



ИИ без серверов: запуск моделей прямо в браузере

В современном веб-разработке появляются инфраструктуры, позволяющие интегрировать ИИ непосредственно в браузер пользователя без собственной серверной обработки. Это значит, что модели машинного обучения выполняются на стороне клиента (на устройстве пользователя) с помощью возможностей браузера – без обращения к серверу разработчика. Такой подход даёт ряд преимуществ: повышение приватности (данные пользователя не покидают устройство), снижение задержек ответа и экономию на серверных мощностях. Ниже рассмотрим текущие технологии для веб-ИИ без серверов, примеры их использования в SaaS-продуктах, перспективную концепцию оркестрации множества небольших моделей (SLM) и сравним преимущества этого подхода с использованием крупных моделей на удалённых серверах (OpenAI GPT-4, Anthropic Claude, Google Gemini и др.).

Текущие возможности: ИИ в браузере без сервера

Встроенные возможности браузеров. Современные браузеры начинают получать встроенные API для работы с ИИ. Например, в Chrome 2024–2025 годов появились экспериментальные API для перевода текста, определения языка, автодополнения и суммаризации, которые позволяют запускать соответствующие модели локально, прямо в браузере ¹. Эти встроенные AI API предоставляют единый простой интерфейс и самостоятельно загружают необходимые модели (при первом использовании браузер может определить, что модели нет в наличии, и автоматически её скачать) ². Благодаря этому разработчики могут добавить функции машинного перевода, проверки грамматики или сводного пересказа текста на сайт, не отправляя данные пользователя на внешний сервер.

JavaScript-библиотеки и WebGPU. Помимо встроенных API, существует множество библиотек для запуска моделей на стороне клиента. Например, **TensorFlow.js**, **ONNX Runtime Web**, **Transformers.js** и др. позволяют загружать предобученные модели в память браузера и выполнять их с аппаратным ускорением через WebGL или WebAssembly ³ ⁴. Новейший стандарт **WebGPU** (поддерживается в актуальных версиях Chrome, Safari, Edge) даёт ещё больший прирост производительности, позволяя запускать даже более тяжёлые нейросетевые модели напрямую на GPU пользователя ⁵ ⁶. К примеру, библиотека **WebLLM** демонстрирует, что большие языковые модели вроде Llama2-7B или Mistral-7B могут выполняться в браузере благодаря WebGPU, без серверной обработки ⁷. Иначе говоря, можно открыть веб-страницу – и она загрузит в фоновой части необходимые веса модели, после чего ИИ начнёт работать практически как ChatGPT, но локально.

Примеры практического использования. Уже сегодня в веб-приложениях реализуются разнообразные AI-функции на клиентской стороне. Например, можно встроить модель классификации изображений или распознавания лиц в онлайн-редактор фото, и все вычисления будут проходить локально в браузере пользователя ⁸. Видеоконференц-платформы могут выполнять шумоподавление или размытие фона средствами WASM прямо на клиенте. Существуют демонстрации онлайн-чатботов, работающих целиком в браузере без бэкенда – пользователь получает ответы, даже если отключить интернет, поскольку модель уже загружена в память браузера. Все это достигается без установки пользователем каких-либо программ:

достаточно загрузить страницу, и веб-приложение само подтянет необходимые данные (веса модели) и выполнит их на месте. Такой подход повышает надёжность (приложение работает даже при проблемах с сетью) и защищает данные: например, текст для суммаризации или перевода не отправляется на внешний сервис, а обрабатывается локально ⁹.

Варианты использования в веб-продуктах (SaaS)

Безсерверный браузерный ИИ открывает новые возможности для SaaS-продуктов. Некоторые примеры:

- **Встроенные чат-боты и помощники для пользователей.** Веб-приложение может включать интеллектуального помощника (например, чат для поддержки клиентов или подсказок внутри интерфейса), который работает без отправки вопросов на внешний API. Такой бот может отвечать на часто задаваемые вопросы, помогать с навигацией по сервису или обучать пользователя. Поскольку обработка происходит на клиенте, конфиденциальные данные (например, переписка или документы) не уходят на сервер, что повышает доверие и упрощает соблюдение требований приватности.
- **Генерация и преобразование контента.** Многие SaaS-сервисы позволяют пользователям создавать или редактировать контент – будь то текст, код, изображения. Интегрировав локальные модели, можно реализовать функции автодополнения текста, рефакторинга кода, перевода заметок, суммаризации отчётов и т.д. прямо в браузере. Например, текстовый редактор может онлайн предлагать пользователю переформулировать предложения или переводить введённый текст на другой язык. Сервис аналитики способен выполнять локальную суммаризацию загруженного пользователем PDF-отчёта, гарантируя, что сами данные отчёта не покидают браузер. Подобные узкоспециализированные задачи (перевод, резюмирование текста, парсинг писем и таблиц) хорошо решаются небольшими языковыми моделями ¹⁰, которые можно встроить во фронтенд.
- **Персонализированные функции на стороне клиента.** Поскольку модель работает у пользователя, её можно адаптировать под конкретного человека без влияния на других. К примеру, почтовый веб-клиент может обучить небольшую модель на письмах самого пользователя для приоритизации важных сообщений или генерации черновиков ответов – и всё это хранится и выполняется локально, учитывая стиль конкретного человека. В корпоративных приложениях локальный ИИ может обрабатывать внутренние данные (таблицы, документы) прямо в браузере сотрудника, соблюдая политики безопасности (данные никогда не отправляются во внешний облачный сервис).
- **AI-ассистент для команды разработки.** Разработчики могут интегрировать AI-агента в процесс разработки и DevOps, чтобы автоматизировать рутинные задачи. Такой агент, работающий, например, как расширение в IDE или внутри облачного репозитория, способен генерировать типовой код, подсказывать исправления ошибок, писать тесты и документацию – по сути, быть «соразработчиком». При этом, используя локальные модели, компания может хранить весь контекст проекта (исходный код, описание задач) прямо в среде разработчика. Это исключает утечку интеллектуальной собственности к сторонним API и снижает задержки при запросах. Интеграция множества небольших специализированных моделей особенно выгодна: одна модель может быть обучена именно для генерации кода на нужном языке, другая – для объяснения ошибок, третья – для генерации SQL-запросов и т.д. Центральный оркестратор будет направлять запрос

разработчика к нужной «микро-модели», учитывая контекст текущего проекта. Такой **полноценный ассистент**, охватывающий многие аспекты разработки продукта – от кода до инфраструктуры – проще реализовать на основе нескольких компактных моделей, чем пытаться обучить одну огромную модель на весь спектр задач.

Оркестрация небольших моделей (SLM) и будущее архитектуры ИИ

Small Language Models vs большие LLM. Термин «*малые языковые модели*» (Small Language Models, SLM) относится не столько к абсолютному числу параметров, сколько к практической *развёртыываемости* модели – SLM можно эффективно запускать на обычных устройствах (ПК, смартфонах) с низкой задержкой ¹¹. Другими словами, «*маленькая*» модель – это та, что **реально работает без специализированного оборудования дата-центра**. Благодаря росту производительности железа граница между «*малой*» и «*большой*» моделью постоянно смещается: то, что сегодня считается тяжеловесным LLM, завтра может быть по силам ноутбуку или даже мобильному устройству ¹². Например, модели, которые мы сейчас считаем «*гигантскими*», через пару лет станут доступны на потребительских GPU или в браузере. Исследователи из NVIDIA выдвигают тезис, что дальнейший прогресс агентных систем может быть достигнут не за счёт наращивания размеров моделей, а благодаря их уменьшению и специализации. В недавней работе **«Small Language Models are the Future of Agentic AI»** отмечается, что: (1) SLM уже могут выполнять большинство задач, требуемых от AI-агентов, (2) они лучше вписываются в модульные системы и (3) их эксплуатация намного более экономична ¹³. Иными словами, текущая одержимость масштабом может тормозить инновации, тогда как переход к множеству маленьких моделей откроет путь к более эффективным автономным агентам.

Идея оркестрации моделей. Вместо использования одной универсальной модели на все случаи (как GPT-4), подход SLM предлагает набор специализированных моделей, каждая из которых обучена под свою задачу, а над ними – *центральный оркестратор*, распределяющий запросы. Такая архитектура похожа на микросервисную: сложная задача разбивается на части, и «по адресу» отправляется тому модулю (модели), который лучше всего с ней справится ¹⁴. Оркестратор может поддерживать отдельный контекст диалога для каждой модели, передавая каждому «эксперту» только релевантную ему информацию. Например, в AI-ассистенте для разработки оркестратор мог бы распознавать, что вопрос касается отладки кода, и подключать модель, обученную на поиске ошибок, а вопрос о бизнес-логике – перенаправлять модели, специализирующуюся на анализе требований. При этом общая история общения хранится централизованно, но каждой узкой модели подаётся только нужный ей срез информации – так достигается и персонализация ответов, и эффективность (модели не «перегружаются» лишними данными).

Эффективность и перспективы. Уже существуют доказательства, что комбинацией SLM можно достичь качества, сопоставимого или превосходящего одну большую модель. В недавней работе SLM-MUX показано, что объединение всего двух небольших моделей способно превзойти гигантскую 72-миллиардную модель Qwen-72B на ряде задач логики и математики. Точность по некоторым тестам (MATH, GSM8K и др.) оказалась выше на ~7–13% по сравнению с результатами каждой модели по отдельности ¹⁵. Причина в том, что каждый «эксперт» (SLM) вносит свои сильные стороны, а оркестратор комбинирует их ответы. Кроме того, небольшие модели обучаются и дорабатываются значительно быстрее: добавить новую способность или откорректировать поведение SLM можно за считанные часы обучения, тогда как гигантские LLM требуют для таких изменений дней или недель работы GPU ¹⁶. Специализированные SLM также

более надёжны в узкой области – они склонны строго придерживаться требуемого формата ответов и меньше «галлюцинируют» вне своей компетенции ¹⁷. Это критично для прикладных агентных сценариев, где важно, чтобы каждый инструмент (модель) возвращал предсказуемый результат (например, строго валидный JSON или корректный SQL-запрос). В итоге система из множества маленьких моделей может быть более устойчивой: если один компонент даёт сбой или устаревает, его проще заменить или улучшить локально, не переобучая всю систему целиком.

В перспективе архитектуры с оркестрацией SLM обещают лучшую масштабируемость и экономию. Большие модели не исчезнут совсем – они по-прежнему нужны там, где требуется действительно универсальное понимание или обширные знания, выходящие за рамки узкой задачи ¹⁸. Однако их роль может сместиться в разряд **«экспертов по вызову»**: крупный LLM будет подключаться эпизодически для самых нетривиальных или междисциплинарных запросов, тогда как львиную долю повседневной работы возьмут на себя небольшие модели ¹⁹. Такой гибридный подход уже просматривается в индустрии. К примеру, платформа **NVIDIA ChatRTX** демонстрирует, как SLM могут работать локально на GPU конечного пользователя, реализуя персонального чат-бота, который полностью онлайн обрабатывает пользовательские данные – приватно и с минимальными задержками ²⁰. В целом, переход к SLM-аналогам напоминает давний сдвиг от монолитных приложений к микросервисам: вместо единого громоздкого «интеллекта», выполняющего всё, мы получаем модульную систему небольших компонентов, каждый из которых выполняет свою функцию максимально эффективно ²¹. Для разработчиков продуктов это означает больше гибкости и контроля: можно комбинировать открытые модели, обученные на данных своей отрасли, и тем самым быстрее внедрять безопасные и недорогие интеллектуальные функции в приложения.

Преимущества подхода перед большими LLM на сервере

Использование локальных моделей в браузере вместо обращения к внешним API большим языковым моделям даёт продуктовой команде ряд существенных преимуществ:

- **Экономия затрат и лучшая масштабируемость.** Малые модели обходятся значительно дешевле в эксплуатации. Например, по оценкам NVIDIA, запуск модели Llama 3B может быть в 10-30 раз дешевле, чем выполнение запроса к её высокопроизводительному «старшему брату» Llama 3.3 (405B параметров) при решении аналогичной задачи ²². Причина – меньшие требования к вычислительным ресурсам: SLM не нуждается в дорогом облачном GPU-сервере, её может выполнить браузер пользователя. Когда тысячи пользователей одновременно используют AI-функции, подход «на устройстве» масштабируется практически бесплатно (нагрузка распределяется по их устройствам, а не ложится на ваш бэкенд). Разработчику не нужно содержать инфраструктуру для LLM или платить за каждый запрос внешнему провайдеру – достаточно однажды встроить модель в приложение.
- **Приватность и контроль над данными.** Все данные обрабатываются на стороне клиента, поэтому чувствительная информация (персональные данные, внутренние бизнес-документы и т.п.) не покидает устройство пользователя. Это упрощает соблюдение требований GDPR и внутренних политик безопасности. Даже при использовании облачных API от сторонних провайдеров часто возникают вопросы доверия – при локальной же обработке данные по определению не передаются третьим лицам. Более того, приложение может работать **оффлайн**: пользователи получают ответы ИИ даже без интернета (например, в пути), поскольку модель уже загружена и не зависит от удалённого

сервиса ⁹. Устранение сетевых запросов также снижает суммарную задержку отклика и исключает простой при проблемах с подключением.

- **Высокая скорость отклика.** Выполнение запроса локально устраниет сетевые задержки. Взаимодействие с моделью происходит практически в реальном времени, особенно если модель оптимизирована и умеет задействовать GPU устройства. Это улучшает UX: AI-ассистент откликается мгновенно, тогда как запрос к удалённому API мог бы тратить сотни миллисекунд или секунды на передачу данных и ожидание ответа сервера.
- **Тонкая настройка и специализация под задачу.** Небольшие модели проще адаптировать под ваши конкретные кейсы. Их можно дообучить на специализированных датасетах компании или отрасли за приемлемое время и без астрономических расходов ¹⁶. Такой уровень кастомизации с проприетарными крупными моделями (через API) зачастую недостижим или очень дорог. Используя ансамбль SLM, каждая модель может быть строго настроена под свою функцию (например, одна генерирует описания товаров, другая переводит их на нужный язык, третья – анализирует отзывы). Это повышает точность и управляемость: модель обучена ровно тому, что нужно, и не содержит «лишних» знаний. Практика показывает, что узконаправленные SLM менее склонны отклоняться от заданного формата и требований, чем универсальные LLM ¹⁷ – а значит, интеграция таких моделей приводит к более предсказуемому поведению системы (например, строгому соблюдению требуемого формата ответов).
- **Независимость и инновации.** Подход с открытыми малыми моделями снижает зависимость от крупных вендоров ИИ. Вы не рискуете внезапным изменением цен или политики доступа к внешнему API – вся ключевая логика работает на стороне клиента и находится под вашим контролем. Кроме того, маленькие модели и их комбинации стимулируют инновации: появление новых алгоритмов с открытым кодом сразу даёт возможность улучшить свою систему (например, заменить одну модель на более совершенную, не переписывая весь стек). Более доступные модели ведут к демократизации ИИ – больше команд и компаний могут позволить себе их использовать и дорабатывать ²³, тогда как топовые LLM долгое время были прерогативой крупных корпораций. Это похоже на ситуацию с микросервисами: вместо одной монолитной системы, развитие которой подвластно только её создателю, множество небольших компонентов могут создаваться и улучшаться разными командами параллельно.

Ограничения и компромиссы. Конечно, у подхода есть и недостатки. Качество ответов небольших моделей пока уступает лучшим LLM в самых сложных, абстрактных задачах – там, где нужна широкая эрудиция или глубокое «понимание» мира, возможно, придётся всё же обращаться к облачному ИИ (либо изредка подключать большую модель в рамках гибридной архитектуры ¹⁸). Также клиентские устройства пользователей разнятся по мощности: на слабом смартфоне крупная модель может не запуститься или работать слишком медленно. Приходится тщательно оптимизировать модели (квантовать веса, сокращать число параметров), чтобы они помещались в память браузера и не разряжали батарею. Кроме того, веб-технологии вроде WebGPU ещё относительно новы – их поддержка может быть неполной (например, в ряде браузеров включается только через флаги) ²⁴, из-за чего разработчику приходится обеспечивать резервные варианты. Наконец, распределённая система из множества моделей сложнее в разработке и отладке, чем вызов одного API. Требуется больше экспертизы в ML-опсе, чтобы грамотно организовать взаимодействие SLM и обновление каждой из них. Тем не менее, ожидается, что эти трудности будут постепенно решаться по мере развития инструментов и роста вычислительных возможностей устройств – а выгоды (приватность, скорость, низкая стоимость) делают направление «ИИ на клиенте» всё более привлекательным для индустрии.

Источники: Connected papers and blogs, including NVIDIA Technical Blog [22](#) [14](#) [18](#), LogRocket Blog [11](#) [13](#), Worldline Engineering Blog [1](#), Grid Dynamics research [9](#), and arXiv research on SLM orchestration [15](#), были использованы для сбора актуальной информации. These sources provide insight into the current capabilities of browser-based AI, use cases in SaaS, the emerging practice of orchestrating small models, and a comparison with large server-hosted LLMs.

[1](#) [2](#) No server needed: running AI in the browser

<https://blog.worldline.tech/2025/08/26/web-builtin-ai.html>

[3](#) [4](#) [5](#) [6](#) [8](#) [9](#) [24](#) Client-side AI for Cost-Effective, Secure Web Apps

<https://www.griddynamics.com/blog/client-side-ai>

[7](#) WebLLM | Home

<https://webllm.mlc.ai/>

[10](#) [11](#) [12](#) [13](#) [20](#) Small language models: Why the future of AI agents might be tiny - LogRocket Blog

<https://blog.logrocket.com/small-language-models/>

[14](#) [16](#) [17](#) [18](#) [19](#) [21](#) [22](#) [23](#) How Small Language Models Are Key to Scalable Agentic AI | NVIDIA

Technical Blog

<https://developer.nvidia.com/blog/how-small-language-models-are-key-to-scalable-agentic-ai/>

[15](#) [2510.05077] Slm-mux: Orchestrating small language models for reasoning

<https://arxiv.org/abs/2510.05077>