

Сравнительный анализ использования открытых моделей нейронных сетей в тестировании программного обеспечения

Латанов К.В., СПбГУ, Санкт-Петербург st134091@student.spbu.ru

Аннотация

В статье рассматривается использование открытых больших языковых моделей (LLM) нейронных сетей и оценивается эффективность их применения в тестировании REST API. Решена задача создания программного кода для автоматизированных проверок тестовых сценариев и проведен сравнительный анализ эффективности выполнения поставленной задачи на основании показателей выбранных метрик.

Введение

В процессе разработки ПО обычно выделяют несколько этапов [1], таких как планирование, разработка, тестирование и др. На практике возникает необходимость эффективного выполнения каждого из этих этапов. В данной статье речь пойдет об этапе тестирования, проводящемся при помощи автоматизации, в том числе с использованием средств искусственного интеллекта. Это позволяет уменьшить трудозатраты на выполнение рутинных задач по ручному тестированию программного обеспечения [2]. Ими, например, являются составление чек-листов, их прохождение с целью выявления дефектов объекта, адаптация существующих кейсов и разработка новых в случае изменения функционала.

Постановка задачи

При тестировании REST API возникает задача проверки работоспособности API-методов, поскольку именно они обеспечивают взаимодействие между компонентами системы, а также обмен данными между конечным пользователем и сервером. Проверка работоспособности API-методов начинается с подготовки чек-листа, состоящего из небольших кейсов и основанного на описанных в документации требованиях к рассматриваемому API. Сами кейсы должны быть атомарными и содержать следующие проверки:

- соответствие требованиям документации;

- корректность получаемых данных: код статуса, заголовок, тело ответа и другое;
- граничные значения;
- обработка невалидного ввода;
- отсутствие неожиданных артефактов и ошибок (например, 5xx) при открытии страницы.

Объектом тестирования будет являться публичное API *Petstore*, содержащее запросы разных типов: GET, PUT, POST и DELETE. Генерация автотестов будет осуществляться для раздела API под названием *Pet* [3].

Критериями сравнительного анализа будем считать:

1. Среднее покрытие эндпоинтов раздела *Pet* – среднее арифметическое отношений количества протестированных статус-кодов одного эндпоинта к количеству его возможных статус-кодов;
2. Общее покрытие статус-кодов раздела *Pet* – отношение количества всех протестированных статус-кодов раздела *Pet* к количеству всех возможных (неуникальных) статус-кодов раздела *Pet*;
3. Общее покрытие эндпоинтов API – отношение полностью протестированных эндпоинтов (у которых были протестированы все заявленные статус-коды) к количеству всех эндпоинтов API;
4. Общее покрытие статус-кодов API – отношение всех протестированных статус-кодов всего API к количеству всех возможных (неуникальных) статус-кодов API.

Цель работы – опираясь на подход, описанный в исследовании [4], оценить и сравнить эффективность применения современных языковых моделей для разработки тестовых сценариев. Будем генерировать тестовые сценарии при помощи больших языковых моделей и оценивать качество результатов. Нейронные сети получают заготовленный промпт с описанием тестовых сценариев, критериями приёма и аспектами тестирования [5].

Ожидаемым результатом являются скрипты на языке Python, использующие библиотеки *pytest* и *requests* и содержащие автоматизированные тесты.

Сценариями тестирования будем считать тесты отдельно взятых модулей (unit-тесты) и глобальные сквозные проверки функционала (end-to-end-тесты). Для первой группы каждый вид рассматриваемого HTTP-запроса подвергнется тестированию основного функционала (прямое использование, по-

зитивные кейсы) и валидации отправляемых данных (неверное использование, негативные кейсы). Во втором случае целью проверок является поиск ошибок при совместном использовании каждой конечной точки в API (endpoint-a), изучение влияния endpoint-ов друг на друга, а также поиск неожиданных артефактов, противоречащих техническому заданию.

Создание автоматизированных тестовых сценариев

Был подготовлен единый промпт следующего вида:

Дано:

1. API - <https://petstore3.swagger.io/>
2. *openapi.json* этого API с описанием эндпоинтов и статус-кодов - <https://petstore3.swagger.io/api/v3/openapi.json>

Цель: Написать автоматизированные проверки на Python с использованием библиотеки *pytest* (файл *main.py*), обеспечив максимальное покрытие тестами раздела *Pet* из рассматриваемого API.

Задачи:

- провалидировать эндпоинты по имеющейся документации (получить для каждого все описанные в *openapi.json* статус-коды)
- провести один *end-to-end* кейс с использованием *CRUD*-операций раздела *Pet* (если есть несколько запросов одинакового типа, то последовательно вызвать каждый)
- добавить несколько тестов на негативные проверки (невалидные данные в *body/header* и т.д.)
- общее количество проверок должно быть не меньше 10
- обеспечить интеграцию создаваемого файла *main.py* (генерируемый) и готового файла *metrics.py* (задан ниже) для анализа покрытия API сгенерированными тестами. Результаты метрик должны выводиться после выполнения тестов в консоль в том же порядке, что в документации (сначала раздел *Pet*, затем все остальные).

С полным текстом файла *metrics.py* можно ознакомиться в [5].

Тестировались такие нейросети, как Chat GPT-4o, Deepseek (базовый и R1), Gemini (2.0 Flash и 2.5 Pro), Grok 3 и Qwen2.5-Max. Результат сравнительного анализа работы нейронных сетей с точки зрения приведённых метрик охарактеризован в таблицах 1 и 2.

	Deepseek	Gemini 2.5 Pro	Grok 3
Кол-во итераций до успеха	3	4	3
Кол-во сгенерированных тестов	10	25	24
Процент верно выполненных	90.0%	84.0%	91.7%
Среднее покрытие эндпоинтов Pet	71.9%	75.0%	86.5%
Покрытие статус-кодов Pet	71.4%	71.4%	85.7%
Общее покрытие эндпоинтов API	15.8%	21.1%	26.3%
Общее покрытие статус-кодов API	33.3%	33.3%	40.0%

Таблица 1: Сравнительный анализ результатов работы нейронных сетей (успешное выполнение).

	ChatGPT-4o	Gemini 2.0 Flash	Qwen2.5-Max	DeepThink (R1)
Кол-во итераций до успеха	3	5	5	2
Кол-во сгенерированных тестов	12	13	16	15
Процент верно выполненных	100.0%	46.2%	100.0%	66.7%
Среднее покрытие эндпоинтов Pet	38.1%	36.5%	33.3%	44.8%
Покрытие статус-кодов Pet	—	33.3%	33.3%	42.9%
Общее покрытие эндпоинтов API	0.0%	10.5%	5.3%	10.5%
Общее покрытие статус-кодов API	—	15.6%	15.6%	20.0%

Таблица 2: Сравнительный анализ результатов работы нейронных сетей (выполнено с ошибками).

Итоги работы каждой из нейронных сетей:

- **Grok 3:** Дал хорошее первоначальное покрытие (~50% для статус-кодов и эндпоинтов) и качественные тесты (~85%, выполнилось успешно 11/13).

На второй итерации значительно увеличил количество тестов (почти в

два раза), тем самым повысив покрытие до 86%. Хорошо структурировал тесты по группам и методам – e2e, позитивные, негативные;

- **Gemini 2.5 Pro:** На первой же итерации показал достойные результаты – среднее покрытие 75%, покрытие статус-кодов 71%. За несколько следующих итераций удалось сохранив эти показатели увеличить процент успешных тестов до 84%.

Код содержит большое количество качественных проверок, а также четко разделен по группам тестов (позитивные, негативные, e2e). Отдельно стоит отметить высокую детализацию сценариев внутри многих тестов;

- **Gemini 2.0 Flash:** Изначально маленькое покрытие (11.5% в среднем при тестировании эндпоинтов) и низкое качество тестов (23.1% успешных). Но после нескольких итераций результат удалось в несколько раз улучшить (до 53% и 34% соответственно);
- **Deepseek:** Плохо начал с точки зрения успешности тестов (40%), но в итоге выполнил поставленную задачу, пройдя минимальный порог по некоторым метрикам.

На выходе получен структурированный код с короткими комментариями, который четко выполняет заданные инструкции (атомарные проверки, e2e-сценарий, вариативные негативные тесты);

- **DeepThink (R1):** В начале выдал достаточно низкий процент успешных тестов (40%) и покрытия (30% для эндпоинтов и 28% для статус-кодов). На второй итерации несколько улучшил показатели, однако все следующие попытки значительно ухудшали результат (сократилось число проверок, покрытие по нескольким метрикам стало меньше 10%).

Код структурирован по виду проверок (позитивные, негативные и e2e), а также снабжен тезисными комментариями;

- **ChatGPT-4o:** Дал неплохое первоначальное покрытие, но неверно выполнил поставленную задачу – проверял входимость полученного статус-кода в список описанных вместо того, чтобы тестировать каждый из них. На следующих итерациях нелегитимно изменил структуру файла, из-за чего часть метрик перестала отображаться;
- **Qwen2.5-Max:** Изначально не справился с поставленной задачей, т.к. на протяжении нескольких итераций не выводил метрики. После явного

исправления этого дефекта метрики отобразились, однако за несколько итераций минимальные пороговые значения так и не были достигнуты (остались в пределах 30%).

Заключение

При том, что некоторые LLM лишь частично реализовали обозначенные требования, каждая модель смогла сгенерировать минимальный набор скриптов с автоматизированными тестами данного API. Полностью с поставленной задачей справился только Grok 3. С незначительной погрешностью, но тоже успешно справились Gemini 2.5 Pro и DeepSeek. Остальные модели не смогли исправить возникшие в ходе запусков ошибки или повысить показатели метрик до минимальных уровней.

Список литературы

- [1] Hossain M. I. Software Development Life Cycle (SDLC) Methodologies for Information Systems Project Management // International Journal For Multidisciplinary Research. 2023. №5. <https://www.ijfmr.com/papers/2023/5/6223.pdf>
- [2] Khankhoje R. An intelligent API-testing: unleashing the power of AI // The International Journal of Software Engineering & Applications (IJSEA). - 2024. - №1. <https://aircconline.com/ijsea/V15N1/15124ijsea01.pdf>
- [3] Swagger Petstore // swagger.io (дата обращения: 28.04.2025). <https://petstore.swagger.io/>
- [4] Pereira A., Lima B., Faria J.P. APITestGenie: Automated API Test Generation through Generative AI (дата обращения: 09.04.2025). <https://arxiv.org/abs/2409.03838>
- [5] Репозиторий статьи // GitHub (дата обращения: 28.04.2025). https://github.com/kir64/math-mech-2025_publication/tree/main
- [6] DeepSeek Chat // deepSeek.com (дата обращения: 28.04.2025). <https://chat.deepseek.com/>

- [7] ChatGPT chat // chatGPT.com (дата обращения: 28.04.2025).
<https://chatgpt.com/?ref=dotcom>
- [8] DeepSeek-R1-Distill-Qwen-7B // huggingFace.com (дата обращения: 28.04.2025).
<https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B>
- [9] Grok chat // grok.com (дата обращения: 28.04.2025).
<https://grok.com/?referrer=website>
- [10] Gemini Chat // gemini.google.com (дата обращения: 28.04.2025).
<https://gemini.google.com/app>
- [11] Qwen3-235B-A22B Chat // chat.qwen.ai (дата обращения: 28.04.2025).
<https://chat.qwen.ai/>