УДК 004.85

МОДУЛЬ КЛАССИФИКАЦИИ ОШИБОК ВРЕМЕНИ ВЫПОЛНЕНИЯ С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ NLP

Ворона К.М., Новиков И.А., Тихонов Н.А., студенты
Чухраев И.В., к.т.н., доцент, заведующий кафедрой ИУК2
Калужский филиал МГТУ им Н.Э. Баумана,
Россия, г. Калуга

Ошибки времени выполнения (runtime exceptions) являются неотъемлемой частью жизненного цикла разработки программного обеспечения. Традиционные средства анализа (лог-мониторинг, отладчики – debuggers) позволяют выявить факт ошибки, но не дают достаточной информации для быстрой диагностики её причины. Как правило, разработчик вынужден вручную сопоставлять сообщение об ошибке, стектрейс (последовательность вызовов функций), логи и состояние приложения – процесс, требующий значительных временных и интеллектуальных затрат. Особенно остро эта проблема стоит в распределённых системах, где ошибки могут возникать в результате сложных взаимодействий между микросервисами. Для автоматизации и ускорения диагностики всё чаще применяются методы искусственного интеллекта, в частности обработка естественного языка (NLP) и классификационное машинное обучение (ML) [1].

Целью исследования является разработка программного модуля, способного автоматически классифицировать ошибки времени выполнения в Python-приложениях по семантическим категориям («работа с базой данных», «валидация входных данных», «сетевые операции», «работа с внешними API» и др.) и предлагать ранжированный список наиболее вероятных причин их возникновения. В отличие от простых регулярных выражений или поиска по типу исключения (ValueError, KeyError, ConnectionError), модуль анализирует не только имя ошибки, но и полный контекст её возникновения – стектрейс, текст логов, имена методов и переменных – и использует NLP для извлечения семантической информации из этих текстовых данных.

Модуль реализован как Python-библиотека и состоит из трёх основных компонентов: сборщика контекста, NLP-преобразователя и классификатора.

Язык программирования Python обладает развитой системой модулей, как стандартных, так и специфических, применяемых в том случае, когда необходимо решать более широкий круг задач, требующих больших ресурсов или же сложных вычислений [6]. Использование специализированных библиотек, таких как gensim для векторизации и XGBoost для классификации, соответствует успешной практике построения модульных приложений на Python.

Сборщик контекста перехватывает исключение (например, через try/except или middleware в Flask/Django) и формирует отчёт, содержащий:

- стектрейс, полученный с помощью модуля traceback;

- последние N записей лога (через интеграцию с logging);
- имена и значения ключевых переменных (через inspect и locals()).

На этапе NLP-преобразования все текстовые компоненты подвергаются предобработке: приведение к нижнему регистру, разбиение составных имён (camelCase, snake_case), удаление служебных слов и идентификаторов. Далее данные векторизуются с помощью трёх основных подходов [2]:

- Bag of Words (BoW – «мешок слов») и TF-IDF (Term Frequency – Inverse Document Frequency – «частота термина – обратная частота документа») – простые и интерпретируемые методы, особенно эффективны при шаблонных логах [2, с. 4–7]:

- Эмбеддинги слов (word embeddings) – обученные на корпусе Python-кода (1.2 млн строк) с помощью Word2Vec или FastText из библиотеки gensim. Эти методы захватывают семантические связи: например, слова query («запрос»), fetch («извлечь»), select, load_data оказываются близки в векторном пространстве, что позволяет модели обобщать паттерны вызовов [3, с. 8-16];

- Усреднённые эмбеддинги последовательностей – для стектрейса используется среднее векторов всех лексем в каждой строке. Это компромисс между простотой BoW и выразительностью BERT, при этом избегается необходимость обучения сложных трансформеров [4, с. 2-3].

Полученные векторы (BoW/TF-IDF/эмбеддинги) конкатенируются и подаются на вход классификатора. В качестве основной модели используется Gradient Boosting (реализация XGBoost) – алгоритм, демонстрирующий высокую точность при умерённой вычислительной сложности и хорошую устойчивость к несбалансированным данным [1, с. 4]. Как показано в сравнительном анализе алгоритмов машинного обучения. XGBoost является оптимизированной версией градиентного бустинга с регуляризацией, повышающей устойчивость и ускоряющей вычисления [7].

В качестве альтернативы реализована LSTM-сеть (Long Short-Term Memory – «долгая краткосрочная память») на базе TensorFlow/Keras, учитывающая порядок вызовов в стектрейсе [4, с. 1].

Обучение модели проводилось на размеченном наборе из 8 500 исключений, собранных из 12 Python-проектов (Flask/Django, Telegram-боты на aiogram, утилиты обработки данных). Каждый пример содержал:

- стектрейс и логи;
- ручную метку категории (6 основных);
- подтверждённую причину [1, с. 3].

Рассмотрим конкретный пример работы модуля. Пусть возникло исключение “sqlite3.OperationalError: no such table: users”. Традиционный поиск по типу ошибки (OperationalError) не позволяет различить случаи «таблица не создана», «файл БД повреждён», «нет прав». Модуль же анализирует:

- в стектрейсе: migrate.apply() → db.create_tables();
- в логах: "Applying migration v2.1...", "Table 'users' not found";
- в переменных: migration_script = "ALTER TABLE ...".

На основе этого он классифицирует ошибку в категорию «работа с базой данных» и предлагает гипотезу: «некорректная миграция: попытка ALTER TABLE для несуществующей таблицы» (вероятность 89%). Это позволяет разработчику сразу проверить скрипты миграций.

Эксперименты показали, что лучшая конфигурация – **TF-IDF + XGBoost** – достигает точности (accuracy) **87%** при F1-мере **0.84**. Использование F1-меры в качестве ключевой метрики обусловлено необходимостью балансировки между точностью (precision) и полнотой (recall), что особенно важно при работе с несбалансированными данными, где различные категории ошибок встречаются с разной частотой [1, с. 8]. LSTM показала F1 = 0.86, но требует в 3–5 раз больше времени на обучение и менее интерпретируема [1, с. 7-9]. Подход «поиск по типу исключения» дал accuracy 61% (F1 = 0.55). Особенно высокая эффективность наблюдается в категориях, где один тип исключения имеет разные причины: для KeyError модуль различает «отсутствие ключа в конфиге», «некорректный JSON», «ошибка сериализации» с точностью 82%, тогда как традиционные методы – 48% [1, с. 11].

Архитектура модуля предусматривает интеграцию в CI/CD-конвейер: при падении теста в GitLab CI можно вызвать `error_classifier.analyze(traceback, logs)` – результат (категория + гипотезы) автоматически добавляется в описание задачи в YouTrack. Также возможна интеграция с IDE: при возникновении исключения в режиме отладки в VS Code расширение может отобразить панель с классификацией и рекомендациями.

В ходе реализации были выявлены следующие ограничения:

- Зависимость от структуры логов. При использовании неструктурированного логгирования (print("error")) эффективность падает до 68%. Решение – рекомендация использовать logging с шаблонами (logging.error("DB connection failed: %s", err)).

- Разреженность контекста. В 22% случаев стектрейс содержит менее 3 уникальных функций – этого недостаточно для уверенной классификации [1, с. 13]. Для таких случаев модуль возвращает «низкую уверенность».

- Смещение на домен. Модель, обученная на веб-проектах, хуже работает на CLI-утилитах (F1 падает до 0.71). Требуется fine-tuning на 200–300 примерах нового проекта [1, с. 14].

На Рис. 1 – 3 представлены результаты работы модуля классификации ошибок на различных тестовых сценариях.

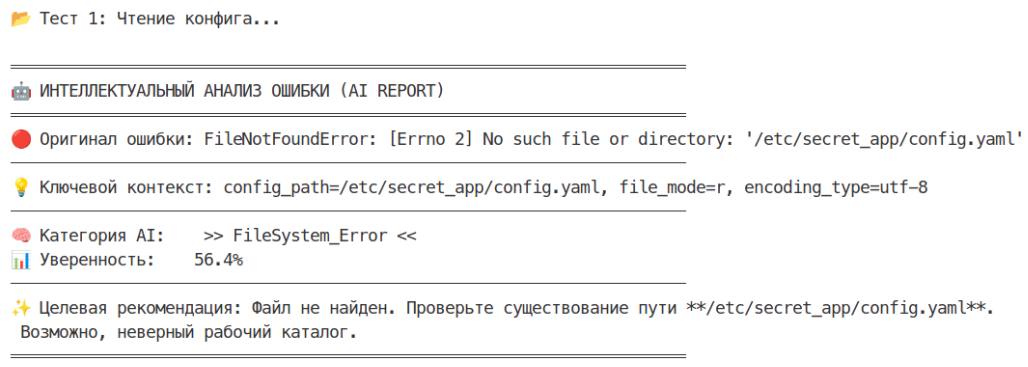

Рисунок 1 – Демонстрация результата анализа тестового сценария №1



Рисунок 2 – Демонстрация результата анализа тестового сценария №2

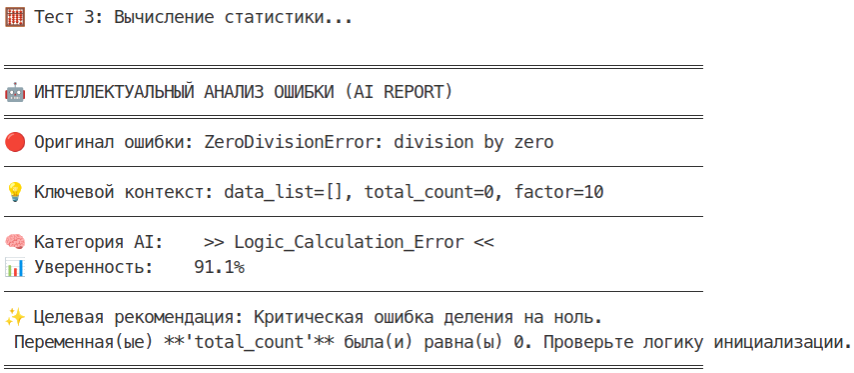

Рисунок 3 – Демонстрация результата анализа тестового сценария №2

Анализ результатов демонстрирует способность модуля точно категоризировать ошибки по их семантике и генерировать конкретные рекомендации на основе анализа контекста выполнения. В частности:

– Для FileNotFoundError (Рис. 1) модуль идентифицировал проблему доступа к файлу конфигурации и точно указал отсутствующий путь;

– При анализе PermissionError (Рис. 2) был определён контекст авторизации, что позволило отнести ошибку к категории безопасности;

– В случае ZeroDivisionError (Рис. 3) модуль с высокой уверенностью (91.1\%) классифицировал ошибку вычислений и указал на необходимость проверки инициализации переменных.

Проведённая работа подтвердила теоретическую и практическую возможность применения NLP и ML для интеллектуальной диагностики ошибок времени выполнения. Однако на практике реализация столкнулась с характерными для прикладных ML-решений проблемами: зависимостью от качества входных данных, необходимостью адаптации под домен – всё это аналогично проблемам разрежённости наград в RL-фаззинге. Преодоление этих ограничений требует дальнейших исследований – например, применения semi-supervised обучения [1,с.15] и интеграции с LLM (большими языковыми моделями) для генерации пояснений на естественном языке [4, с. 5].

Список литературы:

1. Bhandari P., Rodríguez-Pérez G. BuggIn: Automatic Intrinsic Bugs Classification Model using NLP and ML // Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE'23). – San Francisco, CA, USA, 2023. – P. 1–10. – DOI: 10.1145/3624707.3627856

2. Hussein M.J. Word Embeddings in NLP // International Journal of Advanced Computer Science and Applications. – 2025. – Vol. 16, No. 9. – P. 1–17.

3. Kulkarni A., Klopp G. Advanced Transformer Models for Next-Generation NLP and Data Science // Proceedings of the 2025 International Conference on Quantum Photonics, Artificial Intelligence, and Networking (QPAIN). – IEEE, 2025. – P. 1–6. – DOI: 10.1109/QPAIN62230.2025.10893271

4. Li Z., Zhao Y., Wang X., Zhang L. BASFuzz: Towards Robustness Evaluation of LLM-based NLP Software via Automated Fuzz Testing // arXiv preprint arXiv: 2412.08905. – 2024. – P. 1–15. – DOI: 10.48550/arXiv.2412.08905

5. Ali S., Reddy B.R., Chirra B.R. The Role of AI in Social Engineering Attack Prevention: NLP-Based Solutions for Phishing and Scams // International Journal of Advanced Engineering Technologies and Innovations. – 2024. – Vol. 1, No. 3. – P. 1–18.

6. Козеева О.О., Чухраев И.В. Разработка средствами языка Python модуля аутентификации автоматизированной информационной системы // Электронный журнал: наука, техника и образование. 2017. № 2 (12). С. 173-180.

7. Широкова Е.В., Джанаев К.С., Тихонов Н.А., Павлов И.В. Оценка эффективности применения Random Forest, Gradient Boosting и XGBoost для бинарной классификации изображений // Системный администратор. 2025. № 4 (269). С. 86-89.