинального датасета, содержащий 764 500 аномальных сообщений (15,29%).

Thunderbird – открытый датасет логов суперкомпьютера Thunderbird в Сандийских национальных лабораториях [18]. Thunderbird выделяется наибольшим числом уникальных шаблонов среди четырёх датасетов, что создаёт максимальную сложность для парсинга и векторного представления. В данной работе используются первых 10 миллионов непрерывных строк логов, содержащих 4 934 аномальных сообщения (0,49%).

2 Предобработка данных

Качество обнаружения аномалий в журнальных записях определяется не только выбором модели, но и тем, насколько корректно данные подготовлены до её применения. Журнальные записи представляют собой полуструктурированный текст, непригодный для непосредственного использования в алгоритмах машинного обучения. Прежде чем модель получит возможность выявлять отклонения, сырые строки логов должны пройти несколько последовательных этапов преобразования: парсинг, группировку и векторизацию. Ошибка на любом из этих шагов искажает входные данные и неизбежно сказывается на итоговых метриках. На рисунке 1 представлена схема выполнения описанных преобразований.

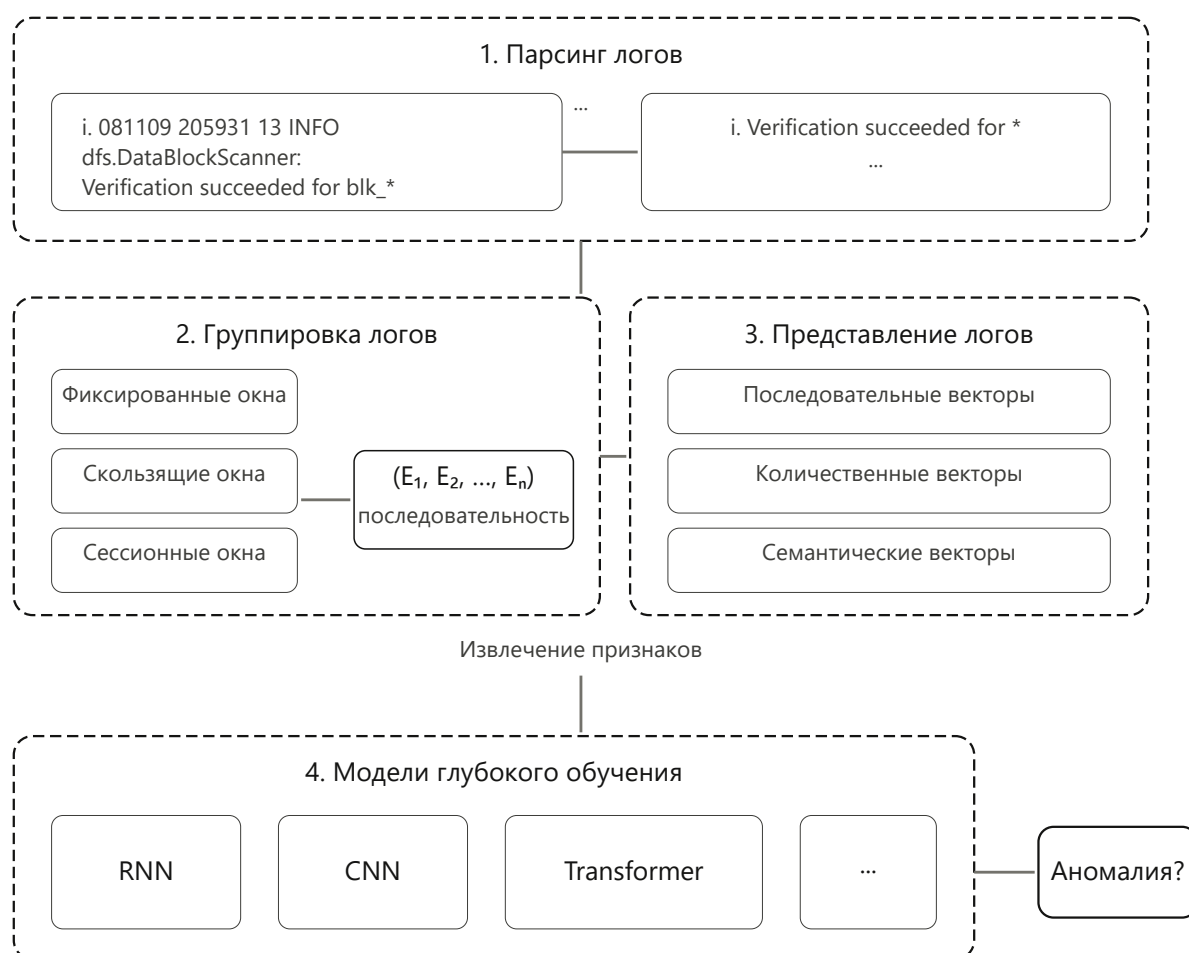


Рисунок 1– Общая схема пайплайна обнаружения аномалий

Это обстоятельство нередко остаётся недооценённым при сравнительном анализе методов. Когда модель демонстрирует низкие значения точности или полноты на известных бенчмарках, исследователи склонны объяснять неудачу её архитектурными ограничениями и обращаться к поиску более сложных решений. Между тем причина может лежать значительно раньше в пайплайне: неверно извлечённые шаблоны событий, некорректно определённые границы сессий или неподходящий способ векторного представления способны до такой степени исказить структуру данных, что даже теоретически сильная модель окажется неспособна уловить значимые закономерности [19].

В далее в работе (в четвертой главе) рассматривается каждый из этапов применительно к исследуемым датасетам. Особое внимание уделяется типичным ошибкам и их влиянию на конечный результат: показывается, каким образом выбор инструмента парсинга, стратегии группировки и метода векторизации формирует то представление данных, с которым в действительности работает модель. Понимание этих зависимостей является необходимым условием корректной интерпретации результатов и обоснованного сравнения методов обнаружения аномалий.

2.1 Парсинг логов

Логи представляют собой полуструктурированные тексты, содержащие различные поля, такие как временная метка и уровень важности. Для последующих задач парсинг логов применяется для автоматического преобразования каждого сообщения лога в конкретный шаблон события (постоянная часть), связанный с параметрами (переменная часть). Схема описанного процесса отражена на рисунке 2.

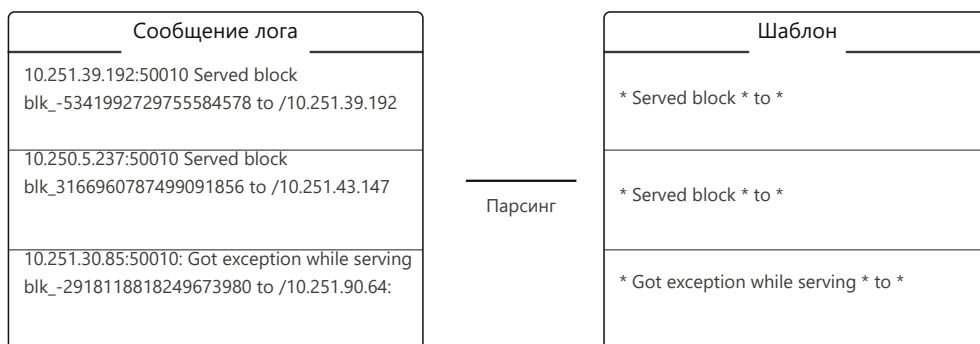


Рисунок 2 – Преобразование логов в шаблоны

Существует множество методов парсинга логов, основанных на частотном анализе паттернов [20, 21, 22], кластеризации [23, 24, 25] и эвристиках [25, 26, 27]. Подходы на основе эвристик используют характеристики логов и показывают лучшие результаты по сравнению с другими методами по точности и временной эффективности [29]. Для извлечения шаблонов логов используется парсер Drain [26] с параметрами по умолчанию [30].

2.2 Группировка логов

Основная цель этого этапа – разделить логи на различные группы, где каждая группа представляет собой конечное количество логов [11]. Эти группы называются последовательностями логов, из которых извлекаются признаки для моделей обнаружения аномалий. Три типа окон применяются для группировки логов: фиксированное окно – логи группируются по фиксированным окнам в зависимости от временной метки или порядка появления в датасете; скользящее окно состоит из двух атрибутов: размера окна и размера шага; сессионное окно в отличие от фиксированных/скользящих окон, основывается на идентификаторе логов. Выбор правильного размера окна является сложной задачей. Если размер окна мал, модели испытывают трудности с обнаружением аномалий, которые охватывают несколько последовательностей. С другой стороны, если размер окна велик, последовательности логов могут включать несколько аномалий и дезориентировать схему обнаружения [31].

2.3 Представление логов

После группировки логи представляются в различных форматах, требуемых моделями глубокого обучения (ГО). Выбор формата представления непосредственно влияет на то, какую информацию модель способна извлечь из журналов событий. Существующие подходы преобразуют логи в три основных типа векторных представлений: последовательные, количественные и семантические векторы.

Последовательные векторы отражают порядок появления событий в рамках заданного временного окна. Такое представление сохраняет информацию о том, в какой последовательности происходили события, что позволяет моделям выявлять аномалии, связанные с нарушением нормального порядка выполнения операций. Например, если в штатном режиме событие А всегда предшествует событию В, то обратная последовательность может сигнализировать об аномалии. Данный подход лежит в основе модели DeepLog, которая моделирует журналы как временные последовательности с помощью LSTM, предсказывая следующее ожидаемое событие и фиксируя отклонения от предсказанного порядка.

Количественные векторы аналогичны векторам подсчёта частот: каждый элемент вектора соответствует определённому шаблону события лога и содержит число его появлений в пределах окна. В отличие от последовательных векторов, количественное представление не сохраняет порядок событий, однако фиксирует их частоту, что позволяет обнаруживать аномалии, связанные с нетипичным распределением событий – например, резким ростом числа ошибок определённого типа. LogAnomaly объединяет оба типа векторов, последовательный и количественный, что позволяет модели одновременно учитывать как структуру потока событий, так и статистику их встречаемости.

Семантические векторы получают с помощью языковых моделей и позволяют кодировать смысловое содержание шаблонов событий, а не только их идентификаторы или частоту появления. Такой подход обеспечивает лучшую обобщаемость: семантически близкие события получают схожие векторные

представления, что помогает моделям справляться с ранее не встречавшимися шаблонами. В зависимости от применяемой языковой модели возможности семантического кодирования существенно различаются. PLELog использует векторы на основе GloVe – предобученной модели, представляющей слова в виде плотных числовых векторов на основе их совместной встречаемости в больших корпусах текстов [32]. Текст шаблона события усредняется по словам для получения итогового вектора. LogRobust и CNN применяют FastText – модель, аналогичную GloVe, но учитывающую морфологию слов через анализ подсловных единиц (n-gramm символов), что делает её более устойчивой к опечаткам и редким словам, часто встречающимся в технических логах [33].

Таким образом, выбор типа векторного представления отражает компромисс между разными аспектами информации о логах: последовательные векторы акцентируют порядок событий, количественные – их статистику, а семантические – смысловое содержание. Ряд моделей комбинирует несколько типов, стремясь охватить более широкий спектр паттернов аномалий.

3 Модели глубокого обучения

Применяются три класса архитектур: рекуррентные нейронные сети (RNN), их варианты – LSTM [21] и GRU [9], – предназначенные для работы с последовательными данными произвольной длины, и свёрточные нейронные сети (CNN), извлекающие локальные паттерны входных данных посредством операции свёртки, получившие широкое распространение в задачах нейронного машинного перевода и применяемые для обнаружения аномалий по журналам в ряде работ. Рассмотрим наиболее представительные современные модели.

3.1 DeepLog

Рекуррентная нейронная сеть (RNN) – это искусственная нейронная сеть, использующая цикл для передачи выхода предыдущего состояния на текущий вход, что позволяет ей отслеживать историю при формировании предсказаний. Сети с долговременной краткосрочной памятью (LSTM) являются разновидностью RNN, обладающей способностью запоминать долгосрочные зависимости в последовательностях. LSTM продемонстрировали высокую эффективность в различных задачах, таких как машинный перевод, анализ тональности и медицинская самодиагностика. LSTM-модель применяется для обучения на нормальных последовательностях выполнения системы путём предсказания следующего события журнала по предшествующим событиям. Аномалия фиксируется в том случае, если очередное событие журнала нарушает результаты предсказания LSTM-модели. Согласно результатам экспериментов, DeepLog достигает значения F-меры 0,96 на наборе данных HDFS.

Архитектура DeepLog включает три основных компонента: модель обнаружения аномалий по ключам журнала, модель обнаружения аномалий по значениям параметров и модель рабочего процесса для диагностики обнаруженных аномалий.

Сначала опишем, как обнаруживать аномалии пути выполнения с помощью последовательности ключей журнала. Поскольку общее число различных операторов печати (выводящих записи журнала) в исходном коде постоянно, постоянно и общее число различных ключей журнала. Пусть $K = \{k_1,$

$k_2, \dots, k_n\}$ – множество различных ключей журнала из исходного кода системы, порождающей журнал.

После разбора записей в ключи журнала последовательность ключей отражает путь выполнения, приводящий к данному конкретному порядку операторов печати. Пусть m_i обозначает значение ключа в позиции i в последовательности ключей. Очевидно, что m_i может принимать одно из n возможных значений из K и сильно зависит от наиболее недавних ключей, появившихся перед m_i .

Обнаружение аномалий в последовательности ключей журнала может быть смоделировано как задача многоклассовой классификации, где каждый отдельный ключ журнала определяет класс. Мы обучаем DeepLog как многоклассовый классификатор над недавним контекстом. Входом служит история недавних ключей журнала, а выходом – распределение вероятностей по n ключам из K , представляющее вероятность того, что следующим ключом в последовательности будет ключ $k_i \in K$.

Пусть t – порядковый номер следующего появляющегося ключа журнала. Входом для классификации служит окно w из h наиболее недавних ключей журнала: $w = \{m_{t-h}, \dots, m_{t-2}, m_{t-1}\}$. Выходом этапа обучения является модель условного распределения вероятностей $\Pr[m_t = k_i | w]$ для каждого $k_i \in K$. Этап обнаружения использует данную модель для формирования предсказания и сравнения предсказанного выхода с фактически наблюдаемым значением ключа журнала. Каждый LSTM-блок хранит состояние своего входа в виде вектора фиксированной размерности. Состояние LSTM-блока с предыдущего временного шага также подаётся на его следующий вход вместе с внешним входом данных для вычисления нового состояния и выхода. Таким образом историческая информация передаётся и поддерживается в пределах LSTM-блока. При наложении нескольких слоёв друг на друга образуется глубокая нейронная сеть LSTM.

Этап обучения заключается в нахождении надлежащих значений весов, так чтобы итоговый выход последовательности LSTM-блоков порождал требуемую

метку (выход), сопровождающую входы в обучающем наборе данных. В процессе обучения каждая пара «вход–выход» инкрементно обновляет эти веса посредством минимизации потерь методом градиентного спуска. В DeepLog вход состоит из окна w из h ключей журнала, а выход – это значение ключа журнала, следующее непосредственно за w . Для обучения используется функция потерь категориальной перекрёстной энтропии.

DeepLog осваивает всесторонние и сложные корреляции и паттерны, заложенные в последовательности записей журнала, порождаемой путями нормального выполнения системы. Далее мы предполагаем, что сами системные журналы защищены и злоумышленник не может нарушить целостность журнала, равно как и изменить исходный код системы для модификации её поведения при журналировании. С учётом этого мы рассматриваем два типа атак: атаки, приводящие к некорректному выполнению системы и, следовательно, к аномальным паттернам в журналах (например, атаки типа «отказ в обслуживании», вызывающие замедление выполнения; атаки, вызывающие повторные перезапуски сервера, такие как BROP-атака; любые атаки, способные привести к аварийному завершению задачи); атаки, способные оставить след в системных журналах вследствие деятельности служб мониторинга системы (например, подозрительная активность, регистрируемая системой обнаружения вторжений, IDS).

3.2 LogAnomaly

LogAnomaly использует векторы частотности событий журнала в качестве входных данных для обучения LSTM-модели [13]. Авторами также разработан метод `template2vec` – основанный на синонимах и антонимах подход к представлению шаблонов журналов в виде семантических векторов для сопоставления новых событий с существующими шаблонами. Как и DeepLog, LogAnomaly использует модель прогнозирования следующего события журнала: если наблюдаемое событие нарушает результаты предсказания, оно помечается как аномальное. В исходной работе LogAnomaly достигает значений F-меры 0,95 и 0,96 на наборах HDFS и BGL соответственно.

Ключевая идея LogAnomaly состоит в том, что большинство системных журналов являются полуструктурированными текстами, выводимыми определёнными процедурами систем, в силу чего интуиции и методы обработки естественного языка могут быть применены или усовершенствованы для обнаружения аномалий по журналам. Программы, порождающие журналы, обычно выполняются согласно фиксированным потокам. Вследствие этого нормальные журналы естественным образом обладают определёнными последовательностными паттернами. Иными словами, для заданной последовательности векторов шаблонов её следующий шаблон предсказуем при отсутствии аномалии. Пусть $\Omega = \{v_1, v_2, \dots, v_n\}$ – полное множество различных векторов шаблонов. Последовательностью для обнаружения служит скользящее окно из w наиболее недавних векторов шаблонов. Поскольку LSTM является распространённой архитектурой рекуррентной нейронной сети, доказавшей способность устойчиво прогнозировать последовательности данных, она применяется для освоения последовательностного паттерна журналов.

Помимо последовательстных паттернов, вектора шаблонов обладают также количественными паттернами. Как правило, нормальное выполнение программы характеризуется некоторыми инвариантами, и в журналах при различных входных данных и рабочих нагрузках неизменно выполняются определённые количественные отношения. Например, каждый открытый файл в конечном счёте будет закрыт на некотором этапе, в силу чего в нормальных условиях число журнальных записей, указывающих на «открытие файла», должно быть равно числу записей, обозначающих «закрытие файла». Эти количественные отношения отражают поведение нормального выполнения программы. Если новая запись нарушает определённые инварианты, можно заключить, что в ходе выполнения системы возникла аномалия. Для освоения количественного паттерна для каждого журнального сообщения вычисляется вектор частотности соответствующей журнальной последовательности, который подаётся в LSTM.

Еще одной особенностью является новый метод представления слов `template2Vec`, основанный на синонимах и антонимах и обеспечивающий эффективное распределённое представление слов шаблонов. Метод `template2Vec` включает три этапа на стадии офлайн-обучения:

- универсальные синонимы и антонимы могут быть найдены в WordNet – лексической базе данных английского языка. Некоторые предметно-специфические синонимы и антонимы, однако, должны добавляться операторами на основе предметных знаний. Таким образом, сначала выполняется поиск синонимов и антонимов слов шаблонов в WordNet, после чего операторы могут обновить эти множества,
- применяется модель dLCE (distributional lexical-contrast embedding) для порождения векторов слов,
- для заданного шаблона его вектор вычисляется как взвешенное среднее векторов слов, входящих в шаблон. На стадии онлайн-обнаружения «временный» вектор шаблона для журналов нового типа сопоставляется с существующим.

Таким образом, метод `template2Vec` объединяет предметные знания операторов и модель dLCE с целью точного порождения векторов шаблонов.

В исходной работе так же приводится пример применения этих идей на практике. Был рассмотрен единичный случай аномалии, произошедшей на агрегирующем коммутаторе, развёрнутом у одного из ведущих провайдеров облачных услуг, в период с 15:00 13 октября 2018 г. по 1:16 14 октября 2018 г. Методы LogAnomaly и DeepLog были развёрнуты для обнаружения аномалий по журналам данного коммутатора в период с 25 сентября по 19 октября 2018 г. Объём трафика, передаваемого коммутатором, снизился с 15:00 13 октября; предоставляемые им услуги были затронуты с 22:15 13 октября вплоть до восстановления коммутатора в 1:16 14 октября. LogAnomaly породил 29 оповещений, все из которых начались с 15:59 13 октября и завершились до 1:16 14 октября. Таким образом, LogAnomaly успешно обнаружил данную аномалию,

не породив ни одного ложного срабатывания. DeepLog, напротив, породил 43 оповещения за период развёртывания (с 25 сентября по 19 октября), 15 из которых оказались ложными.

Очевидно, что в реальных условиях LogAnomaly достиг более высокой точности, чем DeepLog, по следующей причине. У данного провайдера облачных услуг насчитывается более 10 000 шаблонов журналов, причём значительное их число с различными индексами представляет весьма схожие события. DeepLog не способен использовать сходство шаблонов, тогда как LogAnomaly успешно извлёк это сходство, применяя `template2Vec` для освоения семантической информации шаблонов.

3.3 PLELog

Проблему недостатка размеченных данных было предложено посредством вероятностной оценки меток и разработали нейронную сеть на основе GRU с механизмом внимания для обнаружения аномалий [14].

На основе вероятностной оценки меток формируется размеченный обучающий набор, после чего PLELog строит модель обнаружения аномалий посредством машинного обучения с учителем, реализуя преимущества подобных методов. Кроме того, для повышения практичности PLELog должен хорошо работать с нестабильными журнальными данными и обнаруживать аномалии эффективно и результативно. Для удовлетворения первого критерия PLELog использует семантическое векторное представление журнальных последовательностей; для удовлетворения второго – разработана нейронная сеть GRU (Gated Recurrent Unit) с механизмом внимания.

PLELog извлекает структурированные события журнала из сообщений посредством парсинга. В частности, применяется современный метод Drain, доказавший высокую результативность и эффективность.

Для извлечения семантической информации событий журнала каждое событие рассматривается как предложение естественного языка. Поскольку события журнала разработаны программистами для фиксации состояния системы, большинство токенов представляют собой английские слова,

обладающие собственной семантикой. Наряду с ними встречаются несимвольные токены (разделители, операторы, знаки препинания, цифры) и составные токены – конкатенации слов (например, `NullPointerException`). Поэтому PLELog предварительно обрабатывает события, удаляя несимвольные токены и стоп-слова и разбивая составные токены на отдельные слова согласно «верблюжьей» нотации (Camel Case). Затем применяются предобученные на корпусе Common Crawl векторы слов, полученные алгоритмом FastText, для извлечения семантической информации каждого слова и преобразования его в d-мерный вектор ($d = 300$ для векторов FastText).

После преобразования слов в d-мерные векторы PLELog преобразует событие журнала в семантический вектор путём агрегации всех векторов слов с учётом важности каждого слова, измеряемой посредством TF-IDF. Частота термина $TF(w)$ определяется как отношение числа вхождений слова w к общему числу слов в событии; обратная частота документа $IDF(w) = \log(\#L / \#L_w)$, где $\#L$ – общее число событий журнала, а $\#L_w$ – число событий, содержащих w . Вес слова вычисляется как произведение TF и IDF. Итоговый семантический вектор события получается как взвешенное по TF-IDF среднее векторов всех слов. Таким образом, PLELog представляет события журнала семантическими векторами, что облегчает идентификацию семантически схожих событий и различение разнородных, обеспечивая устойчивость к нестабильности данных.

После семантического представления PLELog оценивает метки неразмеченных журнальных последовательностей обучающего набора на основе известных нормальных последовательностей, опираясь на идею PU-обучения. Интуитивно последовательности со схожей семантикой с большей вероятностью разделяют одну метку. Исходя из этого, PLELog сначала применяет продвинутую кластеризацию для отнесения семантически схожих последовательностей к одной группе. Поскольку получение безупречной кластеризации затруднительно, вместо детерминированной метки PLELog присваивает каждой неразмеченной последовательности вероятностную метку,

измеряя вероятность её принадлежности каждой метке для снижения влияния шума.

Поскольку журнальная последовательность содержит серию семантических векторов, для обеспечения кластеризации PLELog формирует для каждой последовательности единый вектор (семантический вектор последовательности) путём суммирования всех её семантических векторов. Применяется алгоритм HDBSCAN (иерархическая плотностная пространственная кластеризация приложений с шумом) для разбиения всех последовательностей обучающего набора на группы. Выбор HDBSCAN обоснован тремя причинами. Во-первых, заранее знать число кластеров затруднительно, в силу чего алгоритмы, требующие предопределения их числа (например, K-средних), неприменимы, тогда как HDBSCAN – плотностный алгоритм, не нуждающийся в этом. Во-вторых, HDBSCAN доказал результативность и эффективность и широко применяется во многих областях. В-третьих, HDBSCAN обладает небольшим числом параметров и устойчив к их выбору, что упрощает его практическое применение.

Посредством кластеризации семантически схожие последовательности относятся к одной группе, что указывает на общность их меток. Исходя из наличия в группе известных нормальных последовательностей, PLELog предварительно оценивает метки: если группа содержит известные нормальные последовательности, последовательности группы с большей вероятностью нормальны; в противном случае – с большей вероятностью аномальны. Поскольку получение безупречной кластеризации затруднительно, прямое присвоение детерминированной метки на основе предварительной оценки позволило бы шуму подорвать эффективность. Поэтому PLELog присваивает каждой неразмеченной последовательности вероятностную метку, измеряя вероятность её принадлежности каждой метке. Каждой последовательности в группе HDBSCAN присваивает оценку (score) в диапазоне от 0 до 1, отражающую неопределённость её принадлежности группе: чем ближе оценка к 0, тем выше уверенность в отнесении. На основе оценок HDBSCAN каждая

предварительная метка преобразуется в вероятностную: $P(\text{аномальная}) = 1 - \text{score}/2$, $P(\text{нормальная}) = \text{score}/2$.

На основе размеченного посредством вероятностной оценки обучающего набора разработана нейронная сеть GRU с механизмом внимания для построения результативной и эффективной модели обнаружения аномалий. На основе размеченного посредством вероятностной оценки обучающего набора разработана нейронная сеть GRU с механизмом внимания для построения результативной и эффективной модели обнаружения аномалий.

Выход GRU на t -м шаге определяется преимущественно текущим событием V_t и последним скрытым состоянием H_{t-1} . При большом числе событий в последовательности возникает проблема долгих зависимостей, способная снизить результативность обнаружения; кроме того, эффективность подрывают зашумлённые события. Для их преодоления в сеть включена стратегия маскирования на основе внимания. Интуитивно событие, сильнее коррелирующее с результатом обнаружения, важнее в последовательности: чем выше корреляция скрытого состояния H_i с результатом, тем больше его вес, указывающий на то, что e_i возможно является индикатором аномалии. Таким образом, важные события выделяются большими весами, а зашумлённые маскируются меньшими. Вектор внимания на t -м шаге собирается как взвешенная сумма скрытых состояний, где веса вычисляются функцией softmax . Итоговый результат для последовательности предсказывается нелинейным слоем с функцией активации $\tanh$.

PLELog успешно применён к двум крупномасштабным реальным системам двух различных организаций – сетевому центру университета и одной из ведущих мировых автомобилестроительных корпораций. Алгоритм достиг значений F1-меры 0,947 и 0,984 соответственно. PLELog так же демонстрирует высокие показатели на наборах HDFS и BGL: 0,96 и 0,98 соответственно.

3.4 LogRobust

Была применена предобученная модель Word2vec (FastText [34]) в сочетании с TF-IDF весами для получения векторных представлений шаблонов

журналов [10]. Полученные векторы подаются на вход двунаправленной LSTM-сети с механизмом внимания. Двунаправленная LSTM (Bi-LSTM) разделяет скрытый слой стандартной LSTM на два направления – прямой и обратный проходы, – что позволяет улавливать достаточную информацию входных последовательностей в обоих направлениях. Скрытые состояния прямого и обратного проходов на каждом шаге конкатенируются.

Поскольку различные события по-разному влияют на результат классификации, в Bi-LSTM вводится механизм внимания, присваивающий событиям различные веса. Тем самым снижается и влияние шума, поскольку шумным событиям, как правило, присваивается меньшая важность и меньшее внимание. Важность осваивается автоматически: к конкатенированному скрытому состоянию добавляется полносвязный слой (слой внимания), выходом которого является вес внимания, отражающий важность события (чем больше вес, тем больше внимания уделяется событию). Все скрытые состояния суммируются с учётом весов внимания, после чего слой softmax выдаёт результат классификации. На этапе обучения в качестве функции потерь используется перекрёстная энтропия, а параметры модели (веса Bi-LSTM и слоя внимания) обучаются методом стохастического градиентного спуска.

LogRobust устойчив к нестабильности событий журнала и достигает значений F-меры 0,99 на исходном наборе HDFS и 0,89–0,96 на синтетических наборах данных.

3.5 CNN

Была применена сверточная нейронная сеть (CNN) для обнаружения аномалий по журналам регистрации [15]. Журнальные записи группируются в сессии, затем преобразуются в обучаемую матрицу, которая подаётся на вход CNN-модели для бинарной классификации последовательности журнальных записей. CNN-модель достигает значения F-меры 0,98 на наборе HDFS.

Нейронные сети – биологически инспирированный подход к распознаванию образов. В обычных полносвязных сетях каждый нейрон связан со всеми нейронами предыдущего слоя, что плохо масштабируется для

высокоразмерных данных. Свёрточная нейронная сеть (CNN), вдохновлённая рецептивными полями зрительной коры кошки, предложена для улавливания локальной семантической информации вместо глобальной и преодоления переобучения.

Свёртка – базовая операция свёрточных слоёв, извлекающая признаки из локальных рецептивных полей карт признаков предыдущего слоя. В качестве функции активации применяется просачивающийся выпрямитель (Leaky ReLU), который позволяет избежать переобучения и решить проблему «мёртвых нейронов» (dead ReLU), устанавливая ненулевой (малый положительный) градиент для отрицательных входов.

Анализ журналов способен выигрывать от достижений в обработке естественного языка. Однако анализ журналов отличается от общей NLP: длинные связи широко встречаются в тексте, тогда как журналы содержат лишь небольшое число ключей; кроме того, цель обнаружения аномалий – поиск неожиданных путей выполнения (бинарная классификация), тогда как NLP классифицирует предложения по нескольким категориям.

В CNN применяются три однослойные свёртки (фильтры) параллельно: размером 3×128 , 4×128 и 5×128 соответственно. После трёх независимых свёрточных операций слой максимальной субдискретизации конкатенирует их выходы. Слой max-pooling также эффективно снижает переобучение, отфильтровывая слабо связанные признаки. Функция прореживания (dropout) применяется в предпоследнем слое в качестве регуляризации. В выходном слое используется функция softmax.

В данном исследовании перечисленные пять моделей систематически оцениваются и сравниваются. DeepLog и LogAnomaly используют подход на основе прогнозирования (предсказание следующего события журнала), тогда как PLELog, LogRobust и CNN – подход на основе классификации (определение принадлежности журнальной последовательности к нормальному или аномальному классу).

4 Программный модуль для анализа производительности ГО

В целях автоматизации проведения экспериментов в рамках заданных исследовательских задач был разработано программное решение для обнаружения аномалий в системных журналах. Проект реализует и унифицирует оценку производительности всех описанных в предыдущем разделе моделей: как обучение с учителем, так и обучение без учителя.

Обработка данных и обучение модели организованы в виде последовательного пайплайна, состоящего из восьми этапов. На первом этапе пользователь задаёт параметры эксперимента через аргументы командной строки или YAML-конфигурационный файл. Далее функция `process_dataset()` выполняет парсинг сырых лог-файлов и группировку строк по сессиям или скользящему окну. На третьем этапе строится или загружается из кэша словарь токенов (класс `Vocab`) вместе с предобученными векторными представлениями. Затем функция `sliding_window()` извлекает из данных признаки трёх типов: последовательные идентификаторы событий, количественные счётчики и семантические эмбединги. Сформированные тензоры упаковываются в `PyTorch Dataset` (`LogDataset` или `MaskedDataset` для `LogBERT`). На шестом этапе инициализируется нейронная сеть выбранной архитектуры в соответствии с конфигурацией `ModelConfig`. Класс `Trainer` проводит обучение с применением планировщика скорости обучения и градиентного накопления, после чего вычисляются итоговые метрики качества на тестовой выборке.

4.1 Входные данные

В текущей реализации программного анализатора поддерживаются пять интересных нас датасетов. Опишем формат исходных данных в целях унификации.

Каждый набор данных предполагает наличие нескольких файлов. Основной лог-файл содержит сырые строки журнала в формате, типичном для конкретной системы. Для датасета HDFS файл `anomaly_label.csv` хранит соответствие между идентификаторами блоков и метками `Normal/Anomaly`. Если предварительно был запущен алгоритм `Drain` для извлечения шаблонов, в

директории данных находятся файлы logTemplates.txt с текстами шаблонов и template{N}.txt с идентификаторами строк, соответствующих каждому шаблону. Для использования семантических векторов в директории dataset/ должен находиться файл nlp-word.vec с предобученными 300-мерными векторами слов в формате fastText.

4.2 Параметры запуска

Все параметры передаются через CLI (helpers.py) или YAML-конфиг файл. Описание параметров приведено в таблице 2.

Таблица 2 – Параметры запуска анализа

Параметр	Тип	Значение по умолчанию	Описание
model_name	str	DeepLog	Выбор модели: DeepLog, LogAnomaly, LogRobust, CNN, PLELog
dataset_name	str	HDFS	Набор данных: HDFS, BGL, Thunderbird, Spirit
data_dir	str	./dataset/	Директория с исходными данными
output_dir	str	./output	Директория для сохранения результатов и моделей
grouping	str	session	Метод группировки: session (по сессии) или sliding (скользящее окно)
window_size	int	5	Размер окна (в записях или минутах)
step_size	int	1	Шаг скользящего окна
history_size	int	10	Длина истории событий для одного примера
train_size	float	0.4	Доля данных для обучения

Продолжение таблицы 2

Параметр	Тип	Значение по умолчанию	Описание
valid_ratio	float	0.1	Доля обучающих данных для валидации
batch_size	int	32	Размер батча
max_epoch	int	200	Максимальное число эпох обучения
lr	float	0.01	Начальная скорость обучения
optimizer	str	adam	Тип оптимизатора: adam, sgd, adamw, adagrad, adadelata, rmsprop
scheduler	str	linear	LR-планировщик: linear, cosine, polynomial, constant
hidden_size	int	128	Размер скрытого состояния LSTM
num_layers	int	2	Количество слоёв LSTM
embedding_dim	int	50	Размерность эмбедингов токенов
dropout	float	0.1	Вероятность dropout
topk	int	9	Топ-k порог для unsupervised моделей
sequential	flag	False	Использовать последовательные вектора (log key IDs)
quantitative	flag	False	Использовать количественные вектора (счётчики событий)
semantic	flag	False	Использовать семантические вектора (word2vec/fastText)

Продолжение таблицы 2

Параметр	Тип	Значение по умолчанию	Описание
is_chronological	flag	False	Хронологическое разбиение train/test
remove_duplicates	flag	False	Удалить дублирующиеся последовательности из обучения
config_file	str	None	Путь к YAML-конфигу (перекрывает CLI-аргументы)

Пример скрипта для запуска анализатора на рисунке 3.

```

1  # DeepLog on HDFS dataset
2  python main_run.py --data_dir ./dataset/HDFS --output_dir ./output/ --model_name DeepLog
   --dataset_name HDFS --grouping session --window_size 20 --step_size 1 --train_size 0.01
   --is_chronological --session_level entry --log_file HDFS.log --batch_size 2048 --lr 0.001
   --accumulation_step 1 --optimizer adamw --sequential --history_size 10 --embeddings
   embeddings_average.json --hidden_size 128 --num_layers 2 --embedding_dim 300 --topk 10 --dropout 0.1
   --max_epoch 10
3

```

Рисунок 3 – Пример скрипта

4.3 Основные компоненты

В таблице 3 приведено описание основных структурных элементов проекта.

Таблица 3 – Основные компоненты

Модуль / Файл	Назначение
main_run.py	Точка входа. Оркестрация: разбор аргументов, построение словаря, выбор модели, запуск обучения и оценки
trainer.py	Класс Trainer. Управление циклом обучения, валидацией, сохранением/загрузкой моделей, вычислением метрик

Продолжение таблицы 3

Модуль / Файл	Назначение
helpers.py	arg_parser(): все CLI-аргументы; get_optimizer(): фабрика оптимизаторов
data/DataLoader.py	Загрузка и предобработка данных. Батчинг, итерация по данным
data/Instance.py	Классы Instance, LogEntry, StepLog. Представление одного лог-экземпляра
data/TensorInstances.py	TInstWithLogits, TInstWithoutLogits. Тензорные батчи для подачи в модели
data/Vocab.py	Класс Vocab. Словарь токенов и меток, загрузка предобученных эмбеддингов
data/Embedding.py	Вычисление TF-IDF взвешенных эмбеддингов шаблонов событий из fastText векторов
models/__init__.py	get_model(): фабричный метод для создания любой поддерживаемой модели
models/lstm.py	DeepLog, LogRobust, LogAnomaly – LSTM-архитектуры
models/cnn.py	CNN: свёрточная нейронная сеть для классификации
models/plelog.py	Обёртка для внешнего PLELog модуля

4.4 Выходные данные

Все артефакты эксперимента сохраняются в директории, указанной параметром `--output_dir`. Путь автоматически расширяется исходя из параметров эксперимента: для скользящего окна он включает размер окна, шаг, признак хронологического выбора данных и долю обучения; для сессионного режима – только долю обучения. Такая схема именования гарантирует, что результаты разных конфигураций не перезаписывают друг друга. В таблице 4 описана структура директории.

Таблица 4 – Структура директории для записи выходных данных

Путь	Содержимое
<code>{output_dir}/{dataset}/session/train{size}/</code>	Корень эксперимента с сессионной группировкой
<code>{output_dir}/{dataset}/sliding/</code>	Корень эксперимента со скользящим окном
<code>{output_dir}/models/{model_name}.pt</code>	Чекпоинт модели: веса сети, состояние оптимизатора и планировщика
<code>{output_dir}/vocabs/{model_name}.pkl</code>	Сериализованный словарь Vocab (формат pickle)
<code>{output_dir}/train.pkl</code>	Предобработанные обучающие данные (формат pickle)
<code>{output_dir}/test.pkl</code>	Предобработанные тестовые данные (формат pickle)
<code>dataset/templates/templates_{dataset}.vec</code>	Кэш векторных представлений шаблонов событий
<code>dataset/templates/idf</code>	IDF-веса токенов, вычисленные по корпусу шаблонов

5 Результаты и основные выводы

Недавние исследования [35, 13] показали, что подходы на основе глубокого обучения могут достигать высокой точности (F-мера выше 90%) на часто используемых датасетах (например, HDFS). Эти результаты, казалось бы, свидетельствуют о том, что проблема обнаружения аномалий может быть решена почти идеально с помощью глубокого обучения. Для исследования реальной эффективности существующих моделей ГО мы оцениваем их производительность согласно поставленным задачам.

Прежде чем приступить к экспериментам, необходимо определить систему оценки, в рамках которой будут анализироваться результаты. Поскольку задача обнаружения аномалий в логах представляет собой задачу бинарной классификации – каждая последовательность событий относится либо к нормальному, либо к аномальному классу – для измерения эффективности моделей применяются классические метрики, основанные на матрице ошибок. Матрица ошибок описывает четыре возможных исхода классификации. Истинно положительный результат (TP, True Positive) означает, что модель правильно идентифицировала аномальную последовательность как аномальную. Ложноположительный (FP, False Positive) – что нормальная последовательность была ошибочно отнесена к аномальным, то есть произошло ложное срабатывание. Истинно отрицательный (TN, True Negative) – нормальная последовательность корректно классифицирована как нормальная. Наконец, ложноотрицательный (FN, False Negative) – аномальная последовательность не была обнаружена и ошибочно признана нормальной, то есть произошёл пропуск аномалии. На основе этих четырёх величин вычисляются следующие метрики [39].

Точность (Precision, Prec) показывает, насколько модель точна в своих положительных предсказаниях, то есть какова доля действительно аномальных последовательностей среди всех тех, что модель отнесла к аномальным: $Prec = TP / (TP + FP)$. Высокая точность означает, что модель редко ошибочно помечает нормальные последовательности как аномальные. Это особенно важно в

условиях, когда ложные срабатывания влекут за собой значительные операционные издержки – например, когда каждое предупреждение требует ручной проверки специалистом.

Полнота (Recall, Rec) характеризует способность модели обнаруживать реальные аномалии, то есть показывает, какую долю от всех существующих аномалий модель выявила: $Rec = TP / (TP + FN)$. Высокая полнота означает, что модель пропускает лишь незначительную часть аномалий. Данная метрика критически важна в сценариях, где пропуск аномалии – например, сбой в критически важной системе – может иметь серьёзные последствия.

Специфичность (Specificity, Spec) дополняет полноту, оценивая качество классификации с противоположной стороны – со стороны нормального класса. Она показывает, какую долю всех нормальных последовательностей модель корректно распознала как нормальные: $Spec = TN / (TN + FP)$. Высокая специфичность свидетельствует о том, что модель редко принимает нормальное поведение системы за аномальное, то есть уровень ложных срабатываний низок.

F-мера (F1-Score) представляет собой среднее гармоническое точности и полноты и служит единой оценкой качества модели: $F1 = 2 \cdot Prec \cdot Rec / (Prec + Rec)$. В отличие от арифметического среднего, среднее гармоническое штрафует за сильную разбалансированность между точностью и полнотой: модель с высокой точностью, но крайне низкой полнотой (или наоборот) получит низкое значение F1. Таким образом, F-мера поощряет модели, способные достигать хорошего баланса между двумя метриками. Это особенно актуально при работе с несбалансированными датасетами.

Совокупное использование всех четырёх метрик позволяет получить полную картину поведения модели: F1 отражает общий баланс между точностью и полнотой, специфичность характеризует уровень ложных срабатываний, а отдельный анализ точности и полноты позволяет понять, в каком направлении модель ошибается чаще – пропускает аномалии или генерирует лишние предупреждения[40].

5.1 Производительность при различных стратегиях выбора обучающих данных

Результаты обнаружения аномалий могут зависеть от выбора обучающих данных. В некоторых исследованиях (DeepLog [6], LogRobust [10], PLELog [14], CNN [15]) обучающие и тестовые данные выбираются на основе временных меток логов – хронологическая стратегия. В других работах [41, 42, 13] применяются фиксированные/скользящие окна для группировки событий логов в последовательности, после чего все последовательности перемешиваются перед разделением на обучающую и тестовую выборки – используется случайная стратегия. Случайная стратегия позволяет моделям видеть больше событий и достигать высокой точности, однако может приводить к проблеме утечки данных на этапе обучения.

Как существующие подходы работают при различных стратегиях выбора обучающих данных? Сравним две стратегии: случайный выбор – последовательности логов перемешиваются и разделяются на обучающую и тестовую выборки в соотношении 80:20, и хронологический выбор – первые 80% исходных логов используются для обучения, оставшиеся 20% – для тестирования. Ниже представлены таблицы 5 – 7 с результатами эксперимента, записанными в формате «случайная стратегия/хронологическая».

Таблица 5 – Сравнение качества моделей на датасете BGL

Модель	Prec	Rec	Spec	F1
DeepLog	0.952/0.270	0.903/0.988	0.990/0.437	0.927/0.426
LogAnomaly	0.961/0.313	0.903/0.798	0.992/0.551	0.931/0.483
PLELog	0.963/0.702	0.935/0.791	0.992/0.899	0.949/0.744
LogRobust	0.972/0.994	0.984/0.942	0.995/0.999	0.981/0.967
CNN	0.994/0.871	0.963/0.947	0.999/0.970	0.978/0.908

Таблица 6 – Сравнение качества моделей на датасете Spirit

Модель	Prec	Rec	Spec	F1
DeepLog	0.867/0.438	1.0/1.0	0.386/0.090	0.929/0.609

Продолжение таблицы 6

Модель	Prec	Rec	Spec	F1
LogAnomaly	0.882/0.438	1.0/1.0	0.456/0.090	0.937/0.609
PLELog	0.956/0.931	0.974/0.767	0.818/0.690	0.965/0.841
LogRobust	0.994/0.985	0.978/0.915	0.979/0.990	0.986/0.949
CNN	1.0/0.986	1.0/1.0	1.0/0.990	1.0/0.993

Таблица 7 – Сравнение качества моделей на датасете Thunderbird

Модель	Prec	Rec	Spec	F1
DeepLog	0.232/0.200	1.0/1.0	0.007/0	0.369/0.333
LogAnomaly	0.234/0.229	1.0/1.0	0.014/0.156	0.371/0.371
PLELog	0.584/0.250	0.344/1.0	0.729/0.250	0.414/0.400
LogRobust	0.803/0.900	0.921/1.0	0.931/0.969	0.941/0.947
CNN	1.0/0.889	0.875/1.0	1.0/0.969	0.933/0.941

Очевидно, что для прогнозирующих моделей (LogAnomaly, DeepLog, PLELog) результаты со случайным выбором значительно лучше, чем с хронологическим. Например, DeepLog достигает F-меры 0,927 со случайным выбором обучающих данных на датасете BGL. При разделении обучающей и тестовой выборки по временному порядку (хронологический выбор) F-мера падает до 0,426. Причина в том, что при случайном выборе модели могут видеть будущие события логов на этапе обучения (утечка данных), что позволяет делать более точные предсказания. Кроме того, DeepLog и LogAnomaly обучают модели, используя индексы событий логов, и игнорируют семантическое значение логов.

Модели с учителем (LogRobust и CNN) работают значительно лучше остальных при обеих стратегиях, поскольку обучаются на большом объеме нормальных и аномальных данных. Например, эти две модели достигают F-меры

около 0,94 на датасете Thunderbird с хронологической настройкой, тогда как другие показывают низкие результаты.

Делаем вывод, что стратегии выбора обучающих данных оказывают значительное влияние на прогнозирующие модели обнаружения аномалий. Хотя случайная стратегия приводит к лучшим результатам, она может вызвать проблему утечки данных и не позволяет оценить эффективность моделей в реальных сценариях.

5.2 Производительность при различных методах группировки данных

Мы обучаем модели на трёх датасетах, используя хронологический выбор с фиксированным окном различных размеров (20, 100, 200 сообщений и 0,5 часа). Результаты показывают, что производительность всех моделей существенно варьируется в зависимости от размера окна. С увеличением размера окна результаты, как правило, улучшаются, но остаются нестабильными, что отражено в таблица 8 – 10.

Таблица 8 – Сравнение качества моделей на датасете BGL

Модель	Метрики	20 сообщ.	100 сообщ.	200 сообщ.	0.5 часа
DeepLog	Prec	0.128	0.166	0.192	0.209
	Rec	0.995	0.988	0.987	0.984
	Spec	0.539	0.53	0.528	0.481
	F1	0.227	0.285	0.322	0.345
LogAnomaly	Prec	0.136	0.176	0.203	0.276
	Rec	0.97	0.985	0.985	0.973
	Spec	0.581	0.562	0.559	0.643
	F1	0.239	0.299	0.336	0.43
PLELog	Prec	0.592	0.595	0.862	0.76
	Rec	0.882	0.88	0.844	0.785
	Spec	0.958	0.968	0.985	0.965

Продолжение таблицы 8

Модель	Метрики	20 сообщ.	100 сообщ.	200 сообщ.	0.5 часа
	F1	0.708	0.71	0.853	0.772
LogRobust	Prec	0.616	0.696	0.684	0.819
	Rec	0.969	0.968	0.963	0.946
	Spec	0.959	0.96	0.949	0.971
	F1	0.753	0.81	0.8	0.878
CNN	Prec	0.634	0.698	0.744	0.837
	Rec	0.969	0.965	0.965	0.935
	Spec	0.962	0.96	0.949	0.974
	F1	0.767	0.81	0.84	0.883

Таблица 9 – Сравнение качества моделей на датасете Spirit

Модель	Метрики	20 сообщ.	100 сообщ.	200 сообщ.	0.5 часа
DeepLog	Prec	0.504	0.5	0.175	0.291
	Rec	0.776	0.861	0.985	1
	Spec	0.986	0.969	0.747	0.068
	F1	0.611	0.633	0.298	0.45
LogAnomaly	Prec	0.498	0.508	0.198	0.33
	Rec	0.773	0.87	0.981	1
	Spec	0.986	0.97	0.783	0.225
	F1	0.606	0.642	0.33	0.496
PLELog	Prec	0.375	0.371	0.141	0.516
	Rec	0.286	0.824	0.552	0.663
	Spec	0.991	0.95	0.816	0.763
	F1	0.325	0.511	0.225	0.581
LogRobust	Prec	0.947	0.943	0.751	0.989
	Rec	0.979	0.954	0.965	0.989
	Spec	0.999	0.998	0.982	0.996

Продолжение таблицы 9

Модель	Метрики	20 сообщ.	100 сообщ.	200 сообщ.	0.5 часа
LogRobust	F1	0.963	0.947	0.845	0.989
CNN	Prec	0.954	0.959	0.961	0.989
	Rec	0.966	0.948	0.95	0.989
	Spec	0.999	0.999	0.998	0.996
	F1	0.96	0.953	0.955	0.989

Таблица 10 – Сравнение качества моделей на датасете Thunderbird

Модель	Метрики	20 сообщ.	100 сообщ.	200 сообщ.	0.5 часа
DeepLog	Prec	0.004	0.017	0.005	0.162
	Rec	0.938	0.963	1	1
	Spec	0.899	0.922	0.005	0.088
	F1	0.008	0.033	0.01	0.279
LogAnomaly	Prec	0.004	0.025	0	0.154
	Rec	0.938	0.963	1	1
	Spec	0.891	0.95	0.005	0.029
	F1	0.008	0.05	0.009	0.267
PLELog	Prec	0.429	0.826	0.692	1
	Rec	0.688	0.704	0.36	0.5
	Spec	1	1	1	1
	F1	0.528	0.76	0.474	0.667
LogRobust	Prec	0.377	0.318	0.289	0.458
	Rec	0.876	1	0.96	0.917
	Spec	0.999	0.997	0.994	0.809
	F1	0.531	0.482	0.444	0.611
CNN	Prec	0.907	0.9	0.72	1
	Rec	0.813	0.67	0.72	0.667
	Spec	1	1	0.999	1

	F1	0.857	0.766	0.72	0.8
--	----	-------	-------	------	-----

Далее оценивается производительность моделей при группировке сессионными окнами на датасетах HDFS и BGL, используя `block_id` и `node_id` соответственно. Данный исследовательский вопрос не оценивался на других наборах данных (Spirit и Thunderbird), поскольку в их журнальных записях отсутствуют идентификаторы. Результаты эксперимента отражены в таблице 11.

Таблица 11 – Сравнение качества моделей при сессионной группировке

Датасет	BGL				HDFS			
Модель/Метрики	Prec	Rec	Spec	F1	Prec	Rec	Spec	F1
DeepLog	0.997	1	0.997	0.998	0.835	0.994	0.994	0.908
LogAnomaly	0.997	1	0.998	0.999	0.886	0.893	0.961	0.966
PLELog	0.995	0.992	0.996	0.994	0.893	0.979	0.996	0.934
LogRobust	1	1	1	1	0.961	1	0.989	0.98
CNN	1	1	1	1	0.966	1	0.991	0.982

Результаты показывают, что все модели достигают высокой производительности при группировке по сессионным окнам (все значения F-меры выше 0,9).

Из полученных результатов следует, что группировки данных могут оказывать значительное влияние на модели обнаружения аномалий. При фиксированном окне модели работают нестабильно при различных размерах окна. Группировка по сессионным окнам может приводить к лучшим результатам.

5.3 Производительность при различных распределениях классов

Ранее была доказана ограниченность традиционных метрик. На наборе Spirit DeepLog и LogAnomaly достигают высоких значений точности, полноты и F-меры ($> 0,86$) при случайном отборе, однако специфичность остаётся низкой ($< 0,5$), что свидетельствует о массовой ложной классификации нормальных записей как аномальных. Это является следствием сильного дисбаланса классов.

Сильный дисбаланс классов существенно усложняет задачу обнаружения аномалий. Для систематической оценки смоделированы различные сценарии дисбаланса путём случайного удаления нормальных или аномальных последовательностей с целью получения соотношений аномалий 0,1%, 0,5%, 1%, 5%, 10% и 15%. Результаты отражены в таблицах 12 и 13.

Таблица 12 – Сравнение качества моделей на датасете HDFS

Модель	Метрики	0.1%	0.5%	1%	5%	10%	15%
DeepLog	Prec	0.942	0.562	0.638	0.879	0.941	0.962
	Rec	0.485	0.894	0.995	0.997	0.994	0.994
	Spec	1	0.997	0.994	0.993	0.994	0.994
	F1	0.64	0.69	0.777	0.935	0.967	0.978
LogAnomaly	Prec	0.975	0.656	0.732	0.896	0.95	0.972
	Rec	0.485	0.905	0.946	0.993	0.994	0.993
	Spec	1	0.998	0.996	0.994	0.995	0.995
	F1	0.647	0.76	0.825	0.942	0.971	0.983
PLELog	Prec	0.947	0.91	0.934	0.972	0.974	0.974
	Rec	0.972	0.69	0.794	0.956	1	1
	Spec	1	0.999	0.999	0.999	0.996	0.996
	F1	0.786	0.785	0.859	0.964	0.987	0.987
LogRobust	Prec	0.519	0.673	0.794	0.926	0.945	0.964
	Rec	0.507	0.646	0.999	1	1	1
	Spec	0.999	0.998	0.997	0.996	0.994	0.994
	F1	0.513	0.659	0.885	0.962	0.972	0.982
CNN	Prec	0.945	0.675	0.793	0.92	0.953	0.967
	Rec	0.388	0.879	0.947	1	0.999	0.999
	Spec	1	0.998	0.997	0.996	0.995	0.995
	F1	0.55	0.763	0.863	0.959	0.976	0.983

Таблица 13 – Сравнение качества моделей при различных распределениях классов на датасете BGL

Модель	Метрики	0.1%	0.5%	1%	5%	10%	15%
DeepLog	Prec	0.124	0.156	0.184	0.237	0.276	0.278
	Rec	0.987	0.969	0.991	1	0.988	0.959
	Spec	0.424	0.424	0.417	0.407	0.435	0.44
	F1	0.221	0.268	0.311	0.383	0.431	0.432
LogAnomaly	Prec	0.146	0.183	0.213	0.274	0.321	0.338
	Rec	0.987	0.979	0.957	0.961	0.971	0.924
	Spec	0.523	0.521	0.53	0.534	0.552	0.593
	F1	0.255	0.308	0.348	0.427	0.482	0.495
PLELog	Prec	0.583	0.564	0.617	0.765	0.786	0.895
	Rec	0.691	0.587	0.706	0.872	0.883	0.817
	Spec	0.915	0.95	0.919	0.942	0.946	0.987
	F1	0.632	0.576	0.659	0.815	0.832	0.855
LogRobust	Prec	0.5	0.506	0.632	0.787	0.851	0.819
	Rec	0.84	0.856	0.896	0.941	0.959	0.924
	Spec	0.93	0.908	0.969	0.953	0.963	0.954
	F1	0.627	0.636	0.741	0.857	0.902	0.868
CNN	Prec	0.911	0.974	0.901	0.818	0.86	0.865
	Rec	0.68	0.784	0.791	0.922	0.93	0.901
	Spec	0.994	0.998	0.988	0.963	0.967	0.968
	F1	0.779	0.869	0.843	0.868	0.894	0.883

Результаты показывают, что по мере увеличения доли аномалий производительность моделей улучшается. Например, показатели LogRobust на наборе BGL улучшаются на 63,8%, 10,1%, 2,6% и 38,4% по точности, полноте, специфичности и F-мере соответственно при переходе от доли аномалий 0,1% к 15%.

5.4 Производительность при различной степени зашумлённости данных

5.4.1. Влияние ошибочной разметки

Для оценки влияния ошибочной разметки в обучающую выборку вводились ошибочно размеченные образцы (от 1% до 10%). Для моделей обучения с подкреплением в обучающую выборку добавлялись аномальные записи; для обучаемых с учителем – метки некоторых аномалий менялись на «нормальная». На рисунке 4 представлен график, отражающий результат эксперимента.

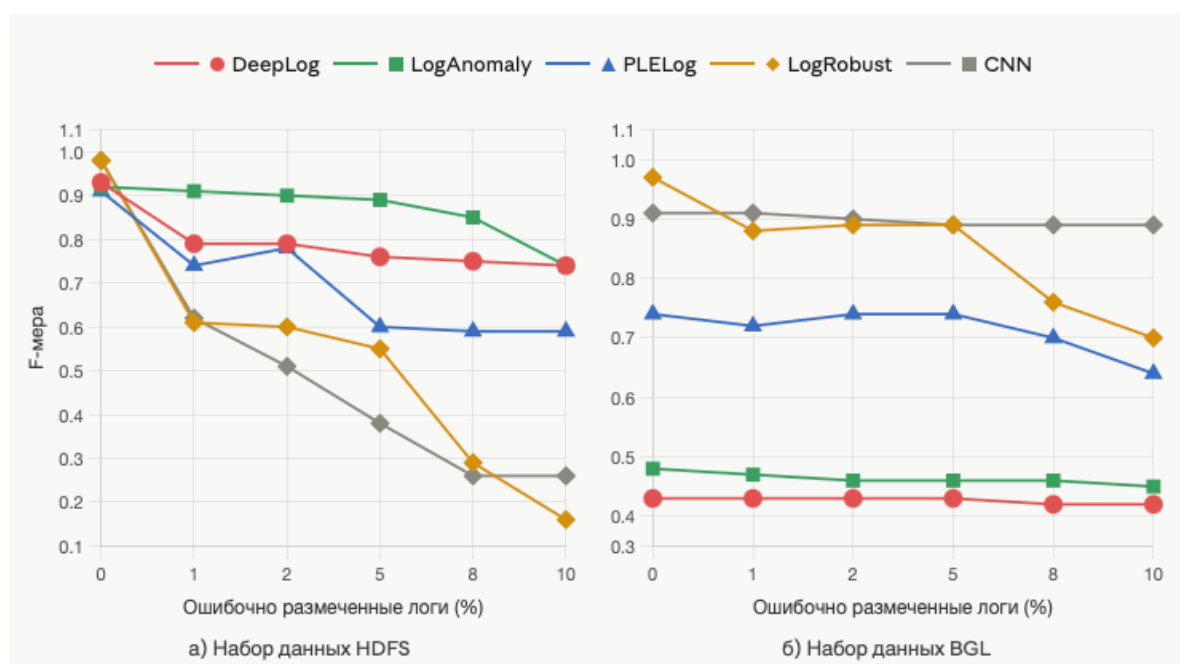


Рисунок 4 – Влияние ошибочной разметки

На датасете HDFS даже при 1% ошибочно размеченных логов F-меры исследуемых моделей значительно снижаются. Это связано с природой датасета: много дублирующихся последовательностей. Ошибочная маркировка одной такой последовательности автоматически затрагивает все её копии, что вызывает каскадное ухудшение. LogRobust и CNN падают на 37,2% и 36,2% соответственно. При уровне шума 10% F-меры LogRobust и CNN падают до 0,161 и 0,263. Интересно, что F-мера LogAnomaly снижается лишь незначительно при 1% ошибочно размеченных логов (с 0,915 до 0,897). При доле ошибочно размеченных записей 5% и 10% LogAnomaly достигает более высоких значений

F-меры по сравнению с остальными моделями. Причина состоит в том, что LogAnomaly использует количественные векторы для извлечения количественных зависимостей, присутствующих в журналах, наряду с последовательными векторами, что позволяет модели точнее предсказывать вероятности следующего события.

Можно сделать вывод, что небольшое количество ошибочно размеченных логов может быстро ухудшить производительность обнаружения аномалий. Модели, обучаемые с учителем, более чувствительны к ошибочной разметке. Модели, использующие подход на основе прогнозирования (DeepLog и LogAnomaly), работают лучше при наличии ошибочно размеченных логов.

5.4.2. Влияние ошибок парсинга логов

Оценивалось влияние шума, вносимого ошибками парсинга. Эксперименты проводились с четырьмя широко используемыми парсерами: Drain [26], Spell [43], AEL [27] и IPLoM [28]. Результаты в таблицах 14 – 17.

Таблица 14 – Сравнение влияния инструмента парсинга на датасете HDFS

Модель	Метрики	Drain	Spell	AEL	IPLoM
DeepLog	Prec	0.835	0.852	0.903	0.889
	Rec	0.994	0.981	0.934	0.791
	Spec	0.994	0.995	0.997	0.997
	F1	0.908	0.912	0.918	0.837
LogAnomaly	Prec	0.886	0.844	0.839	0.863
	Rec	0.993	0.897	0.983	0.955
	Spec	0.961	0.995	0.994	0.995
	F1	0.966	0.87	0.906	0.906
PLELog	Prec	0.893	0.953	0.967	0.913
	Rec	0.979	0.199	0.8	0.781
	Spec	0.996	1	0.999	0.994
	F1	0.934	0.329	0.864	0.842
LogRobust	Prec	0.961	0.938	0.896	0.93

Продолжение таблицы 14

Модель	Метрики	Drain	Spell	AEL	IPLoM
LogRobust	Rec	1	0.999	1	1
	Spec	0.989	0.998	0.997	0.998
	F1	0.98	0.968	0.945	0.964
CNN	Prec	0.966	0.947	0.944	0.952
	Rec	1	1	1	0.999
	Spec	0.991	0.998	0.998	0.998
	F1	0.982	0.973	0.971	0.975

Таблица 15 – Сравнение влияния инструмента парсинга на датасете BGL

Модель	Метрики	Drain	Spell	AEL	IPLoM
DeepLog	Prec	0.27	0.271	0.271	0.273
	Rec	0.988	0.988	0.988	0.988
	Spec	0.437	0.44	0.438	0.443
	F1	0.426	0.426	0.425	0.427
LogAnomaly	Prec	0.313	0.339	0.539	0.548
	Rec	0.798	0.977	0.965	0.971
	Spec	0.551	0.599	0.826	0.831
	F1	0.483	0.504	0.692	0.7
PLELog	Prec	0.702	0.56	0.661	0.655
	Rec	0.791	0.404	0.854	0.433
	Spec	0.899	0.938	0.907	0.952
	F1	0.744	0.476	0.744	0.521
LogRobust	Prec	0.994	0.849	0.844	0.726
	Rec	0.942	0.988	0.982	0.977
	Spec	0.999	0.963	0.962	0.922
	F1	0.967	0.914	0.908	0.833
CNN	Prec	0.871	0.994	0.942	0.937

Продолжение таблица 15

Модель	Метрики	Drain	Spell	AEL	IPLoM
CNN	Rec	0.947	0.965	0.947	0.959
	Spec	0.97	0.999	0.988	0.986
	F1	0.908	0.979	0.945	0.948

Таблица 16 – Сравнение влияния инструмента парсинга на датасете Spirit

Модель	Метрики	Drain	Spell	AEL	IPLoM
DeepLog	Prec	0.438	0.602	0.413	0.607
	Rec	1	1	1	1
	Spec	0.099	0.535	0	0.545
	F1	0.609	0.751	0.584	0.755
LogAnomaly	Prec	0.438	0.612	0.413	0.607
	Rec	1	1	1	1
	Spec	0.099	0.554	0	0.545
	F1	0.609	0.759	0.584	0.755
PLELog	Prec	0.931	0.859	0.863	0.5
	Rec	0.767	0.859	0.887	0.662
	Spec	0.69	0.901	0.901	0.535
	F1	0.841	0.859	0.875	0.57
LogRobust	Prec	0.985	0.947	0.986	0.973
	Rec	0.915	1	1	1
	Spec	0.99	0.96	0.99	0.98
	F1	0.949	0.973	0.993	0.986
CNN	Prec	0.986	0.959	0.986	0.986
	Rec	1	1	1	1
	Spec	0.99	0.97	0.99	0.99
	F1	0.993	0.979	0.993	0.993

Таблица 17 – Сравнение влияния инструмента парсинга на датасете Thunderbird

Модель	Метрики	Drain	Spell	AEL	IPLoM
DeepLog	Prec	0.2	0.211	0.2	0.2
	Rec	1	1	1	1
	Spec	0	0.02	0	0
	F1	0.333	0.348	0.333	0.333
LogAnomaly	Prec	0.229	0.337	0.546	0.55
	Rec	1	0.982	0.977	0.971
	Spec	0.156	0.591	0.828	0.832
	F1	0.371	0.541	0.7	0.702
PLELog	Prec	0.25	1	0.4	1
	Rec	1	0.125	0.5	0.5
	Spec	0.25	1	0.8125	1
	F1	0.4	0.222	0.444	0.667
LogRobust	Prec	0.9	0.805	0.852	0.862
	Rec	1	0.988	0.982	0.982
	Spec	0.969	0.949	0.964	0.967
	F1	0.947	0.887	0.913	0.918
CNN	Prec	0.889	0.965	0.931	0.921
	Rec	1	0.971	0.947	0.959
	Spec	0.969	0.993	0.985	0.983
	F1	0.941	0.968	0.939	0.94

Производительность моделей существенно варьируется от используемого инструмента. Например, DeepLog достигает F-меры 0,755 с парсером IPLoM на датасете Spirit, но при использовании Drain значение падает до 0,609. Результаты также показывают, что LogRobust и CNN лучше справляются с шумом парсинга благодаря использованию семантических векторов.

Следовательно, шум данных от ошибок парсинга логов влияет на производительность моделей. Лишние события логов могут быстро ухудшить

производительность моделей на основе прогнозирования. Методы, использующие семантические векторы, лучше справляются с ошибками парсинга.

5.5 Раннее обнаружение аномалий

Для решения последней исследовательской задачи фиксировалось число просмотренных записей журнала до момента, когда каждая модель поднимала оповещение об аномалии в режиме онлайн-детектирования (обнаружение «на лету»). Распределение значений наблюдений отражены на рисунке 5.

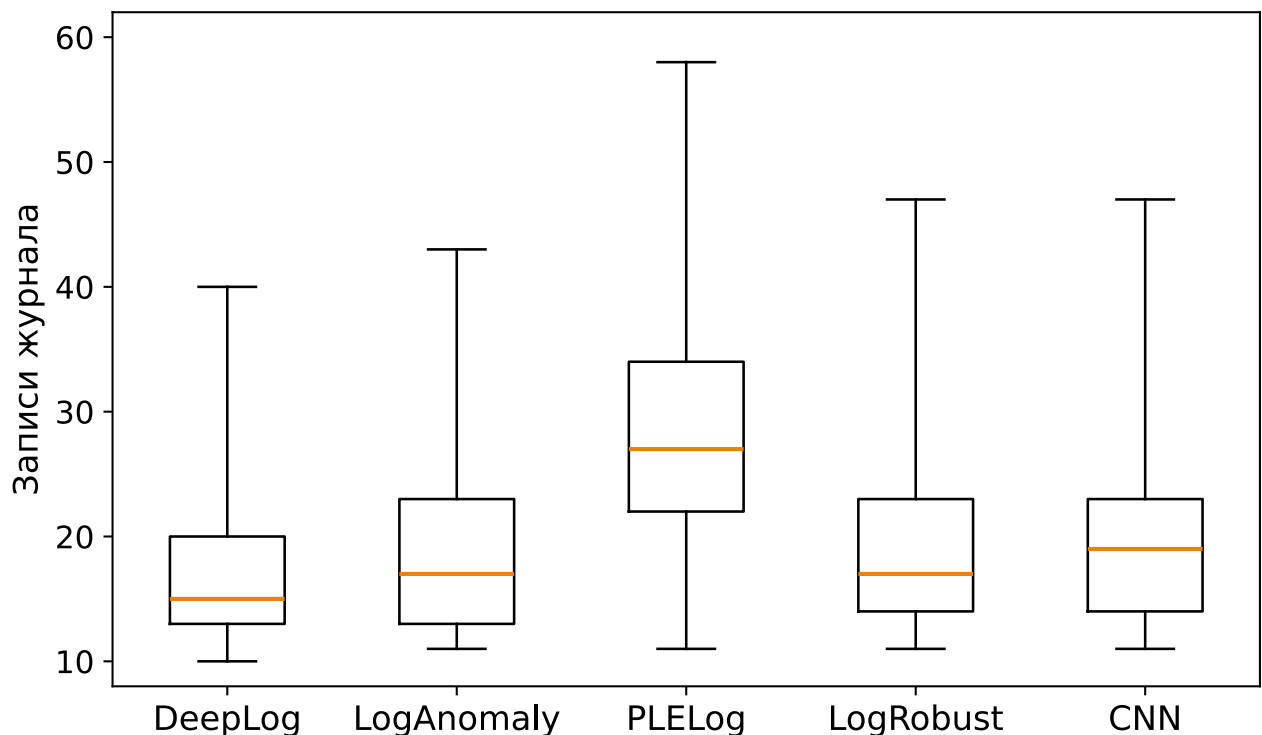


Рисунок 5 – Число просмотренных записей до обнаружения аномалии

Было обнаружено, что DeepLog обнаруживает аномалии раньше всех – в среднем после 14,3 и записей. LogAnomaly обнаруживает аномалии несколько позже, поскольку требует захвата как последовательных, так и количественных отношений. Классификационные модели (LogRobust, CNN и PLELog), опирающиеся на семантические векторы, обнаруживают аномалии значительно позже, поскольку должны накопить достаточно признаков аномального поведения для надёжной классификации. В типичной последовательности из 100

записей на наборе BGL DeepLog поднимает оповещение после 11 записей, PLELog – после 64.

С точки зрения вычислительной эффективности на наборе BGL DeepLog тратит 19,2 и 0,6 мин на обучение и тестирование. LogAnomaly – 56,7 и 1,6 мин. PLELog является наиболее ресурсоёмкой моделью: 36,6 мин на кластеризацию + 69,1 мин на обучение GRU-сети, 10,8 мин на тестирование. PLELog может оказаться непригодным для режима онлайн-детектирования. На рисунке 6 наглядно изображен таймлайн обнаружения аномалии.

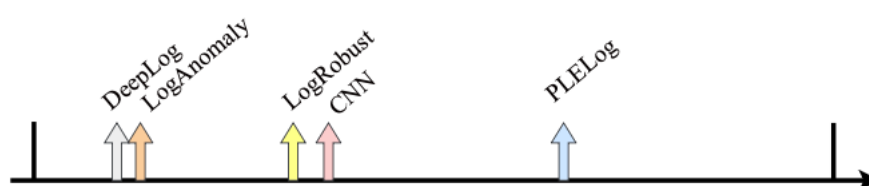


Рисунок 6 – Хронология обнаружения аномалии

Очевидно, что прогнозирующие модели (DeepLog и LogAnomaly) обнаруживают аномалии раньше классификационных (PLELog, LogRobust и CNN).

6 Обсуждение

6.1 Преимущества и недостатки исследованных методов

На основании полученных результатов можно заключить, что ни одна из исследованных моделей не всегда работает столь же хорошо, как заявлено в исходных публикациях. Различные сценарии по-разному влияют на эффективность моделей обнаружения аномалий. Ниже мы указываем преимущества и недостатки каждой модели.

6.1.1 DeepLog

Основное преимущество DeepLog состоит в том, что для построения модели обнаружения на основе последовательных векторов ей не требуются аномальные записи журнала, что снижает трудозатраты на построение модели. Кроме того, она способна обнаруживать аномалии раньше других. Однако вследствие этой же особенности DeepLog демонстрирует низкую эффективность на более сложных наборах данных с большим числом событий журнала (см. раздел 4.1). Помимо этого, DeepLog сильно подвержена влиянию ошибок парсера, поскольку использует лишь индекс шаблонов журналов и игнорирует их семантический смысл (см. раздел 4.4).

6.1.2 LogAnomaly

LogAnomaly использует последовательные и количественные векторы для обучения модели обнаружения, что позволяет снизить влияние зашумлённости данных, вызванной ошибочно размеченными записями журнала. Подобно DeepLog, LogAnomaly способна обнаруживать аномалии на ранней стадии и работать с большими объёмами данных, поскольку обучается только на нормальных записях журнала. Основное достоинство LogAnomaly проявляется на этапе сопоставления схожих шаблонов журналов с помощью семантических векторов. Эта особенность позволяет LogAnomaly повышать точность за счёт сопоставления новых шаблонов журналов с уже существующими в обучающих данных, вместо того чтобы помечать их как аномалии, как это делает DeepLog. Тем не менее, как и DeepLog, LogAnomaly обучает модель с использованием индекса события журнала (последовательные и количественные векторы) и не

способна осваивать семантический смысл шаблонов журналов, вследствие чего сильно подвержена влиянию ошибок на этапе парсинга (см. раздел 4.4). Более того, LogAnomaly не демонстрирует высокой эффективности на крупных наборах данных с многочисленными событиями журнала (см. разделы 4.1 и 4.2).

6.1.3 PLELog

Основное преимущество PLELog состоит в том, что она способна извлекать знания об исторических аномалиях посредством вероятностной оценки меток. PLELog применяет метод кластеризации (а именно HDBSCAN [41]) для вероятностной оценки меток неразмеченных журнальных последовательностей. Такой подход позволяет работать только с нормальными записями журнала. Кроме того, использование семантических векторов и GRU-сети с механизмом внимания повышает эффективность работы PLELog. Однако PLELog требует значительных временных затрат, поскольку нуждается во времени на обучение модели кластеризации (см. раздел 5.5). Более того, алгоритм плохо справляется с шумом в обучающих данных (см. раздел 5.4.1) и не демонстрирует хороших результатов в задаче раннего обнаружения.

6.1.4 LogRobust

LogRobust задействует семантические векторы шаблонов журналов в сочетании с двунаправленной LSTM-моделью с механизмом внимания. LogRobust способна эффективно справляться с шумом, вызванным ошибками парсинга, благодаря использованию двунаправленной LSTM-модели с механизмом внимания, которая обладает способностью улавливать контекстную информацию журнальных последовательностей. Однако, как характерно для моделей с обучением с учителем, LogRobust требует наличия как нормальных, так и аномальных данных на этапе обучения, что сопряжено со значительными трудозатратами на ручную разметку. Ещё одним недостатком LogRobust является то, что она может сильно подвергаться влиянию шума, вызванного ошибочно размеченными записями журнала (см. раздел 5.4).

6.1.5 CNN

Свёрточная нейронная сеть способна выявлять больше зависимостей в контексте журнала за счёт использования множества фильтров. Операция свёртки позволяет CNN улавливать не только корреляцию между шаблонами журналов, но и корреляцию внутри семантического представления (эмбединга) шаблонов журналов [15]. CNN способна достигать высокой точности на многих наборах данных благодаря подходу с обучением с учителем. Однако, подобно LogRobust, CNN требует большого объёма размеченных данных, которые на практике в большинстве случаев недоступны. Кроме того, CNN теряет точность при работе с сильно несбалансированными и зашумлёнными данными (см. разделы 5.3 и 5.4).

6.2 Направления будущих исследований

Один аспект, который никогда не теряет своей актуальности задачах машинного обучения – расширение корпуса наборов данных. Для всесторонней оценки необходимо использовать более широкий спектр наборов данных. Хороший результат на одном наборе данных не гарантирует аналогичных результатов на других в силу различия характеристик данных.

Второе направление связано с ограниченностью размеченных данных. Несмотря на то, что методы с обучением с учителем (LogRobust и CNN) демонстрируют более высокую точность, ручная разметка аномалий крайне трудоёмка. Совершенствование прогнозирующих методов или разработка эффективных классифицирующих моделей является важной задачей будущих исследований.

Третье направление связано с выявлением аномалий в реальном времени. Необходимы исследования по созданию эффективных моделей, обеспечивающих достаточно раннее обнаружение (с минимальным числом просматриваемых записей журнала) при одновременном сохранении высокой точности в режиме онлайн-детектирования.

Так же перспективным направлением может считаться выявление зависимости между событиями журнала. Как показано в разделе 5.2, группировка по

сессионным окнам позволяет лучше уловить зависимости в рамках одного пути выполнения. Необходимо изучение более широкого спектра зависимостей между событиями журнала – логических и интерактивных зависимостей между компонентами системы [45] – для охвата разнообразных аномальных паттернов поведения.

6.3 Угрозы достоверности анализа

Первой и наиболее существенной угрозой внешней достоверности является ограниченное число моделей глубокого обучения, подвергнутых экспериментальной оценке. В рамках настоящей работы анализу подверглись лишь пять репрезентативных моделей, отобранных по критерию публичной доступности исходного кода. Вместе с тем следует подчеркнуть, что отобранные модели охватывают все три выделенных типа векторного представления логов – последовательные, количественные и семантические векторы – и отражают ключевые архитектурные подходы, представленные в литературе. Это обеспечивает репрезентативность выборки в рамках обозначенного спектра методов, однако не позволяет распространять выводы на всю совокупность существующих решений. В качестве направления дальнейшей работы планируется поэтапная реализация и включение в сравнительный анализ моделей, исходный код которых в открытом доступе отсутствует.

Во-вторых, в работе имела место реимплементация. В частности, модель LogRobust была перенесена с фреймворка Keras на PyTorch с сохранением исходной архитектуры и гиперпараметров в точном соответствии с оригинальной публикацией. Необходимость данного переноса обусловлена требованием унификации экспериментальной среды: использование единого фреймворка глубокого обучения для всех оцениваемых моделей исключает влияние межфреймворковых различий в реализации численных операций на сопоставимость результатов. Для моделей LogAnomaly и CNN в качестве компонента семантического эмбединга была применена модель FastText ввиду недоступности оригинальных предобученных векторов, указанных в соответствующих публикациях. Данная замена является методологически

обоснованной, поскольку FastText относится к тому же классу статических дистрибутивных моделей, что и оригинальные компоненты, и обеспечивает семантическое представление сопоставимого качества. Корректность всех выполненных реимплементаций подтверждена воспроизведением результатов, опубликованных в оригинальных работах.

7 Методика выбора модели глубокого обучения для обнаружения аномалий при интеграции в комплексную систему безопасности

Рассмотренные модели глубокого обучения демонстрируют значения F-меры выше 0,9 на общедоступных и широко используемых тренировочных датасетах. Тем не менее проведенное сравнительное исследование показывает, что ни одна из них не обеспечивает стабильно высоких показателей при смене условий эксперимента: изменении стратегии формирования обучающей выборки, метода группировки, соотношения классов, уровня зашумлённости данных и т.д. Следовательно, отчётные показатели, воспроизводимые только в идеализированной лабораторной среде, не могут служить достаточным основанием для принятия инфраструктурных решений. Настоящая методика систематизирует критерии выбора архитектуры модели ГО для обнаружения аномалий по журналам регистрации событий применительно к комплексным системам информационной безопасности.

Методика ориентирована на реальные характеристики входных данных и операционные приоритеты конкретной организации. Методика охватывает два взаимосвязанных измерения:

- структуру и качество доступных журнальных данных,
- бизнес-требования к системе детектирования, прежде всего соотношение между скоростью реагирования и точностью классификации, а также асимметрию операционных издержек ложноположительных и ложноотрицательных срабатываний.

7.1 Аудит данных

7.1.1 Наличие и качество разметки

Принципиальное разграничение проходит между полностью размеченными данными (каждому событию или сессии присвоена метка «норма» / «аномалия»), частично размеченными данными (имеются лишь метки нормального класса – сценарий PU-обучения) и полностью неразмеченными данными.

Полностью размеченные данные открывают доступ к моделям с обучением с учителем – LogRobust и CNN. Исследование демонстрирует, что при хронологической стратегии выбора данных именно эти модели устойчиво превосходят прогнозирующие архитектуры: на наборе Thunderbird LogRobust и CNN достигают F-меры около 0,94, тогда как DeepLog и LogAnomaly – лишь 0,33. Вместе с тем необходимо принимать во внимание существенное ограничение: ручная разметка аномалий в промышленных системах крайне трудоёмка и зачастую недостижима в необходимом объёме.

Частично размеченные данные (только нормальный класс) обуславливают выбор прогнозирующих моделей (DeepLog или LogAnomaly) либо PLELog, применяющего вероятностную оценку меток на основе кластеризации. DeepLog и LogAnomaly обучаются исключительно на нормальных последовательностях, фиксируя аномалию как отклонение от предсказанного паттерна. PLELog дополнительно способен извлекать знания о структуре аномального класса, не требуя явной разметки.

Ошибочная разметка представляет отдельный операционный риск. При введении в обучающую выборку лишь 1% неверно размеченных образцов F-мера LogRobust и CNN падает на 36–37 процентных пунктов; при уровне шума 10% эти модели практически теряют работоспособность. Прогнозирующие модели, в особенности LogAnomaly, демонстрируют значительно более высокую устойчивость к ошибочной разметке в силу использования количественных векторов, способных сглаживать единичные «выбросы» в разметке. Таким образом, перед выбором архитектуры необходимо провести аудит качества разметки: оценить долю потенциально ошибочных меток, установить, верифицированы ли метки независимо, определить, покрывают ли аномальные примеры все актуальные для организации классы угроз. При наличии обоснованных сомнений в качестве разметки использовать прогнозирующие модели вне зависимости от количества доступных меток.

7.1.2 Сбалансированность классов

Дисбаланс классов является одним из наиболее критичных и систематически недооцениваемых факторов. Аномальные события в реальных журналах составляют, как правило, от долей процента до нескольких процентов (в датасете Thunderbird – 0,49%; в HDFS – 2,93%; в BGL – 7,34%). Стандартная F-мера в условиях значительного дисбаланса может оставаться высокой даже при массовой ложной классификации нормальных событий как аномальных: на наборе Spirit при случайном разделении DeepLog и LogAnomaly демонстрируют F-меру 0,929 и 0,937 соответственно, однако специфичность составляет лишь 0,39 и 0,46, то есть более половины нормальных событий ошибочно помечаются как аномальные [46].

Следует особо подчеркнуть: высокое значение F-меры при низкой специфичности свидетельствует не о качестве модели, а о ненадёжности метрики в условиях дисбаланса. Специфичность является обязательной метрикой оценки наряду с F-мерой.

Исследование показывает, что при экстремально низкой доле аномалий (0,1–1%) все рассматриваемые модели демонстрируют заметную деградацию показателей. CNN при доле аномалий 0,1% на HDFS достигает F-меры лишь 0,55 против 0,983 при 15%. Практический вывод состоит в том, что реальные производственные данные, как правило, соответствуют именно крайне несбалансированному сценарию.

Практические рекомендации при дисбалансе классов:

- добавить в оценочную систему специфичность и рассматривать модели с приемлемым балансом между полнотой и специфичностью,
- применять методы аугментации (SMOTE или аналоги) для обогащения аномального класса при обучении моделей с учителем,
- при экстремальном дисбалансе (менее 1% аномалий) отдавать предпочтение прогнозирующим моделям (DeepLog, LogAnomaly) или PLELog, поскольку они не опираются на сбалансированный набор при обучении.

7.1.3 Структура данных и возможность сессионной группировки

Метод группировки журнальных событий в последовательности существенно влияет на эффективность детектирования. Исследование демонстрирует, что при использовании фиксированного окна производительность всех моделей значительно варьируется в зависимости от его размера и остаётся нестабильной. Напротив, при сессионной группировке все рассматриваемые модели достигают значений F-меры выше 0,9 – даже те, что при фиксированном окне показывают неудовлетворительные результаты.

Сессионная группировка возможна только при наличии в журналах стабильного идентификатора сессии или потока выполнения – `block_id`, `request_id` и аналогичных полей. В датасетах Spirit и Thunderbird подобные идентификаторы отсутствуют, что исключает сессионную группировку и вынуждает прибегать к временным окнам с их нестабильностью.

На этапе аудита данных необходимо установить: содержат ли журналы организации поле, однозначно идентифицирующее логически связанную последовательность событий. При наличии такого поля – использовать сессионную группировку как приоритетный вариант. В отсутствие идентификатора – применять временное скользящее окно и эмпирически подбирать его размер по кривой полноты/специфичности на валидационной выборке.

7.1.4 Разнообразие шаблонов и чувствительность к ошибкам парсинга

Парсер журналов является первым звеном в пайплайне обнаружения аномалий и вносит самостоятельный источник шума. На наборе Spirit DeepLog демонстрирует F-меру 0,609 при парсере Drain против 0,755 при IPLoM. Методы, использующие только индекс шаблона (DeepLog, LogAnomaly), принципиально зависят от качества парсинга: неверно разделённые или избыточно объединённые шаблоны прямо транслируются в ошибки предсказания. Семантические модели (LogRobust, CNN) значительно устойчивее к ошибкам парсинга, поскольку работают с содержательным представлением текста.

Датасет Thunderbird имеет наибольшее число уникальных шаблонов среди рассматриваемых наборов, что создаёт максимальную нагрузку на парсер; именно на этом наборе DeepLog и LogAnomaly показывают наихудшие результаты (F-мера 0,33–0,37). Промышленные журналы enterprise-систем нередко имеют сопоставимую сложность шаблонного пространства.

Практическая рекомендация при богатом и нестабильном шаблонном пространстве будет заключаться в том, чтобы предпочесть семантические модели (LogRobust, CNN), устойчивые к шуму парсинга. При выборе парсера – сравнить не менее двух инструментов (например, Drain и IPLoM) по на валидационной выборке.

7.2 Согласование с бизнес-требованиями

7.2.1 Скорость реагирования против точности детектирования

Прогнозирующие модели (DeepLog, LogAnomaly) и классифицирующие (PLELog, LogRobust, CNN) фундаментально различаются по моменту генерации оповещения. DeepLog в среднем поднимает оповещение после просмотра 14,3 записей; PLELog – после 64 записей той же последовательности. На наборе BGL DeepLog генерирует оповещение после 11 записей, PLELog – после 64. Классифицирующим моделям необходимо накопить достаточно признаков аномального поведения для надёжной классификации – это принципиальный архитектурный компромисс.

Данная закономерность имеет прямые операционные последствия. Если критически важна минимизация времени реакции – например, в системах защиты АСУ ТП, где распространение инцидента измеряется секундами, – следует выбирать прогнозирующие модели, принимая неизбежное снижение специфичности. Если допустима задержка в несколько десятков событий, но требуется высокая точность и низкий уровень ложных срабатываний, предпочтение отдаётся классифицирующим моделям.

PLELog выделяется отдельно: его обучение включает этап кластеризации HDBSCAN (36,6 мин) и этап обучения GRU-сети (69,1 мин) – итого свыше 100 мин только на обучение. Такое время переобучения может быть неприемлемым

при частых обновлениях модели. PLELog может оказаться непригодным для сценариев онлайн-детектирования с требованием частого дообучения.

7.2.2 Асимметрия операционных издержек

Выбор модели принципиально зависит от того, какой тип ошибки организация считает более критичным. Ниже формализованы два граничных сценария.

Сценарий А: высокая «цена» ложноположительного срабатывания. Если каждое оповещение системы требует ручной проверки аналитика безопасности и доля ложных срабатываний высока, операторы неизбежно начинают игнорировать оповещения – феномен «усталости от оповещений». В этом случае приоритетной метрикой является точность, а также специфичность. Реальный кейс, описанный в исходном исследовании, демонстрирует: DeepLog сгенерировал 43 оповещения за период развёртывания, 15 из которых оказались ложными (специфичность снижена); LogAnomaly сгенерировал 29 оповещений без единого ложноположительного. В условиях ограниченного числа аналитиков SOC приоритет точности над полнотой является рациональным выбором.

Рекомендуемые архитектуры при приоритете точности: LogAnomaly (при частично размеченных данных), LogRobust и CNN (при полностью размеченных данных), PLELog (как компромисс).

Сценарий Б: высокая «цена» ложноотрицательного срабатывания. Если пропущенная аномалия влечёт за собой критические последствия – остановку производственного процесса, утечку конфиденциальных данных, нарушение функционирования критической инфраструктуры, – то максимизация полноты является приоритетом. Прогнозирующие модели DeepLog и LogAnomaly систематически демонстрируют высокую полноту (нередко 0,97) при хронологической стратегии выбора данных, за счёт снижения специфичности. Они по определению «чувствительнее» к аномалиям, поскольку помечают любое отклонение от предсказанного паттерна.

Рекомендуемые архитектуры при приоритете полноты: DeepLog (при минимальных требованиях к размеченным данным и необходимости раннего

обнаружения), LogAnomaly (при наличии большого числа шаблонов и требованиях к семантической устойчивости).

7.2.3 Вычислительные ограничения и режим развёртывания

Различные архитектуры предъявляют принципиально разные требования к вычислительным ресурсам. На наборе BGL DeepLog тратит 19,2 мин на обучение и 0,6 мин на тестирование; LogAnomaly – 56,7 и 1,6 мин соответственно; PLELog – свыше 105 мин только на обучение (кластеризация + GRU).

В режиме онлайн-детектирования (оценка поступающих событий без переобучения) все рассматриваемые модели применимы, однако PLELog предъявляет значительно более высокие требования к начальной подготовке модели. В режиме непрерывного дообучения – обновления по мере поступления новых данных – лёгкие прогнозирующие модели предпочтительнее ресурсоёмких классифицирующих.

7.3 Сводная матрица принятия решений

В таблице 18 приведена обобщённая матрица, отображающая характеристики данных и бизнес-приоритеты на рекомендуемые архитектуры. Для каждого сочетания условий указана предпочтительная модель и основной аргумент в её пользу.

Таблица 18 – Матрица принятия решений

Характеристика данных	Бизнес-приоритет	Рекомендуемая модель	Ключевой аргумент
Только нормальный класс, нет ошибок разметки	Раннее обнаружение	DeepLog	Минимальное число просмотренных записей до оповещения (~14); не требует аномальных примеров
Только нормальный класс, высокий уровень шума разметки	Точность > Полнота	LogAnomaly	Количественные векторы обеспечивают устойчивость к ошибочной разметке;

Продолжение таблицы 18

Только нормальный класс, богатое шаблонное пространство	Баланс точности и полноты	PLELog	Вероятностная оценка меток через HDBSCAN; семантические векторы устойчивы к ошибкам парсинга
Полная разметка, качественная	Точность> Полнота	LogRobust	Двунаправленная LSTM с механизмом внимания; устойчивость к шуму парсинга; высокие точность и специфичность
Полная разметка, качественная	Полнота> Точность или скорость неважна	CNN	Множество свёрточных фильтров для захвата локальных паттернов; высокая F-мера при полностью размеченных данных
Полная разметка, значительный шум разметки (> 5%)	Любой	LogAnomaly	Наибольшая устойчивость среди всех рассматриваемых моделей при высоком уровне ошибочной разметки
Экстремальный дисбаланс классов (< 1% аномалий)	Любой	PLELog или DeepLog	Не требуют сбалансированного набора; DeepLog, если важна скорость; PLELog, если важна специфичность
Наличие сессионных идентификаторов в журналах	Любой	Любая модель с сессионной группировкой	При сессионной группировке все модели достигают F-меры > 0,9, что снимает остроту архитектурного выбора

7.4 Практический алгоритм выбора

Ниже представлен пошаговый алгоритм, позволяющий систематически пройти от аудита данных к выбору конкретной архитектуры.

1. Установить наличие сессионных идентификаторов в журналах. Если они присутствуют, во всех последующих экспериментах использовать исключительно сессионную группировку: она нивелирует большинство межмодельных различий.

2. Оценить доступность и качество разметки. Если доступна только разметка нормального класса или качество полной разметки вызывает сомнения, следует ограничить выбор прогнозирующими моделями (DeepLog, LogAnomaly) и PLELog. Если высококачественная полная разметка доступна, следует расширить выбор до LogRobust и CNN.

3. Оценить степень дисбаланса классов. При доле аномалий менее 5% обязательно включить специфичность в набор оценочных метрик и проверить выбранные модели при аналогичном дисбалансе на валидационных данных. При экстремальном дисбалансе рассмотреть аугментацию данных для моделей с учителем.

4. Определить бизнес-приоритет: что критичнее – «усталость от оповещений» (ложноположительные срабатывания) или пропущенная аномалия (ложноотрицательные)? В первом случае приоритизировать LogAnomaly или LogRobust/CNN (при наличии разметки). Во втором случае приоритизировать DeepLog или LogAnomaly за высокую полноту.

5. Установить требование к скорости обнаружения. Если необходимо раннее обнаружение (минимум просмотренных записей до оповещения), выбрать DeepLog или LogAnomaly. Если задержка в несколько десятков событий допустима – рассмотреть PLELog, LogRobust, CNN.

6. Оценить вычислительные ресурсы и режим переобучения. При ограниченных ресурсах или необходимости частого дообучения – исключить PLELog из рассмотрения. При достаточных ресурсах и редком переобучении – PLELog, LogRobust, CNN применимы в полной мере.

7. Провести сравнительную валидацию на исторических данных организации при хронологической стратегии выбора данных. Метрики оценки: точность, полнота, специфичность, F-мера. Предпочтение хронологической

стратегии выбора данных как более близкой к реальным условиям развёртывания.

Для удобства описанный алгоритм так же представлен в виде дерева решений на рисунке 7.

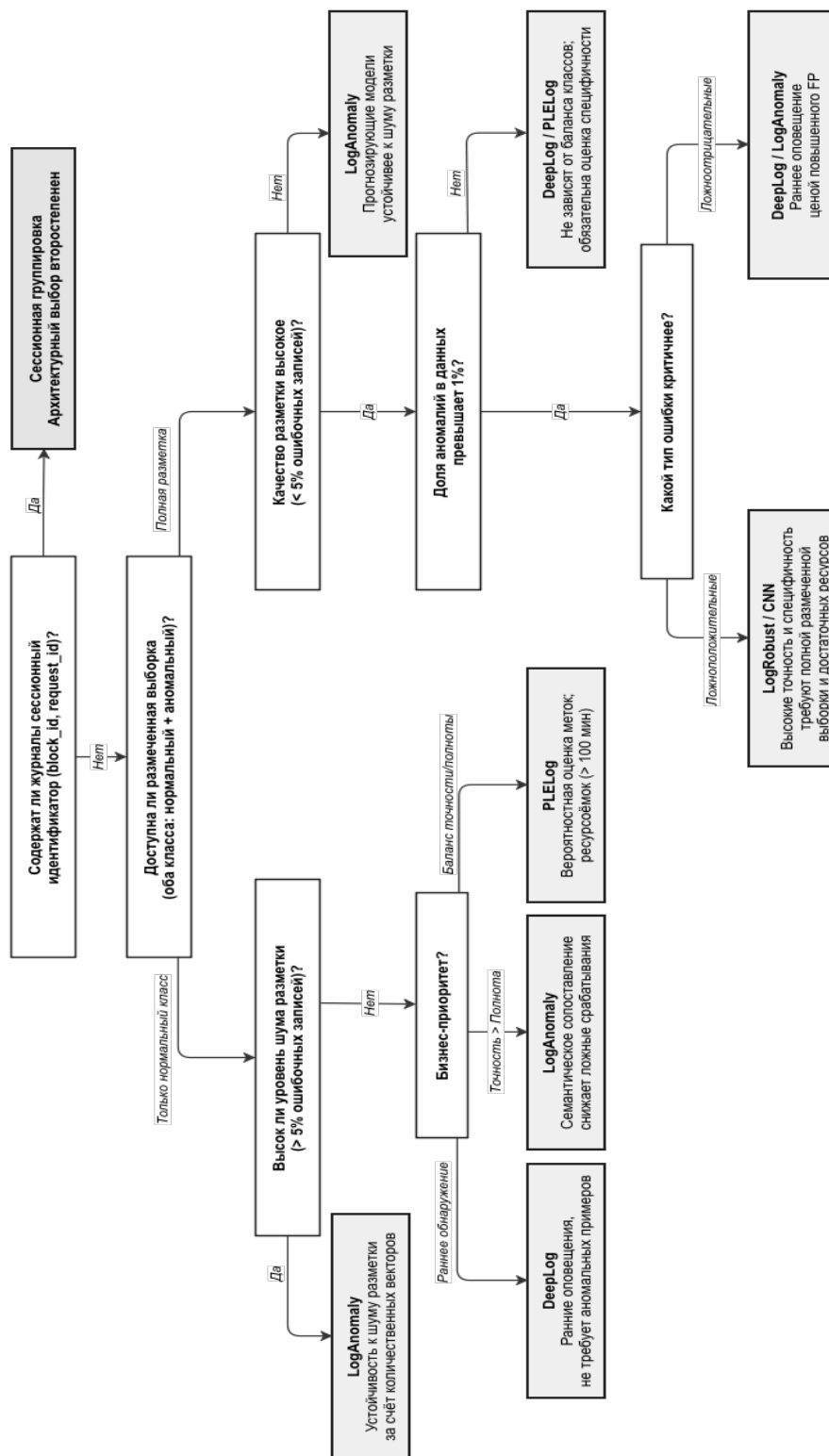


Рисунок 7 – Преобразование логов в шаблоны

ЗАКЛЮЧЕНИЕ

В настоящей работе проведён детальный анализ современных моделей глубокого обучения для обнаружения аномалий по журналам регистрации. Исследованы несколько аспектов оценки моделей и сделаны следующие выводы:

- стратегии отбора обучающих данных (случайная или хронологическая) оказывают существенное влияние на модели обнаружения аномалий с подкреплением. Случайный отбор обучающих данных может вызвать утечку данных и привести к неоправданно высоким показателям точности обнаружения,

- различные методы группировки журнальных данных оказывают значительное влияние на производительность моделей. При уменьшении длины журнальных последовательностей точность моделей, как правило, снижается,

- эффективность моделей существенно страдает при значительном дисбалансе классов. Широко применяемые метрики – точность, полнота и F-мера – недостаточно информативны для оценки модели обнаружения аномалий в условиях сильного дисбаланса классов,

- даже небольшой объём зашумлённых данных – ошибочно размеченных записей и ошибок разбора журналов – снижает качество обнаружения аномалий. По сравнению с методами обучения с подкреплением, обучаемые с учителем модели более чувствительны к ошибкам разметки в обучающей выборке. Модели, способные понимать семантику журнальных данных, позволяют снизить влияние ошибок разбора журналов,

- различные модели обладают неодинаковыми возможностями раннего обнаружения системных аномалий.

Полученные результаты свидетельствуют о том, что все рассмотренные аспекты оказывают значительное влияние на итоговые показатели, а производительность моделей зачастую не соответствует заявленным значениям.

Задача обнаружения аномалий по журналам регистрации по-прежнему не решена в полной мере.

Разработанная методика, описанная в разделе 7, формализует процесс выбора модели ГО для обнаружения аномалий как функцию двух независимо оцениваемых измерений: характеристик доступных данных и операционных приоритетов организации. Основной вывод состоит в следующем: при достаточном качестве разметки и наличии сессионных идентификаторов в журналах архитектурный выбор становится второстепенным – все рассматриваемые модели достигают приемлемых показателей. Напротив, при неполной, зашумлённой разметке, значительном дисбалансе классов или отсутствии сессионных идентификаторов выбор архитектуры приобретает определяющее значение, и приоритет следует отдавать прогнозирующим моделям как более устойчивым в реальных производственных условиях. Публикуемые значения F-меры следует рассматривать исключительно как верхнюю оценку при идеализированных условиях, а не как прогноз производительности в конкретной инфраструктуре.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Актуальные киберугрозы: IV квартал 2022 года [Электронный ресурс]. – URL: <https://www.ptsecurity.com/ru/research/analytics/cybersecurity-threatscape-2022-q4/>
2. Актуальные киберугрозы для промышленных организаций: итоги 2022 года [Электронный ресурс] – URL: <https://www.ptsecurity.com/ru/research/analytics/industrial-cybersecurity-threatscape-2022/#id4>
3. Joulin A. FastText.zip: Compressing Text Classification Models / A. Joulin, E. Grave, P. Bojanowski [et al.] // arXiv preprint arXiv:1612.03651. – 2016.
4. Du M. Spell: Streaming Parsing of System Event Logs / M. Du, F. Li // Proceedings of 2016 IEEE 16th International Conference on Data Mining (ICDM). – IEEE, 2016. – P. 859–864.
5. Du M. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning / M. Du, F. Li, G. Zheng, V. Srikumar // Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. – 2017. – P. 1285–1298.
6. He P. Drain: An Online Log Parsing Approach with Fixed Depth Tree / P. He, J. Zhu, Z. Zheng, M. R. Lyu // Proceedings of 2017 IEEE International Conference on Web Services (ICWS). – IEEE, 2017. – P. 33–40.
7. Makanju A. A. O. Clustering Event Logs Using Iterative Partitioning / A. A. O. Makanju, A. N. Zincir-Heywood, E. E. Milios // Proceedings of 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. – 2009. – P. 1255–1264.
8. Zhang H. An Investigation of the Relationships between Lines of Code and Defects / H. Zhang // Proceedings of 2009 IEEE International Conference on Software Maintenance. – IEEE, 2009. – P. 274–283.
9. Zhu J. Tools and Benchmarks for Automated Log Parsing / J. Zhu, S. He, J. Liu [et al.] // Proceedings of ICSE 2019. – IEEE, 2019. – P. 121–130.

10. A PyTorch Implementation of DeepLog // GitHub : [сайт]. – URL: <https://github.com/wuyifan18/DeepLog> (дата обращения: 05.04.2026).
11. Kabinna S. Examining the Stability of Logging Statements / S. Kabinna, C.-P. Bezemer, W. Shang [et al.] // Empirical Software Engineering. – 2018. – Vol. 23, № 1. – P. 290–333.
12. Таров Е.В. Искусственный интеллект в защите информации // Инновации. Наука. Образование. – 2021. – № 31. – С. 434–438.
13. Pennington J. GloVe: Global Vectors for Word Representation / J. Pennington, R. Socher, C. D. Manning // Proceedings of 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). – 2014. – P. 1532–1543.
14. Zhang H. Comments on «Data Mining Static Code Attributes to Learn Defect Predictors» / H. Zhang, X. Zhang // IEEE Transactions on Software Engineering. – 2007. – Vol. 33, № 9. – P. 635–637.
15. Nagappan M. Abstracting Log Lines to Log Event Types for Mining Software System Logs / M. Nagappan, M. A. Vouk // Proceedings of 7th IEEE Working Conference on Mining Software Repositories (MSR). – IEEE, 2010. – P. 114–117.
16. Chen M. Failure Diagnosis Using Decision Trees / M. Chen, A. X. Zheng, J. Lloyd [et al.] // Proceedings of International Conference on Autonomic Computing (ICAC). – IEEE, 2004. – P. 36–43.
17. Kim Y. Convolutional Neural Networks for Sentence Classification / Y. Kim // Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). – Doha, 2014.
18. Vaarandi R. LogCluster – A Data Clustering and Pattern Mining Algorithm for Event Logs / R. Vaarandi, M. Pihelgas // Proceedings of 11th International Conference on Network and Service Management (CNSM). – IEEE, 2015. – P. 1–7.
19. Deepseek [сайт] – URL: <https://chat.deepseek.com> (дата обращения: 05.03.2026). [Промт: Объясни, почему исследователи часто ошибаются в диагностике причин низких метрик (точности/полноты) на бенчмарках]

20. Guo H. LogBERT: Log Anomaly Detection via BERT / H. GuoS. Yuan, X. Wu // arXiv preprint arXiv:2103.04475. – 2021.
21. Sokolova M. Beyond Accuracy, F-score and ROC: A Family of Discriminant Measures for Performance Evaluation / M. Sokolova, N. Japkowicz, S. Szpakowicz // Proceedings of Australasian Joint Conference on Artificial Intelligence. – Springer, 2006. – P. 1015–1021.
22. Zhang B. Anomaly Detection via Mining Numerical Workflow Relations from Logs / B. Zhang, H. Zhang, P. Moscato, A. Zhang // Proceedings of 2020 International Symposium on Reliable Distributed Systems (SRDS). – IEEE, 2020. – P. 195–204.
23. Hochreiter S. Long Short-Term Memory / S. Hochreiter, J. Schmidhuber // Neural Computation. – 1997. – Vol. 9, № 8. – P. 1735–1780.
24. Xu W. Detecting Large-Scale System Problems by Mining Console Logs / W. Xu, L. Huang, A. Fox [et al.] // Proceedings of ACM SIGOPS 22nd Symposium on Operating Systems Principles. – 2009. – P. 117–132.
25. Yerushalmy J. Statistical Problems in Assessing Methods of Medical Diagnosis, with Special Reference to X-ray Techniques / J. Yerushalmy // Public Health Reports (1896-1970). – 1947. – P. 1432–1449.
26. Jiang Z. M. Abstracting Execution Logs to Execution Events for Enterprise Applications / Z. M. Jiang, A. E. Hassan, P. Flora, G. Hamann // Proceedings of 8th International Conference on Quality Software. – IEEE, 2008. – P. 181–186.
27. Li X. SwissLog: Robust and Unified Deep Learning Based Log Anomaly Detection for Diverse Faults / X. Li, P. Chen, L. Jing [et al.] // Proceedings of ISSRE 2020. – IEEE, 2020. – P. 92–103.
28. Nedelkoski S. Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs / S. Nedelkoski, J. Bogatinovski, A. Acker [et al.] // arXiv preprint arXiv:2008.09340. – 2020.
29. A Toolkit for Automated Log Parsing // GitHub : [сайт]. – URL: <https://github.com/logpai/logparser> (дата обращения: 05.04.2026).

30. Dai H. Logram: Efficient Log Parsing Using n-Gram Dictionaries / H. Dai, H. Li, C. Shao Chen [et al.] // IEEE Transactions on Software Engineering. – 2020. – Vol. 48, № 3. – P. 879–892.
31. Meng W. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs / W. Meng, Y. Liu, Y. Zhu [et al.] // Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI). – 2019. – Vol. 7. – P. 4739–4745.
32. Vaswani A. Attention Is All You Need / A. Vaswani, N. Shazeer, N. Parmar [et al.] // arXiv preprint arXiv:1706.03762. – 2017.
33. Deepseek [сайт] – URL: <https://chat.deepseek.com> (дата обращения: 05.03.2026). [Промт: Объясни различия между последовательными, количественными и семантическими векторными представлениями событий логов]
34. Lin Q. Predicting Node Failure in Cloud Service Systems / Q. Lin, K. Hsieh, Y. Dang [et al.] // Proceedings of ESEC/FSE 2018. – 2018. – P. 480–490.
35. Le V.-H. Log-based Anomaly Detection Without Log Parsing / V.-H. Le, H. Zhang // Proceedings of 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). – 2021. – P. 492–504.
36. Fenton N. E. Quantitative Analysis of Faults and Failures in a Complex Software System / N. E. Fenton, N. Ohlsson // IEEE Transactions on Software Engineering. – 2000. – Vol. 26, № 8. – P. 797–814.
37. Lou J.-G. Mining Invariants from Console Logs for System Problem Detection / J.-G. Lou, Q. Fu, S. Yang [et al.] // Proceedings of USENIX Annual Technical Conference. – 2010. – P. 1–14.
38. Симаворян С.Ж. О концепции создания интеллектуальных систем защиты информации на основе нейросетевых систем обнаружения вторжений в АСОД // Программные системы и вычислительные методы. – 2019. – №. 3. – С. 30–36.

39. Ковун В.А., Каширина И.Л. Применение методов машинного обучения в задачах обеспечения кибербезопасности // Актуальные проблемы прикладной математики, информатики и механики. – 2019. – С. 233–239.
40. Deepseek [сайт] – URL: <https://chat.deepseek.com> (дата обращения: 05.03.2026). [Промт: Объясни, что такое точность (Precision), полнота (Recall), специфичность (Specificity) и F-мера (F1-Score) в контексте обнаружения аномалий]
41. Farzad A. Unsupervised Log Message Anomaly Detection / A. Farzad, T. A. Gulliver // ICT Express. – 2020. – Vol. 6, № 3. – P. 229–237.
42. He S. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics / S. He, J. Zhu, P. He, M. R. Lyu // arXiv preprint arXiv:2008.06448. – 2020.
43. Hamooni H. LogMine: Fast Pattern Recognition for Log Analytics / H. Hamooni, B. Debnath, J. Xu [et al.] // Proceedings of the 25th ACM International Conference on Information and Knowledge Management. – 2016. – P. 1573–1582.
44. Oliner A. What Supercomputers Say: A Study of Five System Logs / A. Oliner, J. Stearley // Proceedings of DSN 2007. – IEEE, 2007. – P. 575–584.
45. McInnes L. HDBSCAN: Hierarchical Density Based Clustering / L. McInnes, J. Healy, S. Astels // Journal of Open Source Software. – 2017. – Vol. 2, № 11. – P. 205.
46. Deepseek [сайт] – URL: <https://chat.deepseek.com> (дата обращения: 05.03.2026). [Промт: Дай практические рекомендации по работе с дисбалансом классов]

Приложение А
(обязательное)

Применение методов машинного обучения для выявления аномалий

безопасности

Код программы

main.py

```
logging.basicConfig(
    format="% (asctime)s - %(levelname)s - %(name)s -
% (message)s",
    datefmt="%m/%d/%Y %H:%M:%S",
    level=logging.INFO,
)

accelerator = Accelerator()
def build_vocab(vocab_path: str,
               data_dir: str,
               train_path: str,
               embeddings: str,
               embedding_dim: int = 300,
               is_unsupervised: bool = False,
               logger: Logger = getLogger("__name__")) ->
Vocab:
    """
    Build vocab from training data
    Parameters
    -----
    vocab_path: str: Path to save vocab
    data_dir: str: Path to data directory
    train_path: str: Path to training data
    embeddings: str: Path to pretrained embeddings
    embedding_dim: int: Dimension of embeddings
    is_unsupervised: bool: Whether the model is unsupervised or
not
    logger: Logger: Logger

    Returns
    -----
    vocab: Vocab: Vocabulary
    """
    if not os.path.exists(vocab_path):
```

```

        with open(train_path, 'rb') as f:
            data = pickle.load(f)
        if is_unsupervised:
            logs = [x['EventTemplate'] for x in data if
np.max(x['Label']) == 0]
        else:
            logs = [x['EventTemplate'] for x in data]
        vocab = Vocab(logs, os.path.join(data_dir,
embeddings), embedding_dim=embedding_dim)
        logger.info(f"Vocab size: {len(vocab)}")
        logger.info(f"Save vocab in {vocab_path}")
        vocab.save_vocab(vocab_path)
    else:
        vocab = Vocab.load_vocab(vocab_path)
        logger.info(f"Vocab size: {len(vocab)}")
    return vocab

```

```

def build_model(args, vocab_size):
    """
    Build model
    Parameters
    -----
    args: argparse.Namespace: Arguments
    vocab_size: int: Size of vocabulary

```

```

    Returns
    -----

```

```

    """
    if args.model_name == "DeepLog":
        model_config = ModelConfig(
            num_layers=args.num_layers,
            hidden_size=args.hidden_size,
            vocab_size=vocab_size,

```

```

        embedding_dim=args.embedding_dim,
        dropout=args.dropout,

criterion=torch.nn.CrossEntropyLoss(ignore_index=0)
    )
    elif args.model_name == "LogAnomaly":
        model_config = ModelConfig(
            hidden_size=args.hidden_size,
            num_layers=args.num_layers,
            vocab_size=vocab_size,
            embedding_dim=args.embedding_dim,
            dropout=args.dropout,

criterion=torch.nn.CrossEntropyLoss(ignore_index=0),
            use_semantic=args.semantic
        )
    elif args.model_name == "LogBERT":
        model_config = ModelConfig(
            num_layers=args.num_layers,
            embedding_dim=args.embedding_dim,
            dropout=args.dropout,

criterion=torch.nn.CrossEntropyLoss(ignore_index=0),
            vocab_size=vocab_size,
            use_semantic=args.semantic,
            num_heads=args.num_heads,
        )
    elif args.model_name == "LogRobust":
        model_config = ModelConfig(
            embedding_dim=args.embedding_dim,
            hidden_size=args.hidden_size,
            num_layers=args.num_layers,
            is_bilstm=True,
            n_class=args.n_class,
            dropout=args.dropout,

```

```

        criterion=torch.nn.CrossEntropyLoss()
    )
elif args.model_name == "CNN":
    model_config = ModelConfig(
        embedding_dim=args.embedding_dim,
        max_seq_len=args.history_size,
        n_class=args.n_class,
        dropout=args.dropout,
        out_channels=args.hidden_size,
        criterion=torch.nn.CrossEntropyLoss()
    )
elif args.model_name == "PLELog":
    raise NotImplementedError
elif args.model_name == "NeuralLog":
    model_config = ModelConfig(
        embedding_dim=args.embedding_dim,
        num_layers=args.num_layers,
        n_class=args.n_class,
        dropout=args.dropout,
        dim_feedforward=args.dim_feedforward,
        num_heads=args.num_heads,
        criterion=torch.nn.CrossEntropyLoss()
    )
else:
    raise NotImplementedError(f"{args.model_name} is not
implemented")
model = get_model(args.model_name, model_config)
return model

def train_and_eval(args: argparse.Namespace,
                   train_path: str,
                   test_path: str,
                   vocab: Vocab,
                   model: torch.nn.Module,

```

```

        is_unsupervised=False,
        logger: Logger = getLogger("__name__")) ->
Tuple[float, float, float, float]:
    """
    Run model
    Parameters
    -----
    args: argparse.Namespace: Arguments
    train_path: str: Path to training data
    test_path: str: Path to test data
    vocab: Vocab: Vocabulary
    model: torch.nn.Module: Model
    is_unsupervised: bool: Whether the model is unsupervised or
not
    logger: Logger: Logger

    Returns
    -----
    Accuracy metrics
    """
    print("Loading train dataset\n")
    data, stat = load_features(train_path,

is_unsupervised=is_unsupervised,
                                is_train=True)
    logger.info(f"Train data statistics: {stat}")
    data = shuffle(data)
    n_valid = int(len(data) * args.valid_ratio)
    train_data, valid_data = data[:-n_valid], data[-n_valid:]

    sequentials, quantitatives, semantics, labels, idxs, _ =
sliding_window(
    train_data,
    vocab=vocab,
    window_size=args.history_size,

```

```

        is_train=True,
        semantic=args.semantic,
        quantitative=args.quantitative,
        sequential=args.sequential,
        is_unsupervised=is_unsupervised,
        logger=logger
    )

    if args.model_name == "LogBERT":
        train_dataset = MaskedDataset(sequentials=sequentials,
vocab=vocab, seq_len=32, idx=idxs)
    else:
        #logger.info(f"Train dataset: {len(sequentials)}")
        train_dataset = LogDataset(sequentials=sequentials,
quantitatives=quantitatives, semantics=semantics,

is_unsupervised=is_unsupervised, labels=labels, idxs=idxs,

remove_duplicates=args.remove_duplicates)
        logger.info(f"Train dataset: {len(train_dataset)}")

    sequentials,    quantitatives,    semantics,    labels,
sequence_idx, session_labels = sliding_window(
    valid_data,
    vocab=vocab,
    window_size=args.history_size,
    is_train=False,
    semantic=args.semantic,
    quantitative=args.quantitative,
    sequential=args.sequential,
    is_unsupervised=is_unsupervised,
    logger=logger
)

    if args.model_name == "LogBERT":

```

```

        valid_dataset = MaskedDataset(sequentials=sequentials,
vocab=vocab, seq_len=32, idx=sequence_idx)
    else:
        valid_dataset = LogDataset(sequentials=sequentials,
quantitatives=quantitatives, semantics=semantics,
                                labels=labels,
idxs=sequence_idx, is_unsupervised=is_unsupervised,

remove_duplicates=args.remove_duplicates)

    logger.info(f"Train dataset: {len(train_dataset)}")
    logger.info(f"Valid dataset: {len(valid_dataset)}")
    optimizer = get_optimizer(args, model.parameters())

    device = accelerator.device
    model = model.to(device)

    logger.info(f"Start training {args.model_name} model on
{device} device")
    pdb.set_trace()
    trainer = Trainer(
        model,
        train_dataset,
        valid_dataset=valid_dataset,
        is_train=True,
        optimizer=optimizer,
        no_epochs=args.max_epoch,
        batch_size=args.batch_size,
        scheduler_type=args.scheduler,
        warmup_rate=args.warmup_rate,
        accumulation_step=args.accumulation_step,
        logger=logger,
        accelerator=accelerator,
        num_classes=len(vocab) if is_unsupervised else
args.n_class,

```

```

    )

    if args.resume and
os.path.exists(f"{args.output_dir}/models/{args.model_name}.p
t"):

        logger.info(f>Loading model from
{args.output_dir}/models/{args.model_name}.pt...")

trainer.load_model(f"{args.output_dir}/models/{args.model_nam
e}.pt")

    args.train = True
    if args.train:
        train_loss, val_loss, val_acc, args.topk =
trainer.train(device=device,
save_dir=f"{args.output_dir}/models",
model_name=args.model_name,
topk=1 if not is_unsupervised else args.topk)
        logger.info(f"Train Loss: {train_loss:.4f} - Val Loss:
{val_loss:.4f} - Val Acc: {val_acc:.4f}")
        if args.model_name == "LogBERT":
            # print("len" ,len(valid_dataset_abnormal))
            # loss_normal , loss_abnormal =
trainer.predict_logbert(valid_dataset_normal ,
valid_dataset_abnormal ,device = device)
            # print(f"loss_normal: {loss_normal} , loss_abnormal :
{loss_abnormal}") 9122863245
            print("compute valid")
            acc, pre, rec, f1 = predict(trainer.model.to(device),
abnormal_dataset=valid_dataset_abnormal,

normal_dataset=valid_dataset_normal, device=device)
            print(f"Validation Result:: Acc: {acc:.4f}, Precision:
{pre:.4f}, Recall: {rec:.4f}, F1: {f1:.4f}")
            elif is_unsupervised:
                acc, recommend_topk =
trainer.predict_unsupervised(valid_dataset,

```

```

session_labels,
topk=args.topk,
device=device,
is_valid=True)
    logger.info(
        f"Validation Result:: Acc: {acc:.4f}, Top-
{args.topk} Recommendation: {recommend_topk}")
    else:
        acc, f1, pre, rec =
trainer.predict_supervised(valid_dataset,

session_labels,

device=device)
    logger.info(f"Validation Result:: Acc: {acc:.4f},
Precision: {pre:.4f}, Recall: {rec:.4f}, F1: {f1:.4f}")
    print("Loading test dataset\n")
    data, stat = load_features(test_path,

is_unsupervised=is_unsupervised,
                                is_train=False)
    logger.info(f"Test data statistics: {stat}")
    label_dict = {}
    counter = {}
    for (s, l) in data:
        label_dict[tuple(s)] = 1
        try:
            counter[tuple(s)] += 1
        except Exception:
            counter[tuple(s)] = 1
    data = [(list(k), v) for k, v in label_dict.items()]

    num_sessions = [counter[tuple(k)] for k, _ in data]
    sequentials, quantitatives, semantics, labels,
sequence_idx, session_labels = sliding_window(

```

```

        data,
        vocab=vocab,
        window_size=args.history_size,
        is_train=False,
        semantic=args.semantic,
        quantitative=args.quantitative,
        sequential=args.sequential,
        is_unsupervised=is_unsupervised,
        logger=logger
    )
    if args.model_name == "LogBERT":
        sequentials_normal = []
        sequentials_abnormal = []
        for i in range(len(labels)):
            if labels[i] == 0 or labels[i] == "0":
                sequentials_normal.append(sequentials[i])
            else:
                sequentials_abnormal.append(sequentials[i])
        test_dataset_normal =
MaskedDataset(sequentials=sequentials_normal,      vocab=vocab,
seq_len=32, idx=sequence_idx)
        test_dataset_abnormal =
MaskedDataset(sequentials=sequentials_abnormal,    vocab=vocab,
seq_len=32,

idx=sequence_idx)
        logger.info(f"Test dataset: {len(test_dataset_normal)
+ len(test_dataset_abnormal)}")
        print("compute valid")
        acc, pre, rec, f1 = predict(trainer.model.to(device),
abnormal_dataset=test_dataset_abnormal,

normal_dataset=test_dataset_normal, device=device)
        print(f"Train Result:: Acc: {acc:.4f}, Precision:
{pre:.4f}, Recall: {rec:.4f}, F1: {f1:.4f}")

```

```

        return 0

    test_dataset = LogDataset(sequentials=sequentials,
                              quantitatitives=quantitatitives, semantics=semantics,
                              labels=labels,
                              idxs=sequence_idxxs, is_unsupervised=is_unsupervised)
    logger.info(f"Test dataset: {len(test_dataset)}")
    if is_unsupervised:
        logger.info(f"Start predicting {args.model_name} model
on {device} device with top-{args.topk} recommendation")
        acc, f1, pre, rec =
trainer.predict_unsupervised(test_dataset,

session_labels,

topk=args.topk,

device=device,

is_valid=False,

num_sessions=num_sessions)
    else:
        acc, f1, pre, rec =
trainer.predict_supervised(test_dataset,

session_labels,

device=device)
    logger.info(f"Test Result:: Acc: {acc:.4f}, Precision:
{pre:.4f}, Recall: {rec:.4f}, F1: {f1:.4f}")
    return acc, f1, pre, rec

def run(args):
    logger = getLogger(args.model_name)

```

```

        logger.info(accelerator.state)
        logger.setLevel(logging.INFO if
accelerator.is_local_main_process else logging.ERROR)
        os.makedirs(args.output_dir, exist_ok=True)

        if args.grouping == "sliding":
            args.output_dir =
f"{args.output_dir}/{args.dataset_name}/sliding/W{args.window
_size}_S{args.step_size}_C{args.is_chronological}_train{args.
train_size}"
        else:
            args.output_dir =
f"{args.output_dir}/{args.dataset_name}/session/train{args.tr
ain_size}"
            train_path, test_path = process_dataset(logger,
data_dir=args.data_dir, output_dir=args.output_dir,

log_file=args.log_file,

dataset_name=args.dataset_name, grouping=args.grouping,

window_size=args.window_size, step_size=args.step_size,

train_size=args.train_size,
is_chronological=args.is_chronological,

session_type=args.session_level)

        # pdb.set_trace()

        os.makedirs(f"{args.output_dir}/vocabs", exist_ok=True)
        vocab_path =
f"{args.output_dir}/vocabs/{args.model_name}.pkl"
        is_unsupervised = args.model_name in ["LogAnomaly",
"DeepLog", "LogBERT"]

```

```

log_vocab = build_vocab(vocab_path,
                        args.data_dir,
                        train_path,
                        args.embeddings,
                        embedding_dim=args.embedding_dim,
                        is_unsupervised=is_unsupervised,
                        logger=logger)
model = build_model(args, vocab_size=len(log_vocab))
train_and_eval(args,
               train_path,
               test_path,
               log_vocab,
               model,
               is_unsupervised=is_unsupervised,
               logger=logger)

if __name__ == "__main__":
    parser = arg_parser()
    args = parser.parse_args()

    if args.config_file is not None and
os.path.exists(args.config_file):
        config_file = args.config_file
        with open(config_file, "r") as f:
            config = yaml.load(f, Loader=yaml.FullLoader)
            config_args = argparse.Namespace(**config)
            for k, v in config_args.__dict__.items():
                if v is not None:
                    setattr(args, k, v)
        print(f"Loaded config from {config_file}!")
    run(args)

```

vocab.py

```
def read_json(filename):
    with open(filename, 'r') as load_f:
        file_dict = json.load(load_f)
    return file_dict

class Vocab(object):
    def __init__(self, logs, emb_file="embeddings.json",
embedding_dim=100):

        self.stoi = {}
        self.itos = ['padding']
        self.pad_token = "padding"

        for line in logs:
            self.itos.extend(line)
        self.mask_index = 4
        self.itos = ['padding'] + list(set(self.itos))
        # self.pad_index = len(self.itos)
        # self.itos.append("padding")
        self.unk_index = len(self.itos)
        self.stoi = {e: i for i, e in enumerate(self.itos)}
        self.semantic_vectors = read_json(emb_file)
        self.semantic_vectors = {k: v if type(v) is list else
[0] * embedding_dim
                                for k, v in
self.semantic_vectors.items()}
        self.semantic_vectors[self.pad_token] = [-1] *
embedding_dim
        self.mapping = {}

    def __len__(self):
        return len(self.itos)
```

```

def get_event(self, real_event, use_similar=False):
    event = self.stoi.get(real_event, self.unk_index)
    if not use_similar or event != self.unk_index:
        return event

    if self.mapping.get(real_event) is not None:
        return self.mapping[real_event]

    for train_event in self.itos[:-1]:
        sim = dot(self.semantic_vectors[real_event],
self.semantic_vectors[train_event]) / (norm(
        self.semantic_vectors[real_event]) *
norm(self.semantic_vectors[train_event]))
        if sim > 0.90:
            self.mapping[real_event] =
self.stoi.get(train_event)
            return self.stoi.get(train_event)
    self.mapping[real_event] = self.unk_index
    return self.mapping[real_event]

def get_embedding(self, event):
    return self.semantic_vectors[event]

def save_vocab(self, file_path):
    with open(file_path, 'wb') as f:
        pickle.dump(self, f)
    @staticmethod
def load_vocab(file_path):
    with open(file_path, 'rb') as f:
        return pickle.load(f)

```

feature_extraction.py

```
def load_features(data_path, is_unsupervised=True, min_len=0,
is_train=True):
    """
    Load features from pickle file
    Parameters
    -----
    data_path: str: Path to pickle file
    is_unsupervised: bool: Whether the model is unsupervised or
not
    min_len: int: Minimum length of log sequence
    pad_token: str: Padding token
    is_train: bool: Whether the data is training data or not
    Returns
    -----
    logs: List[Tuple[List[str], int]]: List of log sequences
    """
    with open(data_path, 'rb') as f:
        data = pickle.load(f)
    if is_train:
        if is_unsupervised:
            logs = []
            no_abnormal = 0
            for seq in data:
                if len(seq['EventTemplate']) < min_len:
                    continue
                if not isinstance(seq['Label'], int):
                    label = max(seq['Label'])
                else:
                    label = seq['Label']
                if label == 0:
                    logs.append((seq['EventTemplate'], label))
                else:
                    no_abnormal += 1
            print("Number of abnormal sessions:", no_abnormal)
```

```

else:
    logs = []
    no_abnormal = 0
    for seq in data:
        if len(seq['EventTemplate']) < min_len:
            continue
        if not isinstance(seq['Label'], int):
            label = seq['Label']
            if max(label) > 0:
                no_abnormal += 1
        else:
            label = seq['Label']
            if label > 0:
                no_abnormal += 1
        logs.append((seq['EventTemplate'], label))
    print("Number of abnormal sessions:", no_abnormal)
else:
    logs = []
    no_abnormal = 0
    for seq in data:
        if len(seq['EventTemplate']) < min_len:
            continue
        if not isinstance(seq['Label'], int):
            label = seq['Label']
            if max(label) > 0:
                no_abnormal += 1
        else:
            label = seq['Label']
            if label > 0:
                no_abnormal += 1
        logs.append((seq['EventTemplate'], label))
    print("Number of abnormal sessions:", no_abnormal)
logs_len = [len(log[0]) for log in logs]
return logs, {"min": min(logs_len), "max": max(logs_len),
"mean": np.mean(logs_len)}

```

```

def sliding_window(data: List[Tuple[List[str], int]],
                  window_size: int = 40,
                  is_train: bool = True,
                  vocab: Optional[Any] = None,
                  is_unsupervised: bool = True,
                  sequential: bool = False,
                  quantitative: bool = False,
                  semantic: bool = False,
                  logger: Optional[Any] = None,
                  # remove_duplicates: bool = False
                  ) -> Any:
    """
    Sliding window for log sequence
    Parameters
    -----
    data: List[Tuple[List[str], int]]: List of log sequences
    window_size: int: Size of sliding window
    is_train: bool: training mode or not
    vocab: Optional[Any]: Vocabulary
    is_unsupervised: bool: Whether the model is unsupervised or
not
    sequential: bool: Whether to use sequential features
    quantitative: bool: Whether to use quantitative features
    semantic: bool: Whether to use semantic features
    logger: Optional[Any]: Logger
    # remove_duplicates: bool: Whether to remove duplicate
events

    Returns
    -----
    lists of sequential, quantitative, semantic features, and
labels
    """
    log_sequences = []
    session_labels = {}

```

```

unique_ab_events = set()

for idx, (templates, labels) in tqdm(enumerate(data),
total=len(data),
                                desc=f"Sliding window
with size {window_size}"):
    line = list(templates)
    line = line + [vocab.pad_token] * (window_size -
len(line) + is_unsupervised)
    if isinstance(labels, list):
        labels = [0] * (window_size - len(labels) +
is_unsupervised) + labels
    session_labels[idx] = labels if isinstance(labels, int)
else max(labels)
    for i in range(len(line) - window_size if
is_unsupervised else len(line) - window_size + 1):
        if is_unsupervised:
            label = vocab.get_event(line[i + window_size],

use_similar=quantitative) # use_similar only for LogAnomaly
        else:
            if not isinstance(labels, int):
                label = max(labels[i: i + window_size])
            else:
                label = labels
        seq = line[i: i + window_size]
        sequential_pattern = [vocab.get_event(event,
use_similar=quantitative) for event in seq]
        semantic_pattern = None
        if semantic:
            semantic_pattern = [vocab.get_embedding(event)
for event in seq]
        quantitative_pattern = None
        if quantitative:
            quantitative_pattern = [0] * len(vocab)

```

```

        log_counter = Counter(sequential_pattern)
        for key in log_counter:
            try:
                quantitative_pattern[key] =
log_counter[key]
            except Exception as _:
                pass # ignore unseen events or padding
key

        sequence = {'sequential': sequential_pattern}
        if quantitative:
            sequence['quantitative'] =
quantitative_pattern
        if semantic:
            sequence['semantic'] = semantic_pattern
            sequence['label'] = label
            sequence['idx'] = idx
            log_sequences.append(sequence)

    if is_train and not is_unsupervised:
        normal_dict = {hash(tuple(seq['sequential'])): 0 for
seq in log_sequences if seq['label'] == 0}
        for i, seq in enumerate(log_sequences):
            try:
                log_sequences[i]['label'] =
normal_dict[hash(tuple(seq['sequential']))]
            except KeyError:
                pass

    sequentials, quantitatives, semantics = None, None, None
    if sequential:
        sequentials = [seq['sequential'] for seq in
log_sequences]
    if quantitative:

```

```

        quantitatatives = [seq['quantitative'] for seq in
log_sequences]
    if semantic:
        semantics = [seq['semantic'] for seq in log_sequences]
        labels = [seq['label'] for seq in log_sequences]
        sequence_idxxs = [seq['idx'] for seq in log_sequences]
        logger.info(f"Number of sequences: {len(labels)}")
        if not is_unsupervised:
            logger.info(f"Number of normal sequence: {len(labels)
- sum(labels)}")
            logger.info(f"Number of abnormal sequence:
{sum(labels)}")

        logger.warning(f"Number of unique abnormal events:
{len(unique_ab_events)}")
        logger.info(f"Number of abnormal sessions:
{sum(session_labels.values())}/{len(session_labels)}")

    return sequentials, quantitatatives, semantics, labels,
sequence_idxxs, session_labels

```

dataset.py

```
class BaseDataset(Dataset):
    def __init__(self,
                  sequentials: Optional[List[List[int]]] =
None,
                  quantitatives: Optional[List[List[float]]] =
None,
                  semantics: Optional[List[List[float]]] =
None,
                  is_unsupervised: bool = True,
                  labels: Optional[List[int]] = None,
                  idxs: Optional[List[int]] = None,
                  remove_duplicates: bool = False):
    """ Base Dataset class for log data
    Parameters
    -----
    sequentials: List of log sequences
    quantitatives: List of quantitative features
    semantics: List of semantic features
    is_unsupervised: unsupervised or supervised
    labels: List of labels
    idxs: List of indexes
    remove_duplicates: remove duplicates from data
    """
    self.sequentials = sequentials
    self.quantitatives = quantitatives
    self.semantics = semantics
    self.is_unsupervised = is_unsupervised
    self.labels = labels
    self.idxs = idxs
    self.weights = []
    if remove_duplicates:
        if self.sequentials is None:
            raise ValueError('Provide sequentials to
remove duplicates')
```

```

        self.remove_duplicates()

def remove_duplicates(self):
    self.remove_duplicates_from_data(self.sequential)

def remove_duplicates_from_data(self, data):
    """
    remove duplicates from data
    :param data:
    :return:
    """
    unique_data = {}
    occurrences = defaultdict(int)
    unique_labels = defaultdict(int)
    for idx, (seq, label) in enumerate(zip(data,
self.labels)):
        if self.is_unsupervised:
            sample = tuple(seq + [label])
        else:
            sample = tuple(seq)
        if sample not in unique_data:
            unique_data[sample] = idx
            occurrences[sample] += 1
            unique_labels[sample] = label
        else:
            occurrences[sample] += 1
            if not self.is_unsupervised:
                unique_labels[sample] =
min(unique_labels[sample], label)
            if not self.is_unsupervised:
                self.labels = [unique_labels[tuple(seq)] for seq in
data]

    filtered_idx = sorted(unique_data.values())
    n_data = len(self.sequential)
    for idx in filtered_idx:

```

```

        if self.is_unsupervised:
            sample = tuple(data[idx] + [self.labels[idx]])
        else:
            sample = tuple(data[idx])
        self.weights.append(occurrences[sample] / n_data)
        self.sequentials = [data[idx] for idx in filtered_idxxs]
        if self.quantitatives is not None:
            self.quantitatives = [self.quantitatives[idx] for
idx in filtered_idxxs]
        if self.semantics is not None:
            self.semantics = [self.semantics[idx] for idx in
filtered_idxxs]
        self.labels = [self.labels[idx] for idx in
filtered_idxxs]
        if self.idxxs is not None:
            self.idxxs = [self.idxxs[idx] for idx in
filtered_idxxs]

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

    def __getitem__(self, idx):
        raise NotImplementedError

    def collate_fn(self, batch):
        return batch

class LogDataset(BaseDataset):
    def __init__(self, sequentials=None, quantitatives=None,
semantics=None, is_unsupervised=False, labels=None,
idxxs=None, remove_duplicates=False):
        super(LogDataset,
self).__init__(sequentials=sequentials,
quantitatives=quantitatives, semantics=semantics,

```

```

is_unsupervised=is_unsupervised, labels=labels, idxs=idxs,

remove_duplicates=remove_duplicates)
    if self.sequential is None and self.quantitatives is
None and self.semantics is None:
        raise ValueError('Provide at least one feature
type')

    def __getitem__(self, idx):
        try:
            item = {'label': self.labels[idx], 'idx':
self.idxs[idx]}
        except:
            pdb.set_trace()
            if self.sequential is not None:
                item['sequential'] =
torch.from_numpy(np.array(self.sequential[idx]))
            if self.quantitatives is not None:
                item['quantitative'] =
torch.from_numpy(np.array(self.quantitatives[idx], )[:,
np.newaxis]).float())
            if self.semantics is not None:
                item['semantic'] =
torch.from_numpy(np.array(self.semantics[idx])).float()

        return item

class MaskedDataset(BaseDataset):
    def __init__(self, sequential, vocab, seq_len,
encoding="utf-8", on_memory=True,
predict_mode=False, mask_ratio=0.75,
idx=None):
        """

```

```

        :param sequentials: log sessions/line
        :param vocab: log events collection including pad, unk
    ...

    :param seq_len: max sequence length
    :param encoding:
    :param on_memory:
    :param predict_mode: if predict
    """
    super(MaskedDataset,
self).__init__(sequentials=sequentials, labels=None, idxs=idx)
    self.vocab = vocab
    self.seq_len = seq_len
    self.idxs = idx
    self.on_memory = on_memory
    self.encoding = encoding
    self.predict_mode = predict_mode
    self.mask_ratio = mask_ratio
    self.pad_index = 0
    self.unk_index = 1
    self.eos_index = 2
    self.sos_index = 3
    self.mask_index = 4

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

    def __getitem__(self, idx):
        seq = self.sequentials[idx]

        masked, label = self.random_item(list(seq))

        # [CLS] tag = SOS tag, [SEP] tag = EOS tag
        seq = [self.sos_index] + masked
        label = [self.pad_index] + label

```

```

        return seq, idx, label

def random_item(self, tokens):
    output_label = []
    for i, token in enumerate(tokens):
        prob = random.random()
        # replace 65% of tokens in a sequence to a masked
token
        if prob < self.mask_ratio:
            # raise AttributeError("no mask in
visualization")

            if self.predict_mode:
                tokens[i] = self.mask_index
                output_label.append(token)
                continue
            prob /= self.mask_ratio

            # 80% randomly change token to mask token
            if prob < 0.8:
                tokens[i] = self.mask_index

            # 10% randomly change token to random token
            elif prob < 0.9:
                tokens[i] =
random.randrange(len(self.vocab))

            # 10% randomly change token to current token
            else:
                tokens[i] = token

        output_label.append(token)
    else:

```

```

            tokens[i] = self.vocab.stoi.get(token,
self.vocab.unk_index)
            output_label.append(0)

    return tokens, output_label

    def collate_fn(self, batch, percentile=100,
dynamical_pad=True):
        lens = [len(seq[0]) for seq in batch]

        # find the max len in each batch
        if dynamical_pad:
            # dynamical padding
            seq_len = int(np.percentile(lens, percentile))
            if self.seq_len is not None:
                seq_len = min(seq_len, self.seq_len)
        else:
            # fixed length padding
            seq_len = self.seq_len

        output = {"sequential": [], "label": [], "idx": []}
        for seq in batch:
            bert_input = seq[0][:seq_len]
            bert_idx = seq[1]
            bert_label = seq[2][:seq_len]

            padding = [self.vocab.pad_index for _ in
range(seq_len - len(bert_input))]
            bert_input.extend(padding),
            bert_label.extend(padding)

            output["sequential"].append(bert_input)
            output["label"].append(bert_label)
            output["idx"].append(bert_idx)

```

```

        output["sequential"] =
torch.tensor(np.array(output["sequential"]), dtype=torch.long)
        output["label"] =
torch.tensor(np.array(output["label"]), dtype=torch.long)
        output["idx"] = torch.tensor(np.array(output["idx"]),
dtype=torch.long)
    return output

```

Приложение Б
(обязательное)

Применение методов машинного обучения для выявления аномалий
безопасности
Презентация

Применение методов машинного обучения для выявления аномалий безопасности

Студент гр. 248ВВим1
Гречишкина М. С.

Научный руководитель
Афонин А. Ю.

1 / 7

Рисунок 1– Титульный слайд

■ АКТУАЛЬНОСТЬ И ЦЕЛЬ

Зачем выявлять аномалии в журналах



Логи — основной источник о состоянии систем

Серверы, кластеры и суперкомпьютеры непрерывно генерируют журналы. Даже незначительный сбой ведёт к финансовым потерям и срыву обслуживания, поэтому обнаружение аномалий критично для надёжности.



Существующие методы переоценены

Модели ГО рапортуют F-меру выше 0,9 на эталонных датасетах. Но эти результаты воспроизводимы лишь в идеализированной лабораторной среде и не служат основанием для инфраструктурных решений.



ЦЕЛЬ ИССЛЕДОВАНИЯ

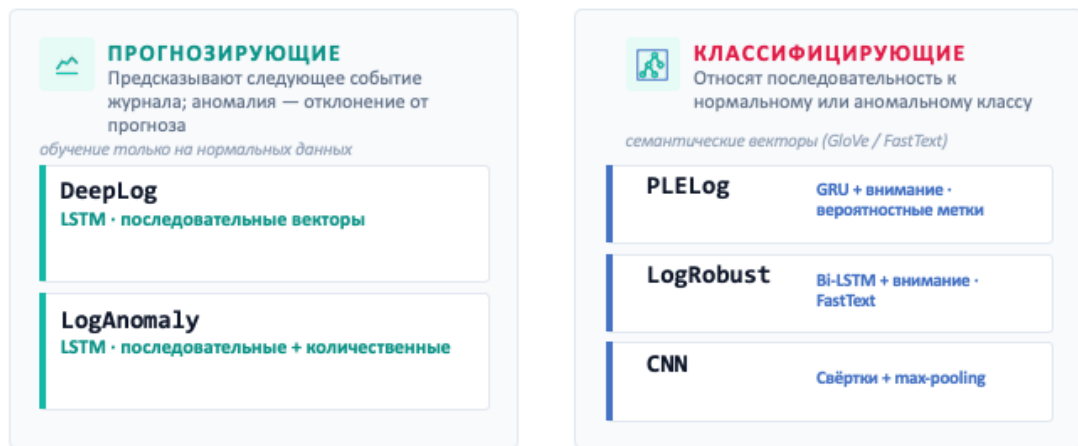
Комплексная оценка моделей ГО и разработка методики обоснованного выбора

- ✓ Унифицированная оценка 5 моделей на 4 датасетах в равных условиях
- ✓ Методика для выбора модели под реальные данные и приоритеты

2 / 7

Рисунок 2 – Второй слайд

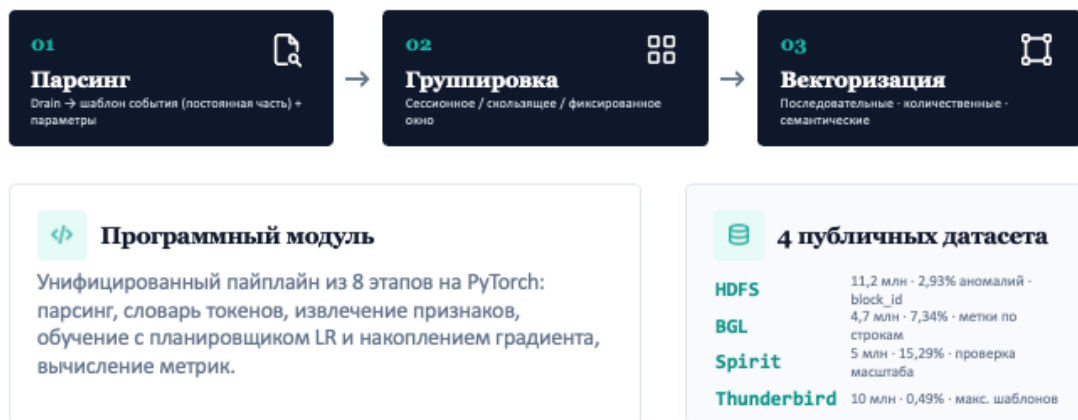
Пять исследуемых моделей ГО



3 / 7

Рисунок 3 – Третий слайд

От сырых логов к признакам



4 / 7

Рисунок 4 – Четвертый слайд

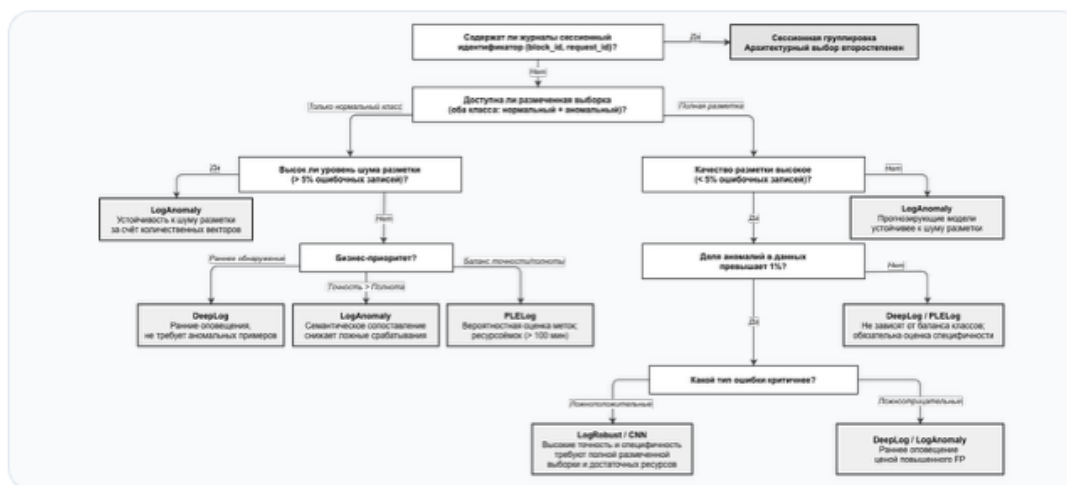
Влияние ключевых факторов на F-меру



5 / 7

Рисунок 5 – Пятый слайд

Дерево решений: данные × приоритеты



6 / 7

Рисунок 6 – Шестой слайд

Ключевые находки и применимость

Главный вывод. При качественной разметке и наличии сессионных id архитектура второстепенна — все модели приемлемы. При неполной/зашумлённой разметке, дисбалансе или отсутствии id выбор решает: приоритет — прогнозирующим моделям. Публикуемая F-мера — лишь верхняя оценка для идеальных условий.



Находки

- Стратегия отбора, группировка и дисбаланс кардинально меняют метрики
- Реальная F-мера часто не дотягивает до заявленной



Ограничения

- Оценено 5 моделей (по доступности кода)
- Реимплементация LogRobust на PyTorch; FastText вместо части эмбедингов



Будущие работы

- Больше датасетов и моделей
- Раннее обнаружение в реальном времени при высокой точности



Применение

- Готовый алгоритм выбора модели для SOC и систем ИБ
- Учёт цены FP/FN

Рисунок 7 – Седьмой слайд