CSCI 578 – Software Architectures Fall 2025 Team Project

PocketLLM Portal

Team Number: 9

Descriptive Architecture (Design Decisions and Departures)

This document outlines the design decisions and departures from the initial architectural plan made during the implementation of the Pocket LLM Portal.

1. Tech Stack Refinement

Decision: Use Express.js instead of NestJS.

Rationale: While the initial requirements suggested "NestJS or Express", we opted for Express.js. For a "Pocket" application with limited scope, Express provides a lighter footprint and faster development cycle while still allowing for the required modular Controller/Service architecture. NestJS's dependency injection system, while powerful, adds unnecessary boilerplate for this specific scale.

2. Inference Engine Abstraction

Decision: Implement a Mock Inference Service for the initial prototype.

Rationale: To satisfy the "CPU-only" and "Limited Resources" constraints during the development and verification phase, we implemented a simulated inference engine (InferenceService.js). This allows us to verify the Streaming Architecture (SSE, Frontend handling, Token appending) without the heavy overhead of loading a multi-gigabyte model file (GGUF) during the initial build. The architecture is designed such that this service can be replaced with a real node-llama-cpp binding with zero changes to the Controller or Frontend.

3. Frontend Styling and UX

Decision: Adopt a Slate and Sky color theme with TailwindCSS v4.

Rationale: The initial requirement was generic, but we refined the UX to be simple and clean. We chose TailwindCSS for its utility-first approach, enabling rapid UI iteration. We specifically used the latest v4 to leverage modern CSS features and reduce build configuration overhead.

4. Docker Deployment Strategy

Decision: Single Container Deployment.

Rationale: Instead of running separate containers for Frontend (Nginx) and Backend (Node), we configured the Node.js backend to serve the static frontend build in production.

Benefits:

- Reduces resource consumption by avoiding memory and CPU overhead of an extra container.
- Simplifies deployment because there is only a single Docker image.

Trade-off: Tightly couples the deployment artifacts, but acceptable for a "Pocket" app.

5. Database Choice

Decision: Better-SQLite3 in-process over a separate SQL server.

Rationale: To strictly adhere to the "Pocket" nature and resource constraints, we used better-sqlite3. This runs in the same process as the Node.js application, eliminating the latency and resource cost of a separate database server process like PostgreSQL or MySQL. It aligns with the requirement for a lightweight, self-contained app.

6. Real-time Streaming via Server-Sent Events (SSE)

Decision: Use native SSE instead of WebSockets for token streaming.

Rationale: SSE is simpler and more appropriate for unidirectional server-to-client streaming. It has less overhead than WebSockets, which are designed for bidirectional communication. For the PocketLLM use case where tokens only flow from server to client, SSE provides better resource efficiency and easier implementation. The frontend uses the native Fetch API with ReadableStream instead of the EventSource API for more granular control over the stream lifecycle.

7. Configurable Thread Limiting

Decision: Default to 2 to 3 threads for model inference, configurable via environment variables.

Rationale: The initial architecture did not specify thread management for the quantized model. During implementation, we discovered that node-llama-cpp by default uses all available CPU cores, which could exceed resource constraints or cause system slowdowns. By limiting threads

through `LLM_THREADS` and `OMP_NUM_THREADS` environment variables, we ensure the application stays within the 4 vCPU constraint and leaves resources for other system processes.

8. Single Port Deployment (3001)

Decision: Run both frontend and backend on port 3001 in production.

Rationale: Simplifies deployment and eliminates CORS configuration. In development mode, the frontend runs on Vite's dev server on port 5173 with proxying, but in production everything runs on a single port. This departs from the typical frontend (80 or 443) and backend (3001) separation but makes the Docker deployment simpler and more resource-efficient.

9. Graceful Stream Abortion

Decision: Implement abort controller pattern for stopping inference mid-stream.

Rationale: During implementation, we added the ability for users to stop generation mid-stream using the AbortController API. This prevents wasted CPU cycles on unwanted responses and improves user experience when users realize early that the response is not what they wanted.