ОТ РУЧНОЙ РАЗРАБОТКИ К ОРКЕСТРАЦИИ ИИ-ИНСТРУМЕНТОВ: ПОСТРОЕНИЕ ВОСПРОИЗВОДИМОГО ПАЙПЛАЙНА ФРОНТЕНД-РАЗРАБОТКИ НА ПРИМЕРЕ КАСТОМНОЙ ВЕБ-КАРТЫ

ГРИГОРЕНКО Марк Александрович

студент

Научный руководитель: **ЛЫСЕНКО Алексей Федорович**

кандидат технических наук, доцент

директор Института опережающих технологий «Школа Икс»

Донской государственный технический университет

г. Ростов-на-Дону, Россия

Статья посвящена переходу от разовых обращений к языковым моделям к системной оркестрации ИИ-инструментов во фронтенд-разработке. На примере кастомной веб-карты показано, что устойчивый результат достигается не отдельной генерацией кода, а воспроизводимым процессом, включающим контекст, проектные правила, планирование, выбор модели под тип задачи и обязательную валидацию результата.

Ключевые слова фронтенд-разработка; большие языковые модели; ИИ-инструменты; агентные IDE; оркестрация; веб-карта

Быстрое распространение больших языковых моделей заметно изменило повседневную практику программирования. Если на раннем этапе ИИ воспринимался прежде всего как средство генерации отдельных фрагментов кода, то сегодня он все чаще встраивается в IDE и начинает участвовать в многошаговом цикле разработки [1; 5; 7]. Поэтому значимым становится уже не только качество отдельного ответа модели, но и то, как именно организовано взаимодействие с ней в рамках инженерного процесса.

Для фронтенд-разработки этот вопрос имеет особое значение. Интерфейсный модуль почти всегда создается на пересечении нескольких ограничений: архитектуры приложения, производительности, пользовательских сценариев, адаптивности и согласованности состояний. В таких условиях даже формально корректный код может оказаться малоценным, если он не встроен в структуру проекта, его внутренние правила и общую логику принятых решений.

В статье рассматривается кейс перехода от ручной разработки и разовых запросов в чат к оркестрации ИИ-инструментов в рамках воспроизводимого пайплайна. В качестве примера выбран сложный UI-модуль – кастомная веб-карта на основе Yandex Maps и PixiJS. Такой кейс показателен, поскольку в нем одновременно критичны интерфейсная логика, производительность, работа с плотными наборами данных и точность архитектурных решений.

Цель статьи – показать, как на основе практических проб и ошибок формируется устойчивый пайплайн ИИ-ассистированной фронтенд-разработки и какие элементы этого пайплайна оказываются решающими. Объект исследования – процесс разработки сложного UI-модуля с применением ИИ-инструментов. Предмет исследования – воспроизводимый способ взаимодействия разработчика с ИИ в задачах, где одновременно значимы архитектура, производительность и внутренняя дисциплина проекта. Методологически работа опирается на аналитическое исследование кейса и сравнительный анализ нескольких режимов взаимодействия с ИИ [3; 4].

1. Ограничения чатового режима

На раннем этапе ИИ использовался как внешний помощник: ему передавался небольшой фрагмент кода или краткое описание задачи, после чего ответ вручную переносился в проект. Такой формат оказался полезным для небольших функций, объяснения чужого кода, черновой верстки и рутинных правок. Однако почти сразу стала заметна и его скрытая цена: значительная часть времени уходила на механическое копирование и вставку результатов. Подобная двойственность хорошо согласуется с эмпирическими работами, в которых ИИ-

инструменты одновременно связываются с ускорением отдельных задач и с новыми ограничениями практического использования [2; 6].

Пока задача оставалась локальной, такие издержки были терпимы. Но если изменение затрагивало разветвленную систему компонентов, приходилось вручную передавать в чат несколько связанных файлов, объяснять структуру проекта или даже отправлять дерево директорий, а затем докладывать недостающий контекст. В результате заметная доля усилия тратилась уже не на решение инженерной задачи, а на обслуживание самого канала взаимодействия с инструментом.

На сложном модуле эти ограничения становятся критичными. На примере кастомной веб-карты особенно ясно видно, что разовые запросы перестают работать устойчиво. Во-первых, модель видит только тот контекст, который ей передан вручную, и поэтому теряет архитектурную рамку решения. Во-вторых, отдельные ответы слабо согласуются друг с другом: локально удачные фрагменты кода при интеграции образуют хрупкую и трудно поддерживаемую систему. В-третьих, резко возрастает объем ручной адаптации результата.

Для данного проекта особенно важной оказалась еще одна проблема: без постоянного напоминания ИИ предлагал решения, исходя из собственных умолчаний, а не из внутренних стандартов кодовой базы. Например, модель могла игнорировать наличие авторской библиотеки компонентов, не учитывать работу по Feature-Sliced Design, правила выноса констант, разделение логики и визуальных компонентов, а также принятую обвязку для взаимодействия с API. В каждом новом чате это приходилось объяснять заново. Следовательно, проблема заключалась не только в нехватке информации, но и в отсутствии механизма, который позволял бы удерживать проектную дисциплину.

Именно здесь становится ясно, что сложный фронтенд требует не просто более сильной модели, а иного режима взаимодействия с ней.

2. Переход к IDE и логике оркестрации

Следующий переломный этап связан с переходом в IDE-среду, где ИИ работает не поверх копируемых сообщений, а внутри кодовой базы [1; 5]. Для меня

это стало первым по-настоящему ощутимым преимуществом по сравнению с чатовым режимом. Инструмент перестал зависеть от ручной передачи контекста, начал самостоятельно находить нужные файлы, учитывать связанную логику и вносить изменения не только в один целевой фрагмент, но и в затронутые части проекта.

Существенную роль сыграла и сама среда: возможность создавать, перемещать и удалять файлы, выполнять рефакторинг внутри проекта и проверять поведение изменений на собственном сервере разработки в агентном режиме заметно сократила долю механической работы. Однако и этого оказалось недостаточно. Без правил поведения, предварительного планирования и различения задач по типам даже сильная модель может вносить слишком широкие или плохо интегрируемые изменения.

Поэтому дальнейшая эволюция практики привела к нескольким устойчивым приемам. Во-первых, к использованию кастомных правил, которые удерживают ИИ в рамках проектных соглашений. В данном проекте к ним относятся работа по Feature-Sliced Design, обязательный вынос констант, запрет на magic values и требования к JSDoc-документации. Во-вторых, к осознанному выбору модели под тип задачи. Для простых задач в моей практике чаще всего используется автоматический режим внутри IDE. Для более сложных сценариев – рефакторинга, изменения структуры данных, оптимизации высоконагруженных компонентов вроде карты – предпочтение отдается более мощным рассуждающим моделям, например моделям семейства Opus. В-третьих, к использованию режима планирования перед внесением сложных изменений.

Для меня переломным моментом стало именно то, что plan mode берет на себя скелет крупных изменений и превращает их в явный список шагов [5]. Благодаря этому модель в меньшей степени склонна хаотично расширять область вмешательства и всегда имеет опору: что уже сделано, что еще осталось и какова исходная цель. Кроме того, план можно сначала прочитать и скорректировать, а

уже затем разрешить переход к реализации. Это повышает управляемость и снижает цену ошибки.

Важно подчеркнуть, что эволюционировал не только набор инструментов, но и сам способ инженерного мышления. На раннем этапе единицей работы был отдельный ответ модели. На более зрелом этапе единицей работы становится уже не ответ, а связанный процесс: постановка задачи, подготовка контекста, выбор режима, согласование плана, выполнение изменений и проверка результата.

3. Воспроизводимый пайплайн

В результате серии проб и ошибок сформировался следующий рабочий пайплайн. Сначала формулируется задача и определяется ее тип: требуется ли архитектурное решение, рефакторинг, локальная правка, верстка по макету или анализ существующего кода. Затем выбирается среда работы: чатовый режим подходит для небольших изолированных задач, тогда как сложные изменения требуют IDE, где ИИ способен видеть проект целиком.

Следующий шаг – сбор релевантного контекста. В случае кастомной вебкарты к нему относятся структура данных объектов, ограничения Yandex Maps, способ интеграции PixiJS, требования к скорости отрисовки, пользовательские сценарии и принятые соглашения по организации кода. После этого выбирается модель или режим взаимодействия под тип задачи. Далее строится план, а затем выполняются изменения.

После реализации обязательна локальная валидация. Ключевым инструментом здесь становится просмотр git-изменений: по ним оцениваются логика вмешательства, отсутствие лишнего или нестандартного кода, корректность границ затронутых файлов, а также то, не были ли изменены участки, которые ИИ вообще не должен был трогать. Дополнительный слой проверки связан с безопасностью и дисциплиной интеграции: важно не закоммитить чувствительные данные и убедиться, что логика распределена по файлам корректно. Принципиально важно, что устойчивость такого процесса обеспечивается не одной сильной моделью, а дисциплиной последовательности.

4. Ключевые опоры качества

Практика показала, что качество ИИ-ассистированной фронтенд-разработки в наибольшей степени зависит от трех факторов: контекста, правил и планирования. Контекст определяет предел понимания задачи. Если модель не знает, как устроен проект, какие сущности являются ключевыми и где находятся ограничения производительности, она почти неизбежно предложит локально правдоподобное, но системно слабое решение. Правила удерживают ИИ в рамках инженерных соглашений проекта. Планирование, в свою очередь, превращает взаимодействие с ИИ из реакции в управляемый процесс.

Не существует универсального режима, одинаково удобного для всех классов задач. Поэтому зрелая работа с ИИ включает не только инженерную постановку запросов, но и инженерную маршрутизацию задач – осознанное распределение задач по наиболее подходящим инструментам и сценариям взаимодействия.

5. Изменение роли разработчика

Именно здесь особенно заметно, как меняется роль разработчика. Он все меньше выступает как исполнитель, вручную реализующий каждый элемент решения, и все больше – как руководитель производственной системы, который выстраивает процессы, настраивает автоматизацию и следит за тем, чтобы вся система работала согласованно. Доля ручного написания отдельных фрагментов кода может сокращаться, но одновременно возрастает значимость постановки задачи, выбора среды, управления контекстом, проверки архитектурной состоятельности результата и оценки того, как локальное изменение повлияет на систему в целом.

С этой точки зрения ИИ не устраняет инженерную работу, а переносит ее на более высокий уровень абстракции. При правильной организации процесса он ускоряет рутину и высвобождает ресурс для более ценных инженерных действий.

Рассмотренный кейс позволяет сделать несколько выводов. В сложной фронтенд-разработке переход от ручного кодирования к ИИ не сводится к механической замене одних действий другими. Речь идет о смене логики работы: от точечных обращений к модели к управлению целостным процессом.

Основной прирост практической ценности возникает не в момент получения удачного ответа, а в момент построения воспроизводимого пайплайна, который включает контекст, правила, планирование, выбор инструмента под задачу и обязательную валидацию результата. Сложный UI-модуль, такой как кастомная веб-карта, особенно ясно демонстрирует пределы разовой генерации и сильные стороны оркестрации.

Таким образом, переход от ручной разработки к оркестрации ИИ-инструментов следует понимать как перераспределение инженерного усилия. Чем точнее разработчик формулирует задачу, управляет контекстом и контролирует цепочку изменений, тем выше отдача от ИИ в реальном проекте. Современный разработчик все в большей степени оркестрирует процесс, а не только пишет код вручную.

СПИСОК ЛИТЕРАТУРЫ

1. Choi, D. Run long horizon tasks with Codex [Электронный ресурс] / D. Choi // OpenAI Developers Blog. – 2026. – URL: https://developers.openai.com/blog/run-long-horizon-tasks-with-codex (дата обращения: 11.04.2026).

2. Peng, S. The impact of AI on developer productivity: Evidence from GitHub Copilot [Электронный ресурс] / S. Peng, E. Kalliamvakou, P. Cihon, M. Demirer. – arXiv, 2023. – URL: https://arxiv.org/abs/2302.06590 (дата обращения: 11.04.2026).

3. Runeson, P. Guidelines for conducting and reporting case study research in software engineering / P. Runeson, M. Höst // Empirical Software Engineering. – 2009. – Vol. 14. – P. 131–164.

4. Runeson, P. Case study research in software engineering: Guidelines and examples / P. Runeson, M. Höst, A. Rainer, B. Regnell. – Hoboken : Wiley, 2012. – 237 p.

5. Smith, J. Introducing Plan Mode [Электронный ресурс] / J. Smith // Cursor. – 2025. – URL: https://cursor.com/blog/plan-mode (дата обращения: 11.04.2026).

6. Vaithilingam, P. Expectation vs. experience: evaluating the usability of code generation tools powered by large language models / P. Vaithilingam, T. Zhang, E. L. Glassman // CHI Conference on Human Factors in Computing Systems Extended Abstracts. – New York : Association for Computing Machinery, 2022. – Article 332.

7. Zhang, Q. A survey on large language models for software engineering / Q. Zhang, C. Fang, Y. Xie [и др.] // Science China Information Sciences. – 2026. – Vol. 69. – Article 190101.

FROM MANUAL DEVELOPMENT TO AI TOOL ORCHESTRATION: BUILDING A REPRODUCIBLE FRONTEND DEVELOPMENT PIPELINE USING A CUSTOM WEB MAP AS A CASE STUDY

GRIGORENKO Mark Aleksandrovich
Student
Scientific supervisor: **LYSENKO Aleksei Fedorovich**
Candidate of Technical Sciences, Associate Professor
Director of the X School Institute of Advanced Technologies
Don State Technical University
Rostov-on-Don, Russia

The article examines the transition from isolated use of language models to systematic orchestration of AI tools in frontend development. Using the case of a custom web map, it shows that stable results are achieved not through single code generations

but through a reproducible process that includes context, project rules, planning, task-based model selection, and mandatory validation of the result.

Keywords frontend development; large language models; AI tools; agent IDEs; orchestration; web map