УДК 519.252+519.254+004.75

МЕТОДЫ АВТОМАТИЗИРОВАННОГО ИЗВЛЕЧЕНИЯ ПАРАМЕТРОВ И ОПИСАНИЙ ПРОГРАММ ДЛЯ ИНТЕГРАЦИИ ИХ НА ВЫЧИСЛИТЕЛЬНЫЕ КОМПЛЕКСЫ

Т. В. Санников[1] [0009-0004-1144-8836], А. Н. Сальников[2] [0000-0001-8669-9905]

[1, 2]*Московский государственный университет им. М. В. Ломоносова, г. Москва, Россия*

[2]*Федеральный исследовательский центр «Информатика и управление» РАН, г. Москва, Россия*

[1]timohaj1@yandex.ru, [2]salnikov@cs.msu.ru

Аннотация

Рассмотрена проблема координации разнородных программных средств в гетерогенных средах распределенного запуска приложений. Ручное конфигурирование параметров запуска для вновь устанавливаемых программ на вычислительный кластер (таких как ключи командной строки, значения переменных окружения и настройки конфигурационных файлов) создает серьезные трудности для исследователей предметных областей из-за больших объемов служебной информации и необходимости сохранения и агрегации информации в некотором фиксированном формате. Предложен метод автоматизированного извлечения параметров запуска, базирующийся на гибридной архитектуре обучения нейронной сети, сочетающей генерацию обучающей выборки большими языковыми моделями и последующее дообучение компактного трансформерного энкодера. Реализация подхода исключает зависимость от дорогостоящих графических ускорителей за счет применения методики низкоранговой адаптации (Low-Rank Adaptation) для моделей размером до 1 млрд параметров, что обеспечивает возможность выполнения модели (инференса) на обычных центральных процессорах управляющих узлов. Для формализации качества извлечения разработана двухкомпонентная метрика, агрегирующая структурную корректность выходной JSON-схемы (наличие в полученных дан-

ных обязательных полей, типов параметров программы) и семантическую точность значений параметров (соответствие описания в документации). Экспериментальная оценка метода ориентирована на корпус документации программных пакетов (man-страницы, README). Результаты проектирования подтверждают возможность аппроксимации процесса анализа документации компактной моделью, что способствует автоматизации жизненного цикла развертывания программного обеспечения и снижению ошибок управления потоками задач в распределенных вычислительных комплексах.

Ключевые слова: *низкоранговая адаптация, извлечение данных, анализ программного кода, автоматизация запуска, обработка естественного языка, научная рабочая среда, высокопроизводительные вычисления.*

ВВЕДЕНИЕ

Распространение высокопроизводительных вычислительных ресурсов в современной научной среде претерпевает существенные изменения. Если ранее доступ к мощным вычислительным комплексам был привилегией узкого круга специализированных организаций, то в настоящее время наблюдается тенденция к демократизации доступа. Исследовательские группы все чаще получают возможность использовать несколько вычислительных комплексов одновременно для решения своих задач, что позволяет из этих комплексов составлять распределенную систему запуска приложений [1].

Существующая практика настройки программного обеспечения в подобных системах зачастую опирается на ручное вмешательство специалиста. Пользователь вынужден самостоятельно анализировать сопроводительную документацию, изучать исходный текст программ и вручную задавать параметры запуска. Такой подход имеет ряд недостатков. Во-первых, он требует от исследователя глубоких знаний в области системного программирования и архитектуры вычислительных комплексов, что не всегда соответствует профилю специалиста предметной области. Биолог, физик или химик может быть экспертом в своей науке, но не обладать достаточной квалификацией для тонкой настройки программного окружения. Во-вторых, человеческий фактор неизбежно ведет к ошибкам и неполному заполнению конфигурационных данных. Разнообразие

способов задания параметров в различном программном обеспечении, неполнота и неоднозначность документации, а также различия в форматах похожих параметров усугубляют ситуацию [2].

Цель настоящей работы заключается в облегчении пользователю процесса интеграции нового программного обеспечения в вычислительный кластер за счет разработки некоторого программного инструмента. Этот инструмент, просмотрев» исходные коды программного обеспечения и документацию, автоматически построит в некотором формате строгое формальное описание параметров программ для автоматизации дальнейшего запуска данных программ внешними средствами запуска на кластере. Программный инструмент также составит «объяснение» назначения каждого параметра в строгом формальном виде для автоматизации интеграции с внешними программами и сайтами.

Создание такого программного инструмента в прошлом было затруднено из-за большого разнообразия форм описания параметров в исходных кодах и документации. По сути применялись методы статического анализа кода и некоторые приемы в духе запустить много разнообразных grep (команды поиска и фильтрации по шаблону). Однако сейчас эта сложность частично преодолена за счет развития графических процессоров и больших нейросетевых моделей, натренированных на гигантских объемах программного кода, что делает возможным создание инструмента по автоматическому «узнаванию» параметров и описаний в коде.

1. СИСТЕМА РАСПРЕДЕЛЕННОГО ЗАПУСКА ПРИЛОЖЕНИЙ

Система распределенного запуска приложений [3] представляет собой программный комплекс, предназначенный для координации выполнения вычислительных задач на нескольких вычислительных кластерах. Архитектура системы построена по клиент-серверному принципу, где центральный узел принимает запросы от исследовательских групп и осуществляет диспетчеризацию задач по доступным вычислительным ресурсам. В основе системы лежит сервер распределения, который выполняет функции координационного центра. Сервер получает от пользователей описания потоков задач с зависимостями, анализирует доступные ресурсы кластеров и формирует оптимальное расписание выполнения задач. Ключевой особенностью архитектуры является отделе-

ние логики предметной области от инфраструктуры исполнения, что позволяет адаптировать систему к различным вычислительным окружениям без модификации исходного кода научных программ.

Принципиальным компонентом системы является механизм управления зависимостями между задачами. Зависимости определяются через файлы данных, которые передаются от выполненных задач к зависимым от них (рис. 1).

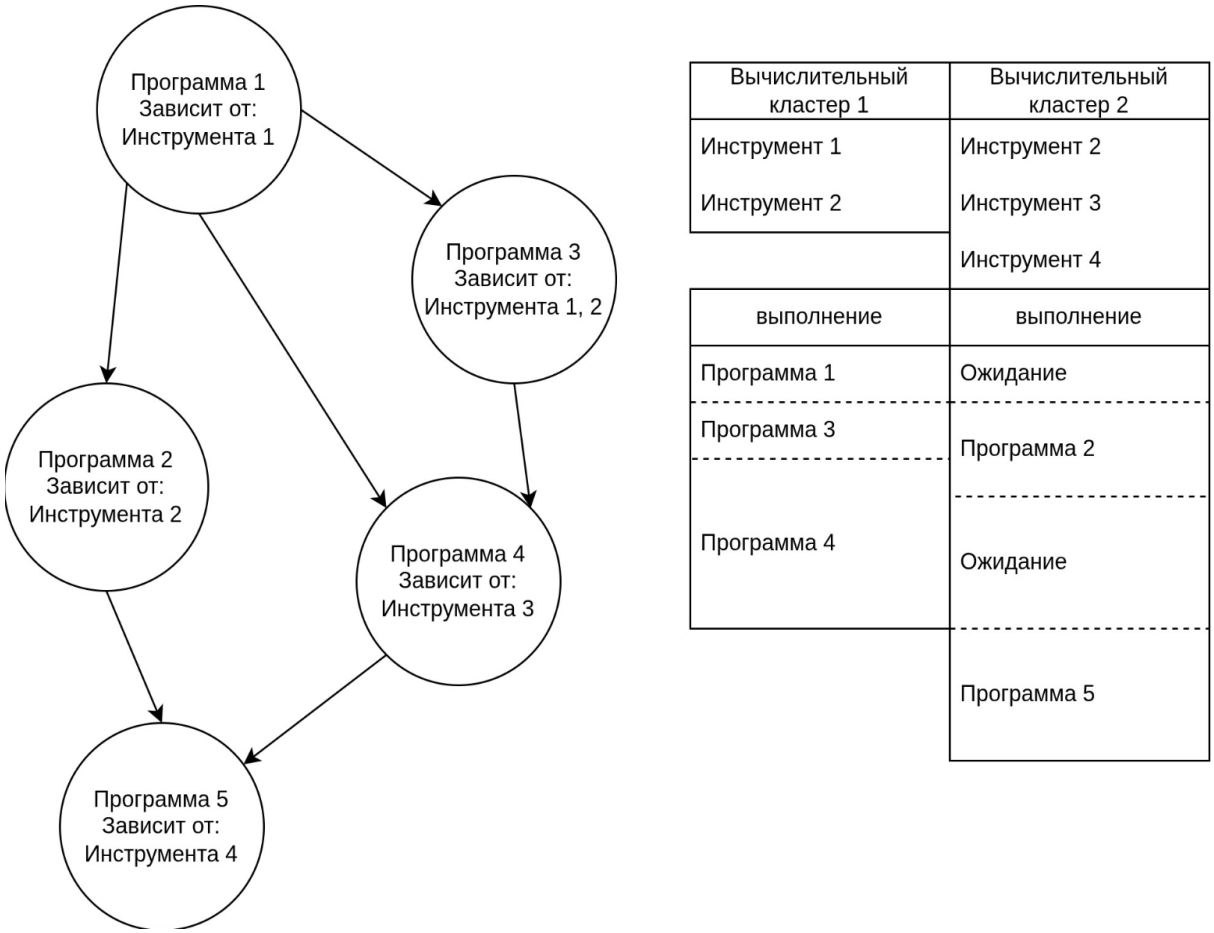


Рис. 1. Иллюстрация потока задач и его распределения на графе.

Система отслеживает состояние выполнения каждой задачи и обеспечивает передачу необходимых файлов только после успешного завершения решения предшествующих задач. Как правило, содержательные параметры научных программ, входные и выходные данные представлены именно фай-

лами и по соответствующим типам файлов образуют группу по различным вариантам обработки данных. Пользователь формирует описание вычислительного потока, которое преобразуется в структурированный формат. Генератор конфигурационных данных извлекает параметры программного обеспечения и создает машиночитаемое описание в формате JSON. Программа распределения задач использует полученные данные для формирования оптимального расписания и размещения задач по кластерам.

2. ОБЗОР МЕТОДОВ ИЗВЛЕЧЕНИЯ ПАРАМЕТРОВ

2.1. Критерии обзора

Для проведения систематического обзора инструментов анализа программного обеспечения был сформирован набор критериев, описывающих ключевые требования к системе автоматизированного развертывания в распределенной вычислительной среде. Выбор таких критериев обусловлен необходимостью интеграции инструмента в существующий конвейер оркестрации задач и минимизации ручного вмешательства на этапе конфигурирования.

Первостепенное значение имеет уровень автоматизации процесса извлечения параметров. Способность инструмента работать без предварительной ручной разметки кода или документации определяет возможность его применения для массового развертывания программного обеспечения на кластерах. Инструменты, требующие декларативного описания параметров непосредственно в исходном коде, накладывают ограничения на использование сторонних программных пакетов, модификация которых нежелательна или невозможна.

Формат вывода данных выступает вторым критическим критерием. Наличие структурированного представления результатов в машиночитаемом формате, таком как JSON, или формате, который возможно преобразовать в него, является обязательным условием для последующей обработки системой управления задачами.

Источники данных, анализируемые инструментом, определяют полноту извлекаемой информации. Поддержка анализа сопроводительной документации, man-страниц и исходного кода одновременно позволяет охватить различ-

ные способы описания параметров в программных продуктах. Документация часто содержит сведения о параметрах, которые не очевидны из статического анализа кода, включая ограничения на значения и семантические описания.

Возможность программного вызова через интерфейс командной строки или программный интерфейс критична для встраивания инструмента в автоматизированный конвейер развертывания. Инструменты, предназначенные исключительно для интерактивного использования, не могут быть интегрированы в процессы непрерывной интеграции и автоматического тестирования.

2.2. Сравнение существующих методов

Инструменты CarpetFuzz [4] и FuzzGen [5] разработаны для решения задач безопасности и тестирования программного обеспечения, а не для автоматизации развертывания в распределенных вычислительных средах. В табл. 1 представлены результаты сравнения инструментов извлечения параметров. (зеленый положиельная характеристика, красный отрицательная)

Табл. 1. Сравнительный анализ инструментов извлечения параметров.

Инструмент	Автоматизация	JSON	Документация	API/CLI	Разметка	Исходный код	Ограничения
CarpetFuzz	+	-	+	+	-	-	Для тестирования на уязвимости
FuzzGen	+	-	-	+	-	+	Требует исходный код
ArgParse	-	+	-	+	+	+	Ручная декларация
Click/Typer	-	+	-	+	+	+	Фремворки
Sphinx/ Doxygen	-	+	+	-	+	+	Генерация документации

CarpetFuzz специализируется на фаззинг-тестировании путем сопоставления документации с реализацией функций, что позволяет выявлять уязвимости и несоответствия в обработке аргументов. FuzzGen ориентирован на генерацию тестовых оберток для программ с использованием анализа исходного кода. Оба инструмента обеспечивают высокий уровень автоматизации и не требуют ручной разметки, однако их архитектура не предусматривает формирования структурированных данных о параметрах запуска, пригодных для последующей обработки системой оркестрации задач.

Разработка собственного метода извлечения параметров обусловлена необходимостью получения машиночитаемых конфигурационных данных в формате JSON для интеграции с системой распределенного запуска. Существующие решения либо требуют ручной декларации параметров в коде, либо не предоставляют программный интерфейс для автоматизации, либо ориентированы на генерацию документации вместо конфигурационных файлов. Предлагаемый подход сочетает автоматизацию анализа документации через языковые модели с возможностью программного вызова и выводом структурированных данных, что позволяет устранить выявленные недостатки и обеспечить автоматизацию процесса размещения программного обеспечения в гетерогенной вычислительной среде.

3. МЕТОД АВТОМАТИЗАЦИИ ИЗВЛЕЧЕНИЯ ПАРАМЕТРОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Предложенный подход основан на использовании языковых моделей большого размера для генерации обучающих данных и последующем дообучении компактной трансформерной архитектуры, способной функционировать без специализированных графических ускорителей. Подобная двухэтапная стратегия позволяет сочетать высокую точность извлечения параметров с практической применимостью в реальных вычислительных средах, где доступ к GPU-ресурсам может быть ограничен или экономически нецелесообразен. Разработанный метод состоит из четырех последовательных этапов, представленных на рис. 2, каждый из которых решает определенную задачу в процессе создания инструмента автоматизации.

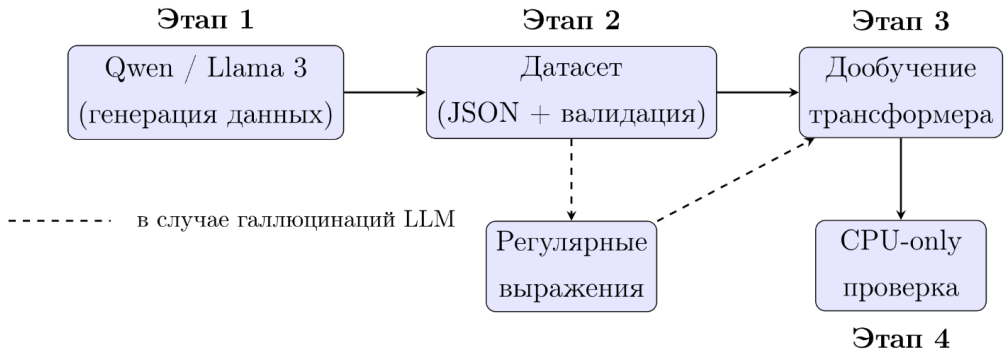


Рис. 2. Этапы реализации инструментов автоматизации на основе 4 этапов.

3.1. Этап формирования «сырого» датасета

Первоначальный этап предполагает сбор и систематизацию документации программного обеспечения, предназначенного для анализа. Источниками данных выступают сопроводительные материалы программных пакетов, включая man-страницы, файлы README, документацию в форматах reStructuredText и Markdown, а также комментарии, встроенные в исходном коде. Выбор разнообразных источников обусловлен необходимостью охвата различных способов описания параметров в программных продуктах.

Формирование «сырого» датасета включает следующие процедуры:

1) автоматизированный сбор документации из репозиториев программного обеспечения;

2) нормализация текстовых данных (удаление форматирования, унификация кодировок);

3) сегментация документов на логические блоки, соответствующие описаниям отдельных параметров;

4) метаданные о источнике документа, версии программного обеспечения и дате публикации [6].

Объем «сырого» датасета определяется количеством программных пакетов, подлежащих анализу в рамках системы распределенного запуска. Для обеспечения репрезентативности выборки рекомендуется включать программы с различными способами задания параметров: флаги командной строки, конфигурационные файлы, переменные окружения. Минимальный рекомен-

дуемый объем составляет 500–1000 образцов документации, что обеспечивает достаточное разнообразие лингвистических конструкций для последующего обучения модели.

3.2. Этап разметки данных языковой моделью

Второй этап является ключевым компоненом метода, где осуществляется автоматизированная генерация структурированных описаний параметров на основе текстов документации. Для решения этой задачи используются предварительно обученные языковые модели большого размера, такие как Qwen или Llama 3. Выбор данных архитектур обусловлен их способностью к пониманию контекста и извлечению семантических связей из неструктурированного текста.

Процесс разметки включает следующие шаги:

1) формирование промптов для языковой модели, содержащих инструкцию по извлечению параметров и пример желаемого формата вывода;

2) последовательная обработка документов сырого датасета через API языковой модели;

3) парсинг полученных ответов и приведение их к единой схеме представления данных;

4) проверка правильности структуры JSON-файла;

5) первичная автоматическая валидация с использованием запуска программ в изолированной среде.

Выходные данные этапа представляют собой JSON-объекты, содержащие для каждого параметра следующие поля: имя параметра, тип значения, значение по умолчанию, текстовое описание и ограничения на допустимые значения. Подобная структура соответствует требованиям системы оркестрации задач и позволяет непосредственно использовать извлеченные данные для генерации конфигурационных файлов.

Важным аспектом данного этапа является управление качеством генерации. Языковые модели склонны к галлюцинациям – формированию правдоподобных, но фактически неверных утверждений. Для минимизации этого риска применяются стратегия множественной генерации с последующим согласованием результатов, а также включение в промпт явных указаний на необходимость описания источника информации в тексте документации. Кроме того, ис-

пользуется подход с попыткой запуска программы по сформированному json и немедленному ее прекращению для проверки минимальной работоспособности.

3.3. Этап ручной валидации датасета

Третий этап предполагает экспертную проверку сгенерированных разметок для обеспечения достоверности обучающих данных. Ручная валидация осуществляется специалистами, обладающими знаниями в области системного программирования и анализа программного обеспечения.

Процедура валидации включает:

- сверку извлеченных параметров с исходным текстом документации;
- проверку корректности типов значений (целочисленные, строковые, булевы, пути к файлам);
- верификацию значений по умолчанию путем сопоставления с примерами использования в документации;
- контроль полноты описания ограничений на значения параметров.

Для формализации процесса валидации разработан чек-лист, содержащий критерии оценки качества каждой разметки. Каждый параметр оценивается по пятикомпонентной метрике полноты описания, представленной в следующем подразделе. Параметры, не прошедшие валидацию, возвращаются на этап разметки для повторной обработки скорректированными промптами. Результатом этапа становится верифицированный датасет, пригодный для использования в качестве обучающей выборки при дообучении трансформерной модели.

3.4. Этап дообучения трансформерной модели

Завершающий этап метода предполагает дообучение компактной трансформерной архитектуры на верифицированном датасете. Выбор легкой модели (не более 1 млрд параметров) обусловлен требованием возможности развертывания программы на центральных процессорах без использования графических ускорителей. Подобное ограничение критично для развертывания инструмента непосредственно на управляющих узлах системы распределенного запуска, где выделение специализированных вычислительных ресурсов

нецелесообразно, тем более, что сервера исследовательских групп могут не иметь GPU, что делает проект бесполезным для такого типа серверов.

Для дообучения применяется методика LoRA (Low-Rank Adaptation) [7], позволяющая эффективно адаптировать предварительно обученную модель к конкретной задаче при минимальных вычислительных затратах. Суть метода заключается в добавлении низкоранговых матриц к слоям внимания исходной модели, при этом основные веса модели остаются замороженными. Подобный подход снижает объем требуемой памяти и ускоряет процесс обучения.

Конфигурация дообучения включает следующие параметры:

- базовая архитектура: трансформер с 6–8 слоями внимания;
- размерность скрытого представления: 512–768;
- ранг матриц LoRA: 8–16;
- размер пакета: 16–32 образца;
- количество эпох: 3–5 с ранней остановкой по валидационной метрике.

Валидационная выборка формируется из части верифицированного датасета и используется для контроля переобучения. Обучение прекращается при достижении плато на валидационной метрике или при превышении максимального количества эпох.

3.5. Регулярные выражения при галлюцинациях нейросетевой модели

Для решения проблем, когда нейросеть добавляет лишние слова, но при этом выдает json-файл как часть сгенерированных слов, применяются регулярные выражения, позволяющие вычленить смысловую часть из варианта ответа, предложенного нейросетью.

4. РЕАЛИЗАЦИЯ ПРЕДЛОЖЕННОГО МЕТОДА

4.1. Метрики оценки качества

Для количественной оценки качества извлечения параметров разработана двухкомпонентная метрика, учитывающая как структурную корректность выходных данных, так и их семантическую точность.

Интегральная метрика качества вычисляется по формуле

$$Q = \alpha \cdot S_{struct} + (1 - \alpha) \cdot S_{sem},$$

где α = 0.6 — весовой коэффициент структурной корректности, S_{struct} — оценка структурного соответствия, S_{sem} — оценка семантической точности.

Структурная корректность S_{struct} оценивает соответствие выходных данных заданной схеме JSON и включает: 1) наличие всех обязательных ключей в объекте параметра; 2) соответствие типов значений, объявленным в схеме; 3) корректность вложенности структур данных.

Семантическая точность S_{sem} отражает содержательную правильность извлеченной информации: 1) соответствие типов значений фактическому назначению параметров; 2) точность значений по умолчанию относительно документации; 3) полнота описания ограничений на значения.

Дополнительно вводится индекс полноты описания параметров

$$C = \frac{\sum w_k \cdot I_k}{\sum w_k},$$

где $I_k \in \{0, 1\}$ — наличие k-го поля в извлеченном описании, w_k — весовой коэффициент поля.

Компоненты индекса полноты:

- I_1: имя параметра (w_1 = 1.0);
- I_2: тип значения (w_2 = 0.8);
- I_3: значение по умолчанию (w_3 = 0.7);
- I_4: текстовое описание (w_4 = 0.5);
- I_5: ограничения на значения (w_5 = 0.9).

Подобная система метрик позволяет проводить детальный анализ качества работы модели на различных аспектах задачи и выявлять направления для улучшения метода.

4.2. Результаты тестирования

В ходе экспериментального исследования была проведена оценка эффективности извлечения параметров командной строки с использованием языковой модели Qwen2.5-7B, базовой предобученной версии трансформера Qwen2.5-0.5B (Base), специализированного адаптера (LoRA), дообученного на данных man-страниц, и исходной модели (Qwen2.5-0.5B). Полученные результаты демонстрируют кардинальное превосходство адаптированного решения:

метрика точности (accuracy) для конфигурации LoRA достигла значения 0.867 ± 0.340, что более чем в три раза превышает показатели базовой модели Qwen2.5-0.5B (0.267 ± 0.442) и исходного варианта Qwen2.5-7B, показавшего низкий результат 0.641 ± 0.321. Аналогичная динамика наблюдается в показателях полноты (completeness), где метод LoRA обеспечил значение 0.715 ± 0.290 против 0.236 ± 0.210 у базы и критически низких 0.140 ± 0.246 у необученной модели, что свидетельствует о неспособности архитектуры «из коробки» корректно интерпретировать жесткую структуру технической документации без предварительной настройки весов.

Низкие показатели исходной модели Qwen2.5-7B подтверждают гипотезу о том, что универсальные языковые модели, не прошедшие адаптацию на репрезентативной выборке специфических форматов описания аргументов, склонны к ошибкам сегментации и не могут надежно выделять семантические связи между опциями и их параметрами в контексте системной документации.

Таким образом, можно заключить, что применение техники эффективной донастройки (LoRA) на специализированном корпусе man-страниц является критически необходимым условием для создания работоспособной системы автоматизированного развертывания программного обеспечения в гетерогенных вычислительных средах. Достигнутый уровень точности 0.867 и полноты 0.715 подтверждает пригодность разработанного подхода для генерации машиночитаемых конфигурационных файлов в формате JSON. В то же время выявленная недостаточность обобщающей способности модели на текущем этапе диктует необходимость расширения обучающей выборки за счет других типов документации и исходного кода, что составит основное содержание дальнейших исследований, направленных на создание универсального инструмента, способного агрегировать параметры из любых доступных источников проекта без потери качества структурирования данных.

ЗАКЛЮЧЕНИЕ

За счет применения методики LoRA упрощена задача процесса конфигурирования программного обеспечения для распределенных вычислительных сред. Проведенный систематический обзор существующих инструментов статического анализа и генерации документации (CarpetFuzz, FuzzGen, Sphinx

и др.) позволил сформулировать набор критических требований, которым должны удовлетворять решения по автоматизации. Требованиям полностью не удовлетворяет ни одно из решений, рассмотренных в обзоре. На основе выявленных ограничений был разработан метод автоматического извлечения параметров запуска с использованием больших языковых моделей. Ключевым результатом стали формирование специализированного датасета на основе man-страниц и успешная реализация прототипа системы дообучения трансформерных архитектур (Qwen, Llama 3). Экспериментально подтверждено, что адаптация модели под предметную область технической документации позволяет достичь метрики точности (accuracy) на уровне 0.867, что делает предложенный подход пригодным для практического применения при помещении нового кода в вычислительные кластерные системы. Реализованный метод обеспечивает работу на центральных процессорах управляющих узлов без необходимости использования графических ускорителей.

Однако предложенная реализация метода имеет ряд ограничений: архитектура системы пока требует участия человека на этапе валидации сгенерированных разметок, что замедляет процесс формирования обучающих выборок для новых пакетов. В связи с этим дальнейшая работа будет сосредоточена на расширении источников данных и повышении степени автономности.

СПИСОК ЛИТЕРАТУРЫ

1. *Suter F. et al*. A terminology for scientific workflow systems // Future Generation Computer Systems. 2026. Vol. 174. P. 107974. https://doi.org/10.1016/j.future.2025.107974

2. *da Silva R.F. et al.* Workflows Community Summit 2024: Future Trends and Challenges in Scientific Workflows: tech. rep. ORNL/TM-2024/3573. Oak Ridge: Oak Ridge National Laboratory, 2024.

3. *Санников Т.В., Сальников А.Н.* Обработка потока задач с зависимостями на нескольких вычислительных кластерах // Параллельные вычислительные технологии XIX Всероссийская конференция с международным участием (ПаВТ'2025). Челябинск: Изд-во ЮУрГУ, 2025. С. 270–283. https://doi.org/10.14529/pct2025.

4. *Wang D., Li Y., Zhang Z., Chen K.* CarpetFuzz: Automatic Program Option

Constraint Extraction from Documentation for Fuzzing // Proc. of the 32nd USENIX Security Symposium. Anaheim: USENIX Association, 2023. P. 2847–2864.

5. *Ispoglou K., Austin D., Mohan V., Payer M.* FuzzGen: Automatic Fuzzer Generation // Proc. of the 29th USENIX Security Symposium (USENIX Security 20). Boston: USENIX Association, 2020. P. 1001–1018.

6. *Wilkinson S.R. et al.* Applying the FAIR principles to computational workflows // Scientific Data. 2025. Vol. 12, No. 1. Art. 328. https://doi.org/10.1038/s41597-025-04451-9

7. *Hu E.J. et al.* LoRA: Low-Rank Adaptation of Large Language Models // Proc. of the International Conference on Learning Representations (ICLR). 2022. 16 p.

URL: https://openreview.net/forum?id=nZeVKeeFYf9 (дата обращения: 28.03.2026).

METHODS FOR THE AUTOMATED EXTRACTION OF PROGRAM PARAMETERS AND DESCRIPTIONS FOR THEIR INTEGRATION INTO COMPUTING SYSTEMS

T. V. Sannikov[1] [0009-0004-1144-8836], A. N. Salnikov[2] [0000-0001-8669-9905]

[1, 2]*Lomonosov Moscow State University, Moscow, Russia*

[2]*Federal Research Center for Information Technologies, Russian Academy of Sciences, Moscow, Russia*

[1]timohaj1@yandex.ru, [2]salnikov@cs.msu.ru

Abstract

This article addresses the problem of coordinating heterogeneous software tools in heterogeneous distributed application execution environments. Here, manually configuring launch parameters for newly installed programs on a computing cluster (such as command-line switches, environment variable values, and configuration file settings) poses significant challenges for domain researchers due to the large volume of utility information and the need to store and aggregate information in a fixed format. We propose a method for the automated extraction of launch parameters based on a hybrid neural network training architecture that combines the

generation of training samples using large language models with the subsequent fine-tuning of a compact transformer encoder. This approach eliminates the need for expensive graphics accelerators by applying the Low-Rank Adaptation (LoRA) technique to models with up to 1 billion parameters, enabling model execution (inference) on standard CPUs in control nodes. To formalize the quality of extraction, a two-component metric has been developed that aggregates the structural correctness of the output JSON schema (the presence of required fields and program parameter types in the obtained data) and the semantic accuracy of parameter values (correspondence with the description in the documentation). The experimental evaluation of the method focuses on a corpus of software package documentation (man pages, README files). The design results confirm the possibility of approximating the documentation analysis process with a compact model, which contributes to the automation of the software deployment lifecycle and the reduction of task flow management errors in distributed computing systems.

Keywords: *low-rank adaptation (LoRA), data extraction, source code analysis, launch automation, natural language processing (NLP), scientific workflow, high-performance computing (HPC).*

REFERENCES

1. Suter F. et al. A terminology for scientific workflow systems // Future Generation Computer Systems. 2026. Vol. 174. 107974. https://doi.org/10.1016/j.future.2025.107974

2. *da Silva R.F. et al.* Workflows Community Summit 2024: Future Trends and Challenges in Scientific Workflows: tech. rep. ORNL/TM-2024/3573. Oak Ridge: Oak Ridge National Laboratory, 2024.

3. *Sannikov T.V., Salnikov A.N.* Processing of Task Streams with Dependencies on Multiple Computing Clusters // Parallel Computational Technologies – 19th International Conference on Parallel Computing Technologies (PaVT'2025): short papers and poster descriptions. Chelyabinsk: South Ural State University Publishing House, 2025. P. 270–283. https://doi.org/10.14529/pct2025.

4. *Wang D., Li Y., Zhang Z., Chen K.* CarpetFuzz: Automatic Program Option Constraint Extraction from Documentation for Fuzzing // Proc. of the 32nd USENIX

Security Symposium. Anaheim: USENIX Association, 2023. P. 2847–2864.

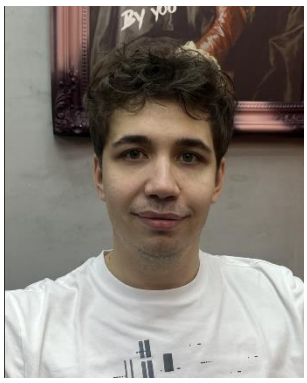
5. *Ispoglou K., Austin D., Mohan V., Payer M.* FuzzGen: Automatic Fuzzer Generation // Proc. of the 29th USENIX Security Symposium (USENIX Security 20). Boston: USENIX Association, 2020. P. 1001–1018.

6. *Wilkinson S.R. et al.* Applying the FAIR principles to computational workflows // Scientific Data. 2025. Vol. 12, No. 1. Art. 328.
https://doi.org/10.1038/s41597-025-04451-9

7. *Hu E. J. et al.* LoRA: Low-Rank Adaptation of Large Language Models // Proc. of the International Conference on Learning Representations (ICLR). 2022. 16 p.
URL: https://openreview.net/forum?id=nZeVKeeFYf9 (accessed 28.03.2026).

СВЕДЕНИЯ ОБ АВТОРАХ

.



САННИКОВ Тимофей Владимирович -- студент магистратуры Московского государственного университета имени Ломоносова. Область научных интересов: архитектура языковых моделей, высокопроизводительные системы.

Timofei Vladimirovich SANNIKOV -- a first-year master's student at Lomonosov Moscow State University.Research interests: language model architecture, scientific workflows, and high-performance systems.

email: timohaj1@yandex.ru;

ORCID: 0009-0004-1144-8836

САЛЬНИКОВ Алексей Николаевич – кандидат физико-математических наук, доцент кафедры Автоматизации систем вычислительных комплексов Московского государственного университета имени М.В. Ломоносова.

Научные интересы: параллельные и распределенные вычисления, интеллектуальный анализ данных, анализ сетей вычислительных кластеров.

Alexey Nikolaevich SALNIKOV – Philosophy doctor of Physics and Mathematics, Associate Professor, Lomonosov Moscow State University, department of Computer Science.

Research interests: parallel and distributed computations, data mining, computer cluster interconnections analysis.

email: salnikov@cs.msu.ru;

ORCID: 0000-0001-8669-9905

Материал поступил в редакцию 16 марта 2026 года