

Генерация API-документации (OpenAPI/Swagger) по текстовому описанию с использованием нейросети GPT-5.1

Саяпин Никита Анатольевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Аннотация

Целью исследования является демонстрация процесса генерации спецификаций OpenAPI (Swagger) по текстовому описанию REST-API с использованием мультимодальной нейросети GPT-5.1. В работе анализируются механизмы интерпретации моделью требований к серверным методам, параметрам запросов, структурам данных и кодам ответа. На основе пошаговых примеров рассматривается корректность сформированных YAML-документов и степень соответствия спецификаций исходному текстовому описанию.

Ключевые слова: API, документация, OpenAPI, Swagger, GPT-5.1, описание сервисов, нейросеть.

Generation of API documentation (OpenAPI/Swagger) based on a text description using the GPT-5.1 neural network

Sayapin Nikita Anatolyevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this study is to demonstrate the process of generating OpenAPI (Swagger) specifications from textual descriptions of REST APIs using the GPT-5.1 multimodal neural network. The paper examines how the model interprets high-level functional requirements, request parameters, response schemas, and endpoint semantics. Several examples of generated YAML specifications are analyzed to evaluate the accuracy and completeness of the produced documentation.

Keywords: API, documentation, OpenAPI, Swagger, GPT-5.1, service description, neural network.

1. Введение

1.1 Актуальность

Современные программные интерфейсы требуют точной и структурированной документации, обеспечивающей корректное взаимодействие между компонентами распределённых систем. Формат OpenAPI стал стандартом описания REST-служб благодаря строгой формализации, поддержке большого количества инструментов и тесной

интеграции с процессами разработки. Однако ручное составление спецификаций нередко сопровождается трудоёмкостью и высокой вероятностью несогласованности между документацией и реальной реализацией сервиса.

Использование крупных мультимодальных моделей, таких как GPT-5.1, открывает возможность преобразовывать естественно-языковые описания требований в формальные спецификации OpenAPI. Модель способна интерпретировать структуру API, выделять сущности, параметры и ответы, а также формировать корректную YAML-документацию. Такой подход снижает порог входа в проектирование интерфейсов и ускоряет этапы подготовки технической документации.

1.2 Обзор исследований

С. В. Николаев демонстрирует практику описания интерфейсов веб-сервиса с помощью OpenAPI-спецификации (Swagger), показывая, что единый формат позволяет автоматически генерировать интерактивную документацию, клиентские библиотеки и серверные заглушки, а также снижает количество ошибок при интеграции между подсистемами. В статье отдельно отмечается, что спецификация должна рассматриваться как центральный артефакт, вокруг которого строится весь цикл разработки и эксплуатации API [1]. Д. Н. Соснин рассматривает вопросы проектирования REST API и указывает, что качественная архитектура интерфейса тесно связана с его документированием: чёткие соглашения о ресурсах, методах, кодах ответов и форматах данных облегчают как реализацию, так и последующее использование API внешними и внутренними клиентами [2]. В. В. Воронин в работе по проектированию и разработке RESTful API подчёркивает важность единообразия и предсказуемости поведения интерфейса, а также необходимости предоставления разработчикам подробной документации, включающей описание ресурсов, примеры запросов и ответов. Отмечается, что наличие формального описания API является обязательным условием для построения масштабируемых распределённых систем [3]. Ю. А. Зотова рассматривает архитектуру REST API для взаимодействия с распределённой системой и показывает, что грамотное проектирование интерфейсов является ключом к надёжному информационному обмену между компонентами [4]. В более широком контексте обработки естественного языка Л. Л. Садовская приводит обзор методов NLP и демонстрирует, что современные модели способны решать задачи семантического анализа и извлечения структурированной информации из текстов, что открывает возможности для автоматизации преобразования естественно-языкового описания в формальные представления, включая элементы спецификаций и конфигураций [5].

1.3 Цель исследования

Цель исследования — продемонстрировать процесс генерации спецификаций API в формате OpenAPI/Swagger по текстовому описанию

требований к веб-сервису с использованием мультимодальной нейросети GPT-5.1 и оценить корректность интерпретации моделью структурных элементов API (эндпоинты, методы, параметры, схемы данных) по сравнению с традиционными подходами к документированию REST-интерфейсов.

2 Материалы и методы

В исследовании использовалась мультимодальная модель GPT-5.1, обладающая способностью преобразовывать описания сервисов, сформированные в виде обычного текста, в структурированную OpenAPI-спецификацию. Процесс рассматривался как последовательное преобразование семантического содержания: исходное описание определяло функциональные возможности API, а модель осуществляла их формализацию в формате YAML.

Анализировалась корректность интерпретации сущностей, маршрутов, типов данных, а также способность модели учитывать связи между объектами и особенности обработки ошибок. Все примеры генерировались в несколько проходов с дополнительными уточнениями, что позволяло наблюдать характер изменений документации при корректировке текстового описания.

3. Результаты

3.1 Генерация простой спецификации API по краткому описанию

Первый эксперимент был направлен на создание минимальной спецификации сервиса, описанного в нескольких предложениях. Формулировка содержала только общее назначение API и одну операцию.

Промпт:

«Сгенерируй спецификацию OpenAPI 3.0.

Описание:

Сервис управляет списком задач.

Нужно получить список всех задач через GET /tasks.

Ответ содержит массив объектов с полями id и title.

Формат — YAML.»

GPT-5.1 корректно создала структуру документа, включив корневые поля openapi, info и paths. Модель сформировала маршрут /tasks, описала метод GET и задала схему возвращаемого массива. Несмотря на минимальность исходных данных, структура получилась согласованной, а формат строго соответствовал стандарту OpenAPI 3.0. В документ также были добавлены поля summary и описание ответа, что повысило информативность спецификации (Рис. 1).

```
openapi: 3.0.3
info:
  title: Task Service API
  version: 1.0.0
  description: API для управления списком задач.

paths:
  /tasks:
    get:
      summary: Получить список всех задач
      description: Возвращает массив задач.
      responses:
        '200':
          description: Список задач успешно получен.
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  properties:
                    id:
                      type: integer
                      example: 1
                    title:
                      type: string
                      example: "Купить молоко"
                  required:
                    - id
                    - title
```

Рисунок 1 — Пример генерации простой спецификации

3.2 Формирование расширенной документации по детализированному описанию

Следующий эксперимент включал более развёрнутое текстовое описание, содержащее несколько операций, параметры запросов и структуру вложенных объектов

Промпт:

«Создай подробную документацию OpenAPI 3.0 в формате YAML.

Описание сервиса:

Пользователь может создавать задачи, редактировать их и менять статус.

Создание задач выполняется через POST /tasks. Тело запроса содержит title и description.

Изменение выполняется через PUT /tasks/{id}.

Статус меняется через PATCH /tasks/{id}/status, где передается новое значение status: todo, in_progress, done.

Все ответы возвращают объект задачи с полями id, title, description, status.»

Модель отразила в спецификации три маршрута, корректно выделила параметры пути, определила тело запросов POST, PUT и PATCH, и сформировала единую схему объекта задачи. В документе появились коды ошибок и типичные служебные описания, не указанные напрямую, но часто встречающиеся в практической документации. Это свидетельствует о способности GPT-5.1 применять контекстуальные знания о структуре OpenAPI (Рис. 2).

```
components:
  schemas:
    Task:
      type: object
      description: Объект задачи.
      properties:
        id:
          type: integer
          description: Уникальный идентификатор задачи.
          example: 1
        title:
          type: string
          description: Краткий заголовок задачи.
          example: "Купить продукты"
        description:
          type: string
          description: Подробное описание задачи.
          example: "Молоко, хлеб, яйца"
        status:
          type: string
          description: Текущий статус задачи.
          enum:
            - todo
            - in_progress
            - done
          example: "todo"
      required:
        - id
        - title
        - description
        - status
```

Рисунок 2 — Генерация расширенной спецификации

3.3 Генерация документации сложного сервиса с несколькими сущностями

Последний эксперимент касался многосущностного сервиса, имеющего связи между объектами и сложную структуру данных. Описание включало поведение пользователей, ресурсов и комментариев.

Промпт:

«Создай OpenAPI-спецификацию.

Описание:

Есть пользователи и их статьи.

Пользователь создаёт статьи через POST /users/{id}/articles.

Статья содержит title, body, дату публикации и массив комментариев.

Каждый комментарий содержит автора и текст.

Нужны операции:

GET /articles — список всех статей,

GET /articles/{id} — получить статью,

POST /articles/{id}/comments — добавить комментарий.

Формат — YAML.»

GPT-5.1 успешно выделила три сущности: пользователь, статья и комментарий. Модель создала повторно используемые схемы (components/schemas), что характеризует зрелое понимание модульности документации. Генерируемая спецификация включала вложенные структуры, соответствующие исходному описанию, и корректно связала маршруты с параметрами пути. Модель также добавила типовые элементы вроде описания ошибок 400 и 404, что повысило практическую ценность результата (Рис. 3,4).

```
paths:
  /users/{id}/articles:
    post:
      tags:
        - Users
      summary: Создать статью пользователя
      description: Создает новую статью для указанного пользователя.
      parameters:
        - name: id
          in: path
          required: true
          description: Идентификатор пользователя, которому принадлежит статья.
          schema:
            type: integer
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ArticleCreateRequest'
            example:
              title: "Первый пост"
              body: "Текст статьи..."
              published_at: "2025-12-05T10:15:00Z"
      responses:
        '201':
          description: Статья успешно создана.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Article'
              example:
```

Рисунок 3— Спецификация многосущностного API

```
/articles:
  get:
    tags:
      - Articles
    summary: Получить список всех статей
    description: Возвращает список всех статей с их комментариями.
    responses:
      '200':
        description: Список статей успешно получен.
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Article'
    example:
      - id: 1
        user_id: 10
        title: "Первый пост"
        body: "Текст статьи..."
        published_at: "2025-12-05T10:15:00Z"
        comments:
          - author: "Иван"
            text: "Отличная статья!"
      - id: 2
        user_id: 11
        title: "Второй пост"
        body: "Ещё немного текста..."
        published_at: "2025-12-06T09:00:00Z"
        comments: []
```

Рисунок 4 — Спецификация многосущностного API

4 Выводы

Исследование показало, что модель GPT-5.1 способна формализовывать текстовые описания программных интерфейсов и преобразовывать их в корректную документацию OpenAPI. Она уверенно интерпретирует структуру API, выделяет сущности и формирует схемы данных, а также учитывает контекст и стандарты оформления. Детализация исходного текста напрямую влияет на полноту полученной спецификации: при кратком описании формируется минимальный набор полей, тогда как развернутый текст позволяет получить полноценную рабочую документацию.

Модель демонстрирует высокий уровень устойчивости к вариативности формулировок, но не учитывает существующие стандарты проекта или требования к версиям API, поэтому сгенерированная документация требует последующей проверки инженером. Тем не менее GPT-5.1 выполняет значительный объем работы на ранних этапах проектирования интерфейсов, ускоряя подготовку документации и снижая количество ошибок.

Библиографический список

1. Николаев С. В., Пасюков А. А., Якимов А. С., Баженов Р. И. Документирование API веб-сервиса с помощью OpenAPI спецификации // Постулат. 2018. № 1(27). С. 55.
2. Соснин Д. Н., Лукинов В. Л. Проектирование REST API // Современные проблемы телекоммуникаций: материалы Российской научно-технической конференции, Новосибирск, 20–21 апреля 2022 года. Новосибирск: Сибирский государственный университет телекоммуникаций и информатики, 2022. С. 243–248.
3. Воронин В. В., Громов П. А. Проектирование и разработка RESTful API веб-сервера // Информационные технологии XXI века: сборник научных трудов / отв. за выпуск Е. А. Шеленок. Хабаровск: Тихоокеанский государственный университет, 2015. С. 474–480.
4. Зотова Ю. А., Котилевец И. Д. Разработка архитектуры REST API для взаимодействия с сервисами приложения // Информационные технологии и математическое моделирование систем 2018: труды международной научно-технической конференции, Одинцово, 19–21 ноября 2018 года. Одинцово: Центр информационных технологий в проектировании РАН, 2018. С. 61–65.
5. Садовская Л. Л., Гуськов А. Е., Косяков Д. В., Мухамедиев Р. И. Обработка текстов на естественном языке: обзор публикаций // Искусственный интеллект и принятие решений. 2021. № 3. С. 66–86.