DOI 10.37539/2949-1991.2026.36.1.009

Толстых Роман Андреевич, Магистрант,
АНО ВО “Российский новый университет”
Tolstykh Roman Andreevich, Master's Student,
ANO VO "Russian New University"

ГИБРИДНЫЙ МЕТОД ГЕНЕРАЦИИ ФРОНТЕНД-КОДА С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ: ДЕТЕРМИНИРОВАННЫЙ СИНТЕЗ ШАБЛОННОГО КОДА НА ОСНОВЕ СГЕНЕРИРОВАННОЙ СТРУКТУРИРОВАННОЙ СПЕЦИФИКАЦИИ
HYBRID LLM-BASED APPROACH TO FRONTEND CODE GENERATION: DETERMINISTIC TEMPLATE CODE SYNTHESIS USING LLM-GENERATED STRUCTURED SPECIFICATION

Аннотация. В настоящей работе рассматривается проблема надежности генерации программного кода большими языковыми моделями. Предлагается гибридный метод, при котором генерируется промежуточная спецификация, которая затем валидируется Pydantic-моделями и преобразуется в код шаблонизатором Jinja2. Проводится сравнительный анализ решения со сквозной end-to-end генерацией.

Abstract. This work addresses the reliability of program code generation by large language models. The article describes the implementation of a hybrid LLM-based approach which involves the generation of a structured intermediate specification and its usage in deterministic generation of Jinja2 templated code. The paper presents a comparative analysis of the hybrid method against standard LLM end-to-end generation.

Ключевые слова: Генерация программного кода, большие языковые модели, шаблонный код, гибридная генерация, шаблонная генерация.

Keywords: Program code generation, llm, template code, hybrid generation, template generation.

Введение

В настоящее время, ввиду интенсивного развития технологий искусственного интеллекта, в частности технологий генерации программного кода, современная индустрия разработки ПО переживает значительную трансформацию. Такие инструменты, как Cursor или GitHub Copilot на основе текстового описания на естественном языке позволяют генерировать синтаксически и логически корректные фрагменты кода и даже целые программные модули [1, 2]. Однако по мере интеграции LLM в критически важные процессы и системы выявляются ограничения и проблемы, свойственные вероятностному подходу. К таким ограничениям можно отнести галлюцинации, неточное следование инструкциям, дрейф контекста и генерацию небезопасных паттернов [3, 4].

В ответ на эти вызовы разрабатывается гибридная парадигма [5], в которой LLM выступает в роли семантического переводчика, преобразуя намерения пользователя в формализованную спецификацию. Затем эта спецификация проверяется и преобразуется в код с помощью детерминированных шаблонов. В настоящей работе описывается гибридная методология генерации фронтенд-компонентов, основанная на архитектуре LLM Спецификация → Шаблон → Код, и одновременно предлагается оценка этого подхода.

Сквозная генерация кода (E2E) основана на авторегрессионном предсказании токенов. Модели создают наиболее вероятные синтаксически точные структуры [6]. Но если рассматривать, как этот механизм реализуется на практике, возникает ряд серьезных рисков.

Например, выявляются случаи «пакетных галлюцинаций», при которых добавляются импорты библиотек, которых не существует, что открывает путь для атак через цепочку поставок [4]; «ограниченное контекстное окно», когда при рефакторинге модель обновляет функцию, но забывает обновить вызовы [7]; «низкая воспроизводимость», которая означает, что одни и те же запросы генерируют разный код [8]. Попытки решить эти проблемы путем доработки исходных запросов и инструкций не дают устойчивого результата, поскольку не существует формального контракта между намерением и структурой кода.

Метод

В данной работе предлагается гибридный метод, который является ответом на проблему стохастичности и реализуется за счет введения жесткого промежуточного контракта и разделения конвейера на три этапа, каждый из которых имеет четко определенные зоны ответственности.

Первым этапом процесса является семантический разбор запроса пользователя при помощи LLM. На этом шаге LLM анализирует входящие данные, которые представлены текстовым запросом пользователя, контекстом проектной документации, извлекаемой из векторного хранилища данных, и информацией о структуре целевого шаблона кода. В результате работы LLM генерирует структурированный вывод, соответствующий валидируемой JSON-схеме [9]. При таком подходе модель не ограничивается синтаксическими тонкостями целевого языка, а сосредоточена на семантике.

Следующим шагом является стадия верификации. Валидация спецификации достигается за счет использования типизированных Pydantic-моделей [10], включающих проверку типов, проверку бизнес-правил и нормализацию значений. Необходимо различать эти два уровня: JSON-схема помогает обеспечить структурную целостность выходных данных LLM, в то время как Pydantic облегчает семантическую проверку на уровне приложения. Полученная на данном этапе типизированная спецификация выступает в качестве «семантического моста» между семантическим и синтаксическим уровнями [11]. Она включает в себя идентификатор сущности, правила валидации, описание полей и действий с соответствующими типами и ограничениями, параметры пользовательского интерфейса.

Финальным этапом конвейера является детерминированная генерация кода. На этом шаге собранная на предыдущих этапах структурированная спецификация подается в процессор шаблонов Jinja2 для генерации финального программного кода [12]. Детерминированный синтез можно определить как фундаментальную характеристику, при которой результат является предопределенным и последовательным при условии, что спецификации и шаблон остаются неизменными. Достижение полной воспроизводимости конвейера зависит от спецификации, контекста поиска и настроек модели.

Эксперимент и результаты

Описанный выше гибридный подход был реализован в программном модуле на языке Python. В результате работы конвейера создается полный набор артефактов, включающий в себя валидированную спецификацию и отчет, а также готовый UI-компонент с интегрированными тестами и документацией к использованию. Основной конвейер программы включает опциональный механизм repair loop, при котором LLM анализирует ошибки, возникающие во время рендеринга шаблонов. Важно, что LLM не исправляет сам рендеринг, а дорабатывает спецификацию, на основе которой производятся дополнительные попытки рендеринга. В ходе цикла исправления сохраняются артефакты, которые обеспечивают трассируемость и воспроизводимость.

Для оценки эффективности предлагаемого подхода было проведено экспериментальное исследование. Выборка была сформирована из 15 стандартных задач фронтенд-разработки, которые были сгруппированы по трем категориям: формы с валидацией; формы с полями

выбора; таблицы с сортировкой. В качестве LLM использовалась модель GPT-5-mini с параметром температуры 0.1. В исследовании использовались следующие критерии оценки корректности: корректность структуры; воспроизводимость при неизменной спецификации; полнота артефактов; доля ошибок, которые были исправлены в ходе repair loop. Результаты проведенных тестов представлены в Таблице 1.

Таблица 1.

Результаты пилотной оценки (N=15, 3 прогона)

Метрика	**Значение**
Валидность JSON (структурированный вывод)	98% (44/45)
Успешный рендеринг без repair loop	89% (40/45)
Успешный рендеринг после repair loop	98% (44/45)
Воспроизводимость при фиксированной спецификации	100%
Полнота артефактов	100%

Структурированный вывод модели обеспечивает 98% валидного JSON, это можно объяснить хорошей обученностью моделей семейства GPT-5 на корректную генерацию структурированных данных и вызовов функций. Механизм repair loop повышает количество успешных сценариев с 89% до 98%, что подтверждает его пользу в архитектуре. Неуспешные случаи связаны с исходными требованиями, выходящими за рамки шаблона.

Результаты проведенных экспериментов и данные из литературы позволяют сделать качественное сравнение классической вероятностной E2E-генерации кода LLM и предложенного гибридного подхода. Результаты сравнения представлены в Таблице 2.

Таблица 2.

Сравнение E2E и гибридного подхода

Метрика	**E2E-подход**	**Гибридный подход**
Контроль структуры	Вероятностный	Детерминированный
Воспроизводимость	Низкая	Высокая
Риски процесса поставки	Высокие	Локализованные
Сопровождаемость	Низкая	Высокая

Заключение

Таким образом, можно сделать вывод, что гибридный подход обеспечивает лучшую трассируемость и воспроизводимость результата, что является критичным при разработке ПО корпоративного или критически значимого уровня. Тем не менее, гибридная генерация имеет ряд ограничений, которые напрямую влияют на масштабируемость решения. Подготовка и валидация исходных шаблонов является довольно трудоемким процессом, который требует наличия компетенций в ручном программировании и хорошей базы проектной документации. В этом ключе дальнейшие направления исследований в области гибридной генерации кода будут включать исследование возможностей автоматизации подготовки шаблонов рендеринга кода, в том числе с использованием LLM.

Список литературы:

1. Chen M., Tworek J., Jun H. и др. Evaluating Large Language Models Trained on Code // arXiv:2107.03374 [cs.CL]. – 2021. – (Preprint). URL: https://arxiv.org/abs/2107.03374 (дата обращения: 19.01.2026).

2. Vaithilingam P., Zhang T., Glassman E. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models // In CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI `22 Extended Abstracts); 29 апреля – 5 мая 2022 г.; Нов-Орлеан (США). – Нью-Йорк: ACM, 2022. – С. 1-7. DOI: 10.1145/3491101.3519665.

3. Pearce H., Ahmad B., Tan B., Dolan-Gavitt B., Karri R. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions // 2022 IEEE Symposium on Security and Privacy. – 2022. – С. 754-768. DOI: 10.48550/arXiv.2108.09293.

4. Spracklen J., Wijewickrama R., Sakib A. H. M. N., Maiti A., Viswanath B., Jadliwala M. We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs // arXiv:2406.10279 [cs.SE]. – 2024. – (Preprint). URL: https://arxiv.org/abs/2406.10279 (дата обращения: 19.01.2026).

5. Chaudhuri S., Ellis K., Polozov O., Singh R., Solar-Lezama A., Yue Y. Neurosymbolic Programming // Foundations and Trends in Programming Languages. – 2021. – Vol. 7, № 3. – С. 158-243. DOI: 10.1561/2500000049.

6. Xu F. F., Alon U., Neubig G., Hellendoorn V. J. A Systematic Evaluation of Large Language Models of Code // Proc. MAPS Workshop at PLDI 2022. – Нью-Йорк: ACM, 2022. – С. 1-10. DOI: 10.1145/3520312.3534862.

7. Jimenez C. E., Yang J., Wettig A., Yao S., Pei K., Press O., Narasimhan K. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? // International Conference on Learning Representations. – 2024. – (Conference paper, ICLR 2024). URL: https://arxiv.org/abs/2310.06770 (дата обращения: 19.01.2026).

8. Ouyang S., Zhang J. M., Harman M., Wang M. An Empirical Study of the Non-determinism of ChatGPT in Code Generation // arXiv:2308.02828 [cs.SE]. – 2023. – (Preprint). URL: https://arxiv.org/abs/2308.02828 (дата обращения: 19.01.2026).

9. OpenAI. Structured model outputs – guide // OpenAI API Documentation. – 2024. URL: https://platform.openai.com/docs/guides/structured-outputs (дата обращения: 19.01.2026).

10. Pydantic. Pydantic Validation // Документация Pydantic. – 2024. URL: https://docs.pydantic.dev/latest/ (дата обращения: 19.01.2026).

11. Tai C. A., Nie P., Golab L., Wong A. NL in the Middle: Code Translation with LLMs and Intermediate Representations // arXiv:2507.08627 [cs.SE]. – 2025. – (Preprint). URL: https://arxiv.org/abs/2507.08627 (дата обращения: 19.01.2026).

12. Ronacher A. Jinja Documentation (3.2.x) // Pallets Projects. – 2024. URL: https://jinja.palletsprojects.com/en/latest/ (дата обращения: 19.01.2026).