ChatGPT

Архитектура AI-образовательной платформы на WebGPU и Transformers.js

Рисунок: Сравнение традиционной схемы инференса (слева) и WebGPU-ориентированного (справа). В модели с WebGPU вычисления выполняются на стороне пользователя без отправки данных на сервер, что даёт низкие задержки и высокую приватность [1] [2].

Приложение спроектировано как **одностраничный веб-сайт** с минимумом серверной логики: вся ML-логика и генерация контента работают прямо в браузере. Для этого используются библиотеки ONNX Runtime Web и Transformers.js (JS), которые загружают предобученные модели и запускают их на клиенте. Например, ONNX Runtime Web «сам выбирает» между бэкендом WebGPU или WASM в зависимости от наличия поддержки браузером, обеспечивая высокую скорость инференса и приватность данных [2] [1].

- **Интерфейс (Frontend):** HTML/CSS/JS (React, Vue или VanillaJS) для пользовательского интерфейса – отображения курсов, заданий, чат-бота и прочего. Веб-приложение работает как PWA (Progressive Web App): кэширует ресурсы (HTML/JS/модели), поддерживает offline-режим (см. рисунок выше).
- **ML-пайплайн (клиент):** На клиенте загружаются ONNX-модели (из CDN или хранилища) и запускаются через Transformers.js. Ключевая опция – установка `device: 'webgpu'`, чтобы выполнять вычисления на GPU браузера [3]. При отсутствии WebGPU библиотека автоматически переходит к WASM (CPU) [2] [4]. Все веса моделей и данные остаются на стороне клиента (абсолютная приватность) [5] [1].
- **Контент и данные:** Структуры курсов/дорожные карты могут храниться в формате Markdown/JSON в публичных репозиториях GitHub или на CDN, загружаться динамически через API (GitHub, REST). Это позволяет легко масштабировать и обновлять контент (похожий подход используется на roadmap.sh). Пользовательские ответы (обратная связь, заметки) можно сохранять через облачные функции или браузерное хранилище.
- **Аутентификация/профили:** Система аутентификации (OAuth) – вход через GitHub, Google и пр. как на roadmap.sh [6]. После входа создаётся профиль ученика с настройками и историей. Подписка (например, через Stripe) открывает доступ к расширенным функциям. За серверной частью (user DB, оплата) можно «привязать» лёгкий Node.js или сервисы безсерверных функций – остальная логика остаётся фронтовой.

Машинное обучение в браузере

Приложение использует **Transformers.js + ONNX Runtime Web** для запуска моделей в браузере. Эти библиотеки позволяют импортировать предобученные модели HuggingFace (в формате ONNX) и выполнять инференс прямо на клиенте [7] [8]. Например:

```
const pipe = await pipeline('text-generation','Xenova/Qwen2-0.5B-Instruct',{
  device: await navigator.gpu ? 'webgpu' : 'cpu'  // WebGPU или WASM
});
```

Это соответствует практике: ONNX Runtime Web сам маршрутизирует вычисления либо на WebGPU, либо на WASM [2] [4]. Такая схема даёт низкие задержки (нет сетевых запросов) и защищает данные (всё остаётся в браузере) [2] [1].

Для масштабирования в браузере используют **мелкие специализированные модели (SLM)**. Вместо единого большого LLM платформа разбивает задачи на «микросервисы» в виде небольших моделей. Это согласуется с концепцией «сотрудничества лёгковесных моделей» [9]. К примеру, можно держать отдельные ONNX-модели для классификации, суммаризации, генерации ответов, оценки сложности и т.д., и вызывать их по мере необходимости. Для каждого подзадачи выбирается оптимальная модель (с учётом контекста, языка, размера) – эта «оркестровка» маршрутизирует запрос к нужной сети [9] [10]. Если нужно встраивать знания, применяется Retrieval-Augmented Generation: векторизация базы учебных материалов (через ONNX-модель-эмбеддингов) и поиск релевантных фрагментов для передачи в генеративную модель [11].

Рисунок: Рекомендуемый workflow развёртывания моделей в браузере: (1) квантование модели, (2) разбиение на чанки и кэширование (IndexedDB), (3) детектирование WebGPU и запуск на GPU, (4) fallback на WASM/CPU при необходимости. Квантизованные модели (8/4 бита) занимают в разы меньше места и ускоряют инференс [12] [13].

- **Квантизация:** модели нужно переводить в низкоразрядные форматы (INT8, INT4 и т.д.). Это критично для браузера: 4-битная модель может быть в десятки раз меньше оригинала, сохраняя точность [12] [13]. Таким образом реально запускать даже миллиардные параметры в браузере при ограниченной памяти.
- **Кэширование:** после первого запуска модельные веса (чанки) сохраняются в IndexedDB или локальном хранилище, чтобы при повторном запуске не скачивать их заново. Это ускоряет работу и позволяет офлайн-доступ (если модель закеширована).
- **Проверка WebGPU:** библиотека делает feature-detection (через `navigator.gpu`); если WebGPU включён, выполняет инференс на GPU, иначе – через WASM на CPU [2] [4]. Такой плавный «падение» обеспечивает совместимость и стабильную скорость.
- **IO Binding:** при работе с ONNX Runtime Web полезно держать данные (тензоры) на GPU и избегать лишних копий в JS. Это ускоряет обработку, особенно при сложных модельных графах.

В итоге архитектура ML-процесса выглядит так: фронтенд загружает ONNX-модель (файл *.onnx) из локального/удалённого хранилища, вызывает `InferenceSession.create(modelPath, { executionProviders: ['webgpu'] })` [14], а далее применяет пайплайн Transformers.js для токенизации и генерации. Всё это – чистый JavaScript, без серверных вызовов.

AI-репетитор и персонализация

Платформа предлагает интерактивный учебник/репетитор на основе диалога. Пользователь выбирает тему или задаёт вопрос, и система генерирует индивидуальную программу или пояснение. Здесь задействуются те же малые модели: например, модель «тутор» (Qwen-0.5B и т.п.) отвечает на вопросы, генерирует план изучения. При этом можно использовать **Retrieval-Augmented Generation**: библиотека конспектов, документации и курсов кодируется эмбеддингами, а на основе запроса выбирается релевантный контент для передачи модели [11].

AI-репетитор может динамически строить «дорожные карты» навыков: на основе интересов и целей пользователя модель генерирует перечень тем, тестов, упражнений. Это аналог функционала roadmap.sh, но не на готовых статичных курсах, а на лету подгоняется под запрос. Примерно такая идея реализуется при помощи моделей: вопрос → логика приложения

определяет нужный модуль → собирает контекст (например, ранее изученные темы) → вызывает LLM. Transformers.js здесь выступает «мостом» между моделью и приложением: он сам токенизирует запрос, загружает веса и выдаёт ответ [15]. Пользовательский чат/квиз-движок просто оборачивает вызовы к этим JS-моделям.

Важно, что все запросы к ИИ происходят быстро и приватно: нет обращения к OpenAI API или другому внешнему сервису, все данные пользователя остаются в браузере [5] [1]. Одна из целей – дать «обучение по интересам» без жёсткой схемы: AI-тренер анализирует предпочтения и адаптирует материал, генерируя пояснения и тесты на лету, а не выдаёт заранее прописанный курс.

Бэкенд, аккаунты и подписки

Хотя основная логика на клиенте, нужен и лёгкий сервер для пользователей и платежей. Типичный стек: Node.js или serverless-функции для работы с базой пользователей, авторизацией и оплатой. Системы OAuth (GitHub, Google, LinkedIn) используют готовые JS-решения – roadmap.sh, например, предлагает «Continue with GitHub/Google» [6]. Это позволяет не тратить ресурсы на пароли и безопасность.

После логина создаётся профиль ученика (напр. в MongoDB или Firebase): там хранятся прогресс, настройки, история чатов. Платная подписка (через Stripe или аналог) открывает безлимитный доступ к курсам и генерации контента. Архитектура делает упор на легковесность: основное ML — на клиенте (независимом от сервера), а сервер «смотрит» лишь за учётными записями и платёжками. Хостинг можно использовать статический (GitHub Pages, Netlify) плюс облачные функции для динамики, что обеспечивает лёгкое масштабирование.

Масштабирование и практики

Ключевой принцип – **масштабируемость по пользователям** достигается за счёт переноса вычислений на клиент. Если число учеников растёт, компании не нужно увеличивать мощность серверов для инференса: каждый браузер сам «тащит» модель. Компоненты веб-приложения могут кэшироваться CDN, и это хорошо масштабируется. Описанные выше техники (квантизация, кэш, WebGPU/wasm fallback) гарантируют плавную работу на широком наборе устройств [16] [17].

Резюмируя архитектуру: приложение — это статический JS-сайт со встроенными ML-функциями. На практике это значит: весь код доступен через GitHub (способствует открытости), библиотеки (Transformers.js, ONNX Runtime) подключаются через npm или CDN [18]. Модели хранятся как файлы рядом с фронтендом. Пользовательский браузер проверяет, поддерживает ли он WebGPU (например, Chrome 113+ на десктопе, Safari 15+ на iOS, или экспериментальный Firefox) [19], загружает квантованные веса модели и выполняет инференс локально. В случае отсутствия WebGPU происходит тихий переход на WASM-инференс [4] [2]. Такая схема делает AI-платформу **быстрой, приватной и легко масштабируемой**: нет зависимостей от собственных серверов ИИ и нет перетоков трафика при каждом запросе к модели [20] [1].

Источники: современные гайды по ML в браузере [2] [1] [5], документация по Transformers.js [8] [3] и примеры использования ONNX Runtime Web + WebGPU [7] [21] описывают указанные подходы. Дополнительно концепция SLM и оффлайн-инференса обсуждается в исследованиях и статьях [9] [22].

---

1 12 16 17 AI In Browser With WebGPU: 2025 Developer Guide
https://aicompetence.org/ai-in-browser-with-webgpu/

2 Instant JS Inference with WebGPU + ONNX | by Nikulsinh Rajput | Oct, 2025 | Medium
https://medium.com/@hadiyolworld007/instant-js-inference-with-webgpu-onnx-e12797e9f0f5

3 8 13 18 GitHub - huggingface/transformers.js: State-of-the-art Machine Learning for the web. Run Transformers directly in your browser, with no need for a server!
https://github.com/huggingface/transformers.js

4 8 WebGPU + ONNX Runtime Web Inference Guides | by Modexa | Oct, 2025 | Medium
https://medium.com/@Modexa/8-webgpu-onnx-runtime-web-inference-guides-4220cff29ad8

5 15 20 Run Your Own AI in the Browser: Build a Real-Time Small Language Model Using WebGPU | by Dr. Ernesto Lee | Oct, 2025 | Medium
https://drlee.io/run-your-own-ai-in-the-browser-build-a-real-time-small-language-model-using-webgpu-8dbaa477b295?gi=4652834b3ee6

6 AI Tutor
https://roadmap.sh/ai-roadmaps/ccsp

7 10 11 14 21 Use WebGPU + ONNX Runtime Web + Transformer.js to build RAG applications by Phi-3-mini | Microsoft Community Hub
https://techcommunity.microsoft.com/blog/educatordeveloperblog/use-webgpu--onnx-runtime-web--transformer-js-to-build-rag-applications-by-phi-3-/4190968

9 22 Small language models: Why the future of AI agents might be tiny - LogRocket Blog
https://blog.logrocket.com/small-language-models/

19 Running models on WebGPU
https://huggingface.co/docs/transformers.js/en/guides/webgpu