Адаптивная система парсинга веб-ресурсов на основе больших языковых моделей с использованием семантического кэширования

Д. С. Мурадян
Санкт-Петербургский государственный университет
muradyan.denis@inbox.ru

О. А. Шутько
Санкт-Петербургский Федеральный исследовательский центр Российской академии наук
olegshutko54@gmail.com

Ф. В. Бушмелев
Санкт-Петербургский Федеральный исследовательский центр Российской академии наук
fvb@dscs.pro

***Аннотация*. Информация, публикуемая на Интернет-ресурсах, являются важным источником данных для функционирования множества современных сервисов и используется в задачах сбора, анализа, мониторинга, бизнес-аналитики и других. Однако, отсутствие API, высокая вариативность и частые изменения структуры сайтов существенно усложняют разработку и требуют постоянной актуализации парсеров. В последние годы большие языковые модели (Large Language Models, LLM) демонстрируют высокую эффективность в задачах анализа текста и генерации программного кода, что открывает новые возможности для автоматизации анализа данных из веб-ресурсов. В данной работе предлагается архитектура адаптивной системы извлечения веб-данных, расширяющая возможности традиционных методов веб-парсинга за счет внедрения и адаптации посредством LLM. Решение поддерживает два режима работы: генерацию Python-скриптов для парсинга страницы и структурирование информации из текста страницы. Для оптимизации затрат токенов LLM на генерацию парсеров, внедрен механизм семантического кэширования, позволяющий повторно использовать ранее сгенерированные парсеры на основе поиска похожих запросов в векторном пространстве. Проведена экспериментальная оценка системы на основе бенчмарка LiveWeb-IE. Результаты показали, что предложенная система с точностью 90% извлекает релевантные запросу данные. Дополнительное кэширование сокращает время ответа на 1 запрос более чем в 10 раз.**

Ключевые слова: большие языковые модели; веб-парсинг; автоматизация анализа данных; семантическое кэширование; генерация кода

I. Введение

Современные веб-сервисы являются неоднородными: наряду со статическими страницами широко используются динамические веб-приложения [9]. Структура DOM, расположение элементов и механизмы загрузки данных могут существенно различаться между различными ресурсами и даже в разных пользовательских сценариях в рамках одной страницы [2]. В результате, задача извлечения данных из подобных источников требует периодической адаптации методов парсинга и динамической актуализации к особенностям, изменениям на конкретных сайтах и структурам их разметки [2].

Работа выполнена при финансовой поддержке в рамках проекта по государственному заданию СПБ ФИЦ РАН № FFZF-2025-0006

Вместе с этим, развитие больших языковых моделей (LLM) в последнее время позволило повысить эффективность в задачах анализа текстов и генерации программного кода [1, 4]. В контексте извлечения веб-данных применение LLM позволяет реализовать автоматизированные методы обработки страниц, способные адаптироваться к различным структурам и формировать релевантный запросу ответ по информации с сайта [4, 6].

В существующих подходах к применению LLM для извлечения веб-данных можно выделить два направления: структурированное извлечение информации [4, 6] из содержимого страницы и генерацию программного кода парсера с последующим использованием [5].

Несмотря на высокую гибкость подобных методов, обращение к LLM для каждого нового запроса связано с дополнительными вычислительными затратами и увеличением времени отклика [3].

С целью повышения временной эффективности обработки данных, снижения удельных вычислительных затрат, в работе предлагается архитектура адаптивной системы, использующая механизм семантического кэширования. Сгенерированные парсеры сохраняются и могут повторно переиспользоваться при обработке семантически близких запросов без повторного обращения к языковой модели [3].

II. Архитектура решения

Предлагаемая система реализует автоматизированный конвейер извлечения данных из веб-источников и ориентирована на работу как со статическими страницами, так и с динамическими веб-приложениями [2, 9].

В предлагаемой системе реализованы два режима обработки, далее обозначаемые как “Structuring” и “Codegen”. Режим Structuring предполагает, что модель получает очищенный текст страницы и возвращает структурированную информацию [4, 6]. Второй подход – “Codegen”, предполагающий генерацию программного кода парсера, который затем запускается для извлечения данных. [5]

На вход подаются URL адрес веб-страницы и текстовый запрос пользователя на естественном языке, определяющий необходимую для автоматизированного

сбора извлечения информацию, а также ее формат формирования страницы (Статический / Динамический).

Шаг 1 – “Сбор”: На первом этапе выполняется получение содержимого страницы с автоматическим выбором способа загрузки. Для динамических ресурсов применяется браузерный рендеринг, обеспечивающий построение актуального на момент завершения рендеринга DOM после выполнения клиентских сценариев [9], для статических страниц используется прямое получение HTML. Выбор метода осуществляется на основе эвристического метода анализа документа, который выделяет признаки динамической подгрузки контента [9]. Таким образом, архитектура допускает автоматический выбор режима обработки в зависимости от типа ресурса.

Шаг 2 – “Анализ”: На втором этапе выполняется предварительная обработка HTML-разметки. Этот шаг необходим для сокращения объёма входного контекста при инференсе модели, что позволяет снизить вычислительные затраты и стоимость обработки, а в ряде случаев уменьшить задержку ответа [7]. В предлагаемом подходе используются две стратегии подготовки данных, соответствующие выбранному режиму извлечения. Для режима Structuring (прямого структурирования), в котором извлечение выполняется на основе текстового содержимого страницы, выполняется полная очистка, преобразующая ресурс в текстовое представление с удалением кодовой разметки HTML и сопроводительных блоков, таких как навигационные блоки, рекламные баннеры, мета информация о странице и другие нерелевантные фрагменты [10]. Для режима Codegen (генерации парсера) применяется облегченная очистка, сохраняющая DOM структуру HTML и ключевые элементы разметки, необходимые для формирования селекторов и извлечения целевых объектов. При обработке особо крупных страниц дополнительно учитываются ограничения длины контекста языковой модели: HTML-фрагменты могут быть автоматически сокращены до допустимого числа токенов, чтобы обеспечить корректную обработку запроса [7].

Особенностью системы является поддержка двух режимов интеграции с большими языковыми моделями. Режим Structuring предназначен для прямого извлечения информации из текстового содержимого страницы, в случаях, когда требуемые данные могут быть получены без анализа HTML-структуры. Подход возвращает структурированный ответ в формате JSON, что обеспечивает применимость при работе с разнообразными источниками, содержащими информацию в тексте [4,6]. В режиме Codegen модель генерирует исполняемый программный код парсера, который затем запускается для извлечения данных [5]. Этот режим ориентирован на многократное использование полученного решения и снижение зависимости от повторных обращений к LLM, что особенно важно с учётом вычислительных затрат и задержек инференса [3]. Вместе с этим, при обновлении структуры страницы и/или задачи парсинга существующий код можно будет актуализировать.

В ветке Codegen дополнительно используется автоматизированная система формирования контекста и и системных инструкций на основе few-shot-подхода, предназначенная для повышения качества генерируемого кода парсера и релевантности результата пользовательскому запросу [1, 5]. Механизм реализован в виде вспомогательной LLM-компоненты, которая анализирует облегченный HTML и пользовательский запрос, после чего выделяет в верстке страницы релевантные структурные признаки — теги, классы, атрибуты и селекторы, — указывающие на расположение требуемой информации. Кроме того, компонент генерирует системную инструкцию, описание полей и формат вывода, задающие способ представления результата. Таким образом, ответственность разделяется между двумя LLM-компонентами: первая адаптирует требования и формирует контекст, а вторая генерирует код парсера на основе полученных подсказок, что повышает устойчивость и воспроизводимость результатов [1, 5].

Для повышения эффективности обработки используется механизм семантического кэширования. Пользовательский запрос совместно с адресом ресурса преобразуется в векторное представление с помощью модели Sentence-BERT [11]. Поиск ранее сгенерированных решений выполняется в векторном хранилище ChromaDB, а метаданные и ссылки на сохранённые парсеры хранятся в базе данных SQLite [8]. При обнаружении семантически близкого запроса система может повторно использовать существующий парсер без повторной генерации, что уменьшает суммарные вычислительные затраты и время отклика [3]. Для практического применения реализованы несколько интерфейсов доступа: сервер с REST API, демонстрационное приложение на базе Gradio, развёрнутое в Hugging Face Spaces, а также веб-интерфейс, для интерактивной работы с системой.

III. Эксперимент

Для оценки качества работы предложенной системы использовалось репрезентативное подмножество бенчмарка LiveWeb-IE [12], включающее 30 веб-страниц. Использование подмножества бенчмарка позволяет сделать эксперимент более воспроизводимым и сопоставимым. Выборка была сбалансирована по двум типам задач извлечения, поскольку данные типы в совокупности покрывают все остальные уровни сложности. В бенчмарке они имеют названия Type II и Type IV. При этом подмножество охватывает шесть предметных областей: академическую, электронную коммерцию, пищевую, структурированные базы данных, насыщенные текстом и смешанные домены, по 5 примеров в каждой. Такое распределение обеспечивает разнообразие как HTML-структур, так и семантики извлекаемого контента.

Выбор типов Type II и Type IV обусловлен тем, что они представляют практически значимые и достаточно сложные сценарии извлечения данных. Задачи типа Type II ориентированы на извлечение нескольких атрибутов с одиночными значениями, тогда как задачи типа Type IV дополнительно включают извлечение списков и повторяющихся структурных блоков. Таким образом, экспериментальная выборка позволяет оценить способность системы работать как с относительно простыми структурированными страницами, так и с более сложными случаями много атрибутного извлечения. Тестирование выполнялось с использованием языковой модели OpenAI GPT-5 nano.

Для каждого примера использовались веб-страница, пользовательский запрос для поиска и эталонный структурированный ответ, заданные в рамках бенчмарка. Система получала URL страницы и текстовый запрос на естественном языке, после чего формировала структурированный результат извлечения. Качество оценивалось путём ручного сравнения ответа системы с эталонной разметкой.

Для оценки полученного ответа рассматривались как временные, так и качественные характеристики работы системы. К временным характеристикам относились время выполнения запроса в режиме Codegen без использования кэша, время повторного выполнения запроса при наличии кэша, а также время обработки запроса в режиме Structuring. Режим Structuring использовался в качестве базового метода сравнения. Таким образом, данное сопоставление позволяет оценить эффективность решения, а также, влияние механизма семантического кэширования на скорость повторного извлечения данных.

В эксперименте для режима Codegen рассматривались два сценария запуска. Cold start соответствует обработке запроса при отсутствии подходящего парсера в кэше, когда требуется полный цикл генерации и выполнения нового решения. Warm start соответствует повторной обработке запроса при наличии ранее сгенерированного парсера, который может быть повторно использован без повторного обращения к этапу генерации кода.

Для количественной оценки использовались две метрики. Первая метрика, полнота извлечения (field_acc), определялась как отношение числа полей, присутствующих в ответе системы, к общему числу полей в эталонном ответе. Вторая метрика, точность значений (value_acc), определялась как отношение числа полей, значения которых корректно совпадают с эталоном, к общему числу полей в эталонном ответе. Первая метрика характеризует способность системы находить требуемые элементы ответа, тогда как вторая отражает точность извлечения значений.

Итоговые значения метрик рассчитывались по всей выборке, а также отдельно для различных типов задач и предметных областей. Такой протокол позволяет оценить не только общее качество извлечения, но и устойчивость предложенного подхода к различным видам HTML-структур и содержательных сценариев.

В табл. 1 представлены результаты экспериментальной оценки предложенной системы, включающие временные характеристики работы и метрики качества извлечения данных. Временные показатели отражают эффективность трёх сценариев обработки: генерации парсера при холодном запуске (Codegen cold), повторного выполнения запроса при наличии кэша (Codegen warm) и прямого структурирования страницы (Structuring). Для каждого сценария приведены средние, медианные значения и стандартное отклонение, что позволяет оценить не только общий уровень временных затрат, но и стабильность работы системы.

ТАБЛИЦА I.

Показатель таблицы	**Среднее значение**	**Медианное значение**	**Стандартное отклонение**
Время Codegen (cold), секунды	72.9813	68.7252	19.5232
Время Codegen (warm), секунды	5.1183	3.8802	3.7434
Время Structuring, секунды	40.2525	27.5815	32.8363
Field Accuracy (полнота полей)	0.9290	1.0000	0.1385
Value Accuracy (точность полей)	0.8878	0.8730	0.1258
Агрегированный показатель качества	0.9084	0.9365	0.1328
Разброс средних значений field_acc по источникам	-	-	0.1171
Разброс средних значений value_acc по источникам	-	-	0.1135
Разброс средних значений overall по источникам	-	-	0.1083

IV. Анализ результатов

Прежде всего, стоит обратить внимание на временные показатели работы системы. Результаты эксперимента показывают, что наиболее затратным по времени является режим Codegen cold start: среднее время выполнения составило 72.98 с, медианное до 68.73 с. Это связано с необходимостью выполнения полного цикла обработки, включающего генерацию нового парсера. При наличии кэша среднее время в режиме Codegen warm start снижается до 5.12 с, а медианное до 3.88 с, что соответствует ускорению примерно в 14 раз по среднему и в 18 раз по медиане. Таким образом, семантическое кэширование даёт существенный выигрыш при повторной обработке запросов.

Режим Structuring показал промежуточные временные характеристики: среднее время обработки составило 40.25 с, медианное — 27.58 с. По сравнению с Codegen cold этот режим работает быстрее за счёт отсутствия этапа генерации и исполнения программного парсера, однако уступает Codegen warm, поскольку при каждом запросе требует повторного обращения к языковой модели. В целом это подтверждает, что основное преимущество предложенной архитектуры проявляется именно в сценарии повторного использования ранее сгенерированных решений.

С точки зрения качества извлечения система продемонстрировала высокие результаты. Среднее значение field_acc составило 0.9290, а медианное — 1.0000, что указывает на высокую полноту извлечения и означает, что как минимум в половине примеров были найдены все поля эталонного ответа. Метрика value_acc составила 0.8878 в среднем и 0.8730 по медиане, что свидетельствует о достаточно высокой точности извлеченных значений. Агрегированный показатель качества достиг 0.9084 по среднему значению. Небольшой разброс метрик между источниками

дополнительно показывает, что предложенный подход сохраняет качество на веб-ресурсах различных типов.

Дополнительный анализ результатов по отдельным веб-источникам показывает, что предложенная система демонстрирует устойчивое качество извлечения данных на страницах различного типа. Несмотря на различия в структуре веб-ресурсах и способах представления информации, средние значения метрик качества составляют более 90 %, что подтверждает перспективность предлагаемого подхода.

V. Заключение

В статье предложена архитектура адаптивной системы парсинга и извлечения веб-данных, основанная на использовании больших языковых моделей и механизма семантического кэширования. Подход объединил в себе два способа обработки веб-страниц: прямое структурирование данных с использованием LLM и генерацию программного кода парсера, исполняемого для извлечения данных. Новизна работы заключается в двухрежимной архитектуре, которая позволяет адаптироваться к различным структурам и способам формирования страниц веб-ресурсов и извлекать требуемую информацию в запрашиваемом пользователем формате. В режиме генерации кода применяется механизм семантического кэширования, обеспечивающий повторное использование ранее сгенерированных парсеров при обработке семантически близких запросов. Экспериментальная оценка показала высокое качество извлечения данных для веб-ресурсов различных типов: агрегированный показатель качества составил 0,937. Использование кэширования позволяет сократить время обработки повторных запросов более чем в 10 раз по сравнению с холодным запуском.

Практическая значимость предложенного подхода связана с возможностью его применения в системах мониторинга открытых веб-источников, новостной и финансовой аналитики, а также в решениях для агрегации и структурирования данных из разнородных интернет-ресурсов. Дальнейшие исследования могут быть направлены на расширение набора поддерживаемых источников, совершенствование методов генерации парсеров, а также развитие механизмов интеллектуального кэширования для повышения масштабируемости системы.

Список литературы

[1] Brown Tom B., Mann Benjamin, Ryder Nick, Subbiah Melanie, Kaplan Jared, Dhariwal Prafulla, Neelakantan Arvind, Shyam Pranav, Sastry Girish, Askell Amanda, Agarwal Sandhini, Herbert-Voss Ariel, Krueger Gretchen, Henighan Tom, Child Rewon, Ramesh Aditya, Ziegler Daniel, Wu Jeffrey, Winter Clemens, Hesse Christopher, Chen Mark, Sigler Eric, Litwin Mateusz, Gray Scott, Chess Benjamin, Clark Jack, Berner Christopher, McCandlish Sam, Radford Alec, Sutskever Ilya, Amodei Dario. Language Models are Few-Shot Learners // Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 1877–1901.

[2] Cribbs J., Peters L. A Survey of Static Web Scraping Techniques // Journal of Web Engineering. 2019. Vol. 18, No. 3. P. 345–378.

[3] Dong A., Sun B. Accelerating LLM-based Code Generation through Caching Strategies // ACM Transactions on Software Engineering and Methodology. 2022. Vol. 48, No. 1. P. 1–23.

[4] Kalyan S., Rao V. Leveraging Large Language Models for Web Data Extraction // ACM Transactions on Internet Technology. 2023. Vol. 23, No. 4. P. 1–20.

[5] Kolluru R., Li X., Zhang Y. Hybrid Web Scraping with LLM Assistance // International Journal of Web Engineering. 2023. Vol. 20, No. 2. P. 150–167.

[6] Li X., Zhou Y., Wang L. Zero-Shot Web Scraping with GPT Models // Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics. 2024. P. 2000–2012.

[7] Liu Nelson F., Lin Kevin, Hewitt John, Paranjape Aakanksha, Bevilacqua Michele, Petroni Fabio, Liang Percy. Lost in the Middle: How Language Models Use Long Contexts // Transactions of the Association for Computational Linguistics. 2024. Vol. 12. P. 157–173.

[8] Pan James Jie, Wang Jianguo, Li Guoliang. Survey of Vector Database Management Systems // The VLDB Journal. 2024.

[9] Patil S., Gupta A. Modern Web Scraping with Selenium and Headless Browsers // International Journal of Web Technology. 2021. Vol. 12, No. 1. P. 45–58.

[10] Pomikálek Jan. Removing Boilerplate and Duplicate Content from Web Corpora: PhD Thesis. Brno: Masaryk University, 2011. 146 p.

[11] Reimers Nils, Gurevych Iryna. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks // Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. 2019. P. 3982–3992.

[12] Yang Seungbin, Kim Jihwan, Choi Jaemin, Kim Dongjin, Yang Soyoung, Park ChaeHun, Choo Jaegul. LiveWeb-IE: A Benchmark For Online Web Information Extraction // arXiv preprint arXiv:2603.13773. 2026.