Pocket Brain: Offline Quantized LLM Brainstorming Under CPU-Only and $\leq$ 8GB RAM Constraints

Akbar Aman & Luke Abraham

2026

Abstract

Pocket Brain is a portable, fully offline conversational assistant intended for private brainstorming on constrained hardware. The system targets CPU-only execution and a strict memory budget of $\leq$ 8GB RAM while maintaining predictable latency. The system is designed as a personal, single-user tool that prioritizes determinism, portability, and explicit resource control over feature breadth or cloud-scale generality. Rather than training models, Pocket Brain focuses on systems engineering: orchestrating quantized LLM inference, enforcing bounded context growth, and providing structured interaction modes that increase usefulness without increasing computation.

1 Problem Statement

Modern cloud LLMs are powerful but may be unsuitable for proprietary brainstorming due to privacy, network dependency, and recurring cost. Pocket Brain addresses this by running a quantized LLM locally (CPU-only), exposing a local API for both a Web UI and CLI client, and persisting sessions to local storage (e.g., external SSD). The core technical challenge is maintaining conversational usability while enforcing bounded resource usage, especially under small RAM budgets.

Scope Clarification (v1): Pocket Brain is not intended to be a commercial product in its initial form. Version 1 (v1) is scoped explicitly as a personal, offline assistant designed for private reasoning over sensitive or proprietary information. Design decisions prioritize simplicity, transparency, and controllable resource usage over feature breadth. Product-grade polish (e.g., cross-platform installers, encryption packs, vision models) is intentionally deferred.

2 Constraints and Design Goals

Hard Constraints

- **Offline-only:** no outbound network calls are required for inference or interaction.
- **CPU-only:** the system must run without GPU acceleration.
- **Memory cap:** total runtime must stay within $\leq$8GB RAM.
- **Portable storage:** models and session artifacts should reside on local disk / SSD, enabling plug-and-play usage.

Design Goals

- **Predictable latency:** prioritize bounded and consistent response times over peak throughput.
- **Short-turn conversational workflow:** optimized for paragraph-length messages rather than long-document ingestion.
- **Mode-driven structure:** provide constrained interaction modes (e.g., Perspectives, Risks, Next Steps) to improve output quality on smaller models.
- **Simple persistence:** automatically save sessions to disk for later retrieval and auditability.

3 High-Level Architecture

Pocket Brain is organized into clients (Web UI and CLI), a local API service, a backend orchestrator that enforces policies (modes, context caps), an inference engine (quantized CPU inference), and a local persistence layer.

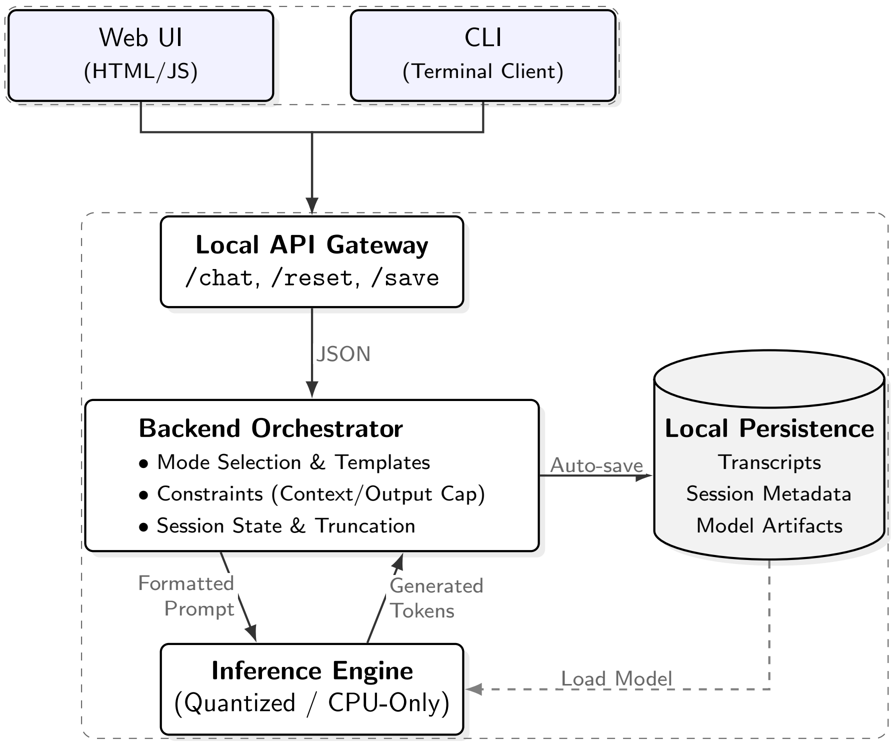

Figure 1: Pocket Brain technology-agnostic architecture, highlighting the separation between orchestration policy and raw inference.

4 How "Memory" Works in Conversational LLMs (Key Concept)

Large Language Models are **stateless** at inference time. They do not store chat history internally between requests. Conversational "memory" is implemented by the application layer by **resending prior turns** (a bounded subset of the transcript) along with the new user message on every request.

Implication

Each new model call includes:

- System prompt (global behavior)
- Mode prompt (task framing)
- Selected conversation history (bounded window)
- Current user message

If the prompt exceeds the model's context window, the application must drop or compress older content. Therefore, conversation continuity is a **systems policy decision**, not a property of the model.

5 Resource Model: What Actually Consumes RAM

On CPU-only inference, RAM usage is dominated by two primary components:

1. **Model weights:** quantization reduces memory footprint substantially (e.g., 4-bit weights vs 16-bit).
2. **KV cache (context memory):** as context grows, the key/value cache grows, increasing working-set size and memory traffic.

Why Context Management Matters

Longer contexts expand the KV cache and increase memory bandwidth pressure. This tends to reduce tokens-per-second and increase latency on CPU systems, especially as working sets exceed CPU cache and spill into main memory. Pocket Brain therefore enforces **bounded context growth** and **bounded output length**. While transcripts may be persisted to disk for long-term storage, only a limited working set is ever re-sent to the model. This distinction between persistent storage and active inference context is critical to maintaining predictable latency and memory usage under $\leq$8GB RAM.

6 Context Management Policy (Pocket Brain Design)

This section formalizes the policy you described (mode constraints + last N turns + new chat rollover + auto-save).

6.1 Mode Scoping

Pocket Brain enforces **one mode per session** (chat), selected at session creation. Example modes:

- `perspectives`: generate alternative viewpoints and angles
- `risks`: identify weaknesses, failure modes, and unknowns
- `next_steps`: produce actionable next actions and priorities

Rationale: mode scoping stabilizes behavior and reduces prompt bloat by avoiding multi-mode instruction stacking.

6.2 Sliding Window History (Last N Exchanges)

For each new request, Pocket Brain includes only the most recent N user-assistant exchanges (e.g., $N = 10$). Older content is not sent to the model for subsequent turns. This bounded window provides conversational continuity while controlling KV cache growth.

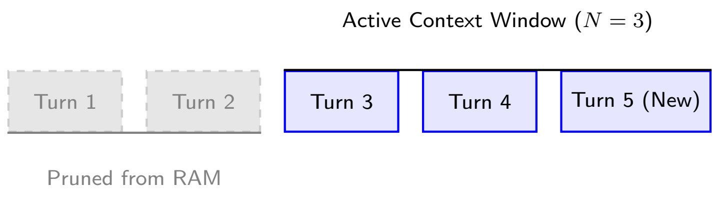

Figure 2: Visualizing the sliding window policy. Older turns are persisted to disk but dropped from the active inference context.

6.3 Explicit Session Rollover ("New Chat")

Pocket Brain supports a **New Chat** operation that:

- terminates the active session state,
- auto-saves the transcript to disk/SSD,
- creates a new session (optionally with a different mode).

This provides a simple and user-visible way to reset context and latency.

6.4 Optional Hard Stop (optional session rollover limit)

Some hosted systems stop a conversation after a usage threshold is reached. Pocket Brain does **not** require a hard stop for technical correctness; truncation is sufficient. However, an optional UX policy can be implemented:

- if repeated truncation is detected or the user reaches a configured turn limit,
- the UI suggests starting a new chat to preserve responsiveness,
- the system auto-saves and rolls over on user confirmation.

Parameter	Default (v1)	Rationale
Mode per session	Fixed	Keeps prompts stable and avoids instruction stacking.
History window	Last $N = 10$ exchanges	Bounds KV cache growth to ensure predictable latency.
Output cap	Configurable max tokens	Prevents runaway responses on CPU.
New chat behavior	Auto-save then reset	Provides user-visible rollover and clear state.
Hard stop	Off by default	Truncation is sufficient; hard stop is optional UX.

Table 1: Context management parameters for v1. Values may be tuned empirically once baseline performance is measured.

7 Model Candidates and Selection (v1)

Pocket Brain does not train or fine-tune models. Instead, it integrates existing open-weight, instruction-tuned language models that have been quantized for efficient CPU inference. Model selection is driven by strict memory and latency constraints rather than benchmark maximization.

7.1 Selection Criteria

Candidate models must satisfy the following:

- Availability in GGUF or equivalent quantized format
- Stable CPU-only inference under $\leq$8GB RAM
- Instruction-tuned behavior suitable for conversational use
- Acceptable latency when context and output lengths are bounded

7.2 Candidate Models

Pocket Brain v1 considers the following models as viable options:

- **Qwen2.5 3B Instruct**: A strong default candidate offering balanced reasoning quality and efficiency for its size. Suitable for structured brainstorming tasks and mode-driven prompts.
- **Phi-2 / Phi-3 Mini**: Smaller, faster models well-suited for short-turn interactions and rapid iteration. These models trade some depth for lower latency and reduced memory pressure.
- **Mistral 7B Instruct**: A higher-capacity option reserved for deeper reasoning profiles. Use requires aggressive context and output caps to remain within memory constraints.

Model	Size	Target	Strengths for v1	Role
Qwen2.5 Instruct	3B	Q4/Q5	Balanced quality/speed, strong instruction following.	**Default**
Phi-3 Mini	3.8B	Q4/Q5	Fast iteration, low memory pressure.	Alt-fast
Mistral Instruct	7B	Q4	Higher coherence, better complex reasoning.	Optional

Table 2: Candidate models for Pocket Brain v1. Quantization targets are selected to maintain CPU-only feasibility under $\leq$8GB RAM.

7.3 Model Usage Strategy

Pocket Brain v1 defaults to a **single-model strategy**, where interaction modes are implemented as prompt templates rather than model switching. This approach minimizes operational complexity and preserves consistent behavior across sessions.

Support for multiple models (e.g., a "fast" and "deep" profile) is considered a future extension and is not required for v1.

8 API-Level Feature Set (Driven by Clients)

The Web UI and CLI are thin clients; all core policies live server-side. The minimal API features required to support the memory and mode behavior above are:

- **Create session:** establish `session_id` and fixed `mode`
- **Chat:** submit a message, return assistant output
- **Reset / New chat:** clear session state and/or create a new session
- **Save:** persist transcript to disk/SSD
- **Health/status:** confirm model loaded and service operational

9 Technology Option Sets (Top 3 per Component)

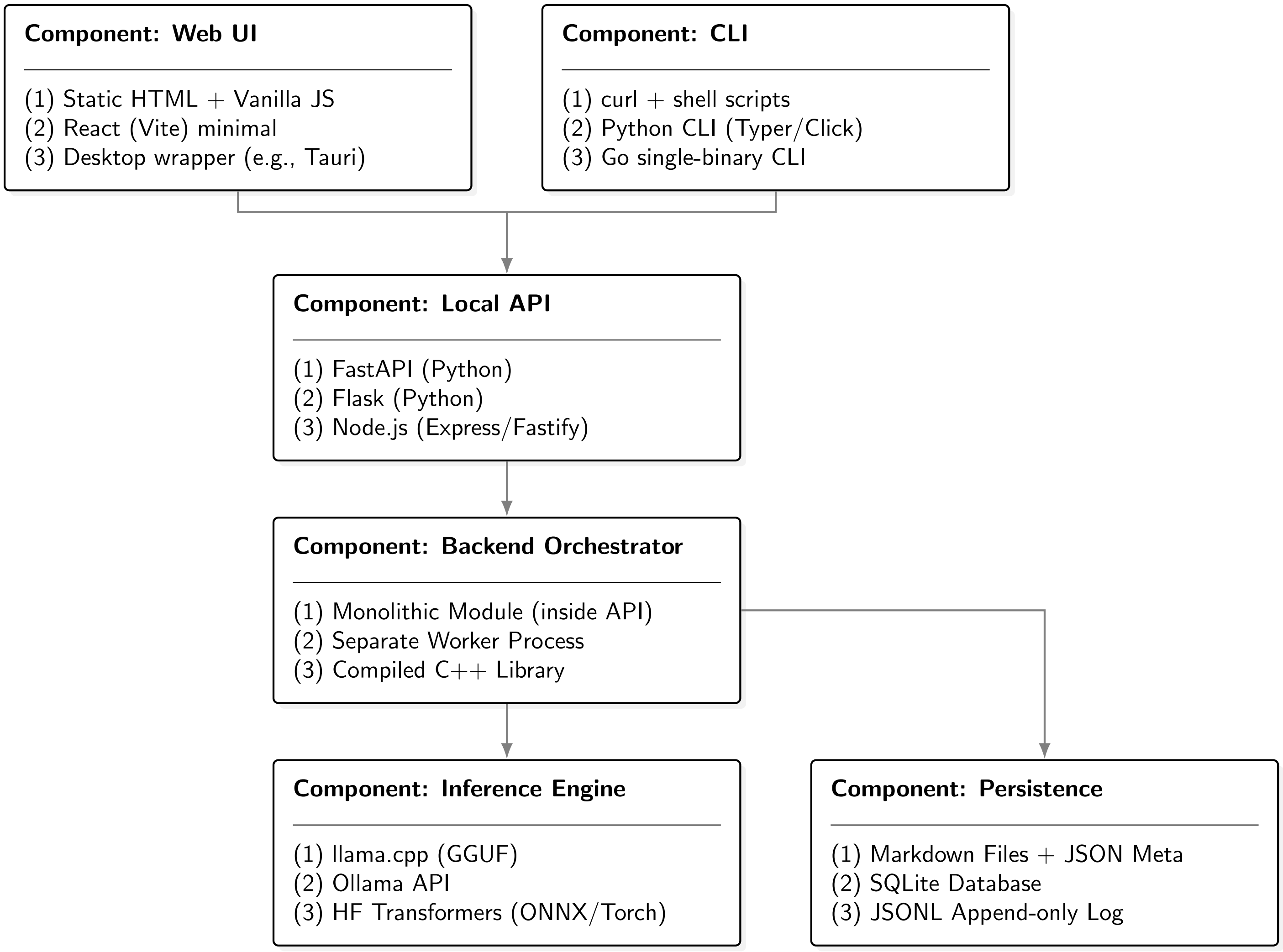
Component: Web UI
(1) Static HTML + Vanilla JS
(2) React (Vite) minimal
(3) Desktop wrapper (e.g., Tauri)
Component: CLI
(1) curl + shell scripts
(2) Python CLI (Typer/Click)
(3) Go single-binary CLI
Component: Local API
(1) FastAPI (Python)
(2) Flask (Python)
(3) Node.js (Express/Fastify)
Component: Backend Orchestrator
(1) Monolithic Module (inside API)
(2) Separate Worker Process
(3) Compiled C++ Library
Component: Inference Engine
(1) llama.cpp (GGUF)
(2) Ollama API
(3) HF Transformers (ONNX/Torch)
Component: Persistence
(1) Markdown Files + JSON Meta
(2) SQLite Database
(3) JSONL Append-only Log

Figure 3: Constrained technology option sets. The modular architecture allows swapping components (e.g., replacing Flask with FastAPI) without redesigning the system.

10 Deliverables and Milestones (Implementation Plan)

Milestone 1: End-to-End Skeleton

- API service running locally
- Session creation, chat request, and response
- Auto-save transcript to disk on demand

Milestone 2: Enforced Constraints

- Fixed mode per session
- Sliding window history (last N exchanges)
- Output caps (max tokens) and context caps

Milestone 3: Usability

- Web UI and CLI client parity
- "New Chat" rollover and clear chat controls
- Health/status endpoint + basic logging

Component	Option 1	Option 2	Option 3	**Select**
Web UI	Static HTML + JS	React (Vite)	Tauri wrapper	TBD
CLI	curl + shell	Python (Typer)	Go CLI	TBD
Local API	FastAPI	Flask	Node (Express)	TBD
Orchestrator	In-API Module	Worker process	C++ Lib	TBD
Engine	llama.cpp (GGUF)	Ollama	Torch/ONNX	TBD
Persistence	Markdown/JSON	SQLite	JSONL Log	TBD

Table 3: Technology decision matrix. The selection will be finalized based on v1 implementation constraints.

11 Versioning and Planned Evolution

Version 1 (Current Scope)

Pocket Brain v1 focuses on:

- text-only interaction,
- a single quantized model,
- fixed interaction modes per session,
- sliding window context truncation,
- local persistence of transcripts.

Potential Future Extensions

Future versions may explore:

- multi-profile inference (fast vs deep),
- improved launchers for Windows and macOS,
- optional import/export tooling,
- limited multimodal extensions.

These extensions are intentionally excluded from v1 to preserve clarity, stability, and bounded complexity.

12 Non-Goals

Pocket Brain explicitly excludes:

- model training or fine-tuning,
- long-document ingestion or RAG in v1,
- multi-user authentication and cloud deployment,
- GPU acceleration requirements.

13 Conclusion

Pocket Brain v1 demonstrates that a useful conversational assistant can be built under strict CPU-only and memory-constrained conditions by treating context management and interaction design as first-class systems problems. Rather than maximizing model size or feature count, the system emphasizes bounded resource usage, predictable latency, and structured prompting. This approach enables private, offline brainstorming workflows while remaining transparent, portable, and technically tractable.