AI Agents on Evernode

How LLMs Can Run Inside Evernode Instances Outside HotPocket Consensus

Scott Robert Chamberlain · May 2026

1. Summary

1.1 This paper explains how to run an LLM inside an Evernode instance by having a daemon run outside HotPocket consensus while still being able to talk to any contract running inside consensus via file-based IPC.

2. HotPocket Can Host Anything But Kills Long-Running Processes

2.1 Evernode is a DePIN that slices Linux machines into hosting slots that tenants can lease to run any POSIX compliant code inside docker containers. These leased slots are called *Instances*. What differentiates Evernode from other decentralised compute networks is that every instance comes with HotPocket, an out-of-the-box BFT consensus mechanism.

2.2 Via HotPocket consensus, each instance in a cluster runs the same code, sees the same inputs, and produces the same outputs. The cluster is the application; no single host is in charge. HotPocket transforms a cluster of Evernode instances into a layer 1 blockchain with its own shared canonical state across a cluster of independent hosts.

2.3 That model provides incredible smart contract flexibility:

(a) **Any language.** Evernode runs Docker containers, so an instance can host any code that can be containerised — JavaScript, Python, Rust, Go, C++, whatever the developer prefers. The talent pool is every developer, not a narrow blockchain-specific subset.

(b) **Any behaviour.** An Evernode dApp can do anything an ordinary application can do — maintain rich state, make external API calls, run heavy computation, serve full progressive web apps. Most blockchain "smart contract" platforms force you into a narrow execution model. Evernode does not.

(c) **Any geography.** With ~7,000 active hosts in 24+ countries means a developer can target hosts in specific jurisdictions, distribute clusters geographically for latency or regulatory reasons, or deploy globally and let the network sort it out.

(d) **Any scale.** A single-instance deploy works for solo applications; multi-instance clusters provide BFT consensus when you need it; cluster size scales with the resilience requirements of the application. You provision the topology that fits the job.

2.4 This flexibility includes the ability to run autonomous AI Agents, provided those Agents are deployed "outside consensus".

3. Why Agents Can't Be Run Inside Consensus

3.1 Evernode instances are highly flexible. Part of that flexibility allows the developer to choose which files to run "inside consensus" and which to keep "outside consensus". Inside consensus means the file is part of the contract the instance is going to run as a shared canonical state it maintains with all other instances running the same contract. Outside consensus means the files belong solely to the instance and not part of the shared canonical state with any other instance.

3.2 AI Agents can't run inside HotPocket consensus for two related reasons:

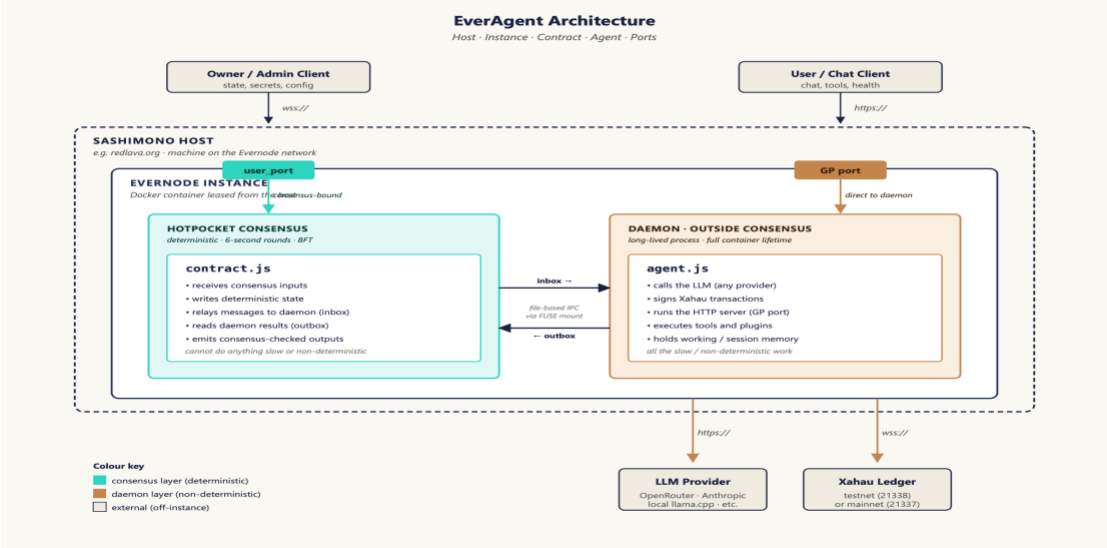
(a) **Nature of AI Agents**: Large language models are non-deterministic by design. The same prompt to the same model — even at temperature zero — does not reliably produce the same output token by token across runs. Three nodes running an identical prompt produce three different responses. Under HotPocket's rules, that's a consensus failure. The cluster's outputs don't match, the round fails silently, and the response never reaches the user.

(b) **Nature of HotPocket:** A HotPocket cluster runs the application code identically on every node. HotPocket starts contract.js fresh at the beginning of each consensus round and expects it to complete within the round window (about 6 seconds). An LLM call inside the contract would not finish in time — the round would fail, the contract would be re-instantiated for the next round, and the LLM call would start over from scratch. Run the LLM directly in the contract and you get nothing but endless failure and restarts.

3.3 So, if the Agent cannot live inside HotPocket, it needs to run outside consensus.

4. EverAgents: The Two-Process Model

4.1 We can use this inside/outside consensus structure to run LLMs inside Evernode instances without them being ritually killed off and restarted every six seconds.



4.2 The architecture splits the Agent between contract.js which has access to the user ports, agent.js which accesses a GP port for HTTP, and file-based IPC for the two code bases talk to each other:

(a) **contract.js** runs inside HotPocket consensus. It executes one round every six seconds, reads inputs from connected users, writes deterministic state, and produces deterministic outputs. It can do anything the consensus layer permits: counters, state transitions, message relay, configuration propagation. It cannot make a network request, cannot call an LLM, cannot do anything that takes more than a fraction of a second or that might produce a different answer on a different node.

(b) **The user port** uses WebSockets and goes through HotPocket consensus. Anything sent to it is processed by every node in the cluster, and every node has to produce the same response. It's the right port for deterministic operations: state queries, configuration injection, admin commands. Any output sent back through the user port is consensus-checked.

(c) **agent.js** is the daemon. It runs from the Docker image filesystem for the entire container lifetime — not gated by consensus rounds, not bound by determinism rules. It calls the language model. It signs Xahau transactions. It runs the agent's own HTTP server. It handles tool execution. It does all the work that the EverAgent actually exists to do.

(d) **The GP port** uses plain HTTP and goes directly to a single node's daemon. No consensus involved. Only the node you're talking to responds. It's the right port for non-deterministic operations: chatting with the agent, asking it to use a tool, running a health check. The GP port is how users actually interact with the EverAgent — every message in, every response out, goes through HTTP to that agent's daemon, bypassing consensus entirely.

(e) **File-based IPC -** local-data/inbox and local-data/outbox directories shared via the FUSE mount – allows the two file bases to talk to each other. The contract writes work orders to the inbox and reads responses from the outbox. The daemon polls the inbox for work, processes it, and writes results back to the outbox. The contract advances on its 6-second consensus cadence; the daemon runs at whatever speed the LLM and external services allow. Neither blocks the other.

4.3 Under this model, the contract is the protocol-facing process through which every consensus interaction passes, while the daemon is the agent-being-an-agent process: every LLM call, every tool execution, every Xahau signature happens there. Splitting them is what makes the entire architecture possible.

5. Why This Is The Right Shape

5.1 This split is a feature not a bug for several reasons:

(a) **Agent Standard:** First, every serious design for AI on decentralised compute will end up with this architecture: a lightweight daemon deployed on CPU compute with API access to a bigger brained GPU LLM as required. OpenClaw, Hermes – all are designed as lightweight harnesses calling a bigger brained LLM via API. As CPU Agents get more powerful, the daemon will be able to handle increasingly complex calls without recourse to the larger LLM, keeping cost at a minimum.

(b) **Consensus Preserved:** Second, the consensus layer is not weakened by having the AI call sit outside it. The consensus layer protects the things consensus is good at: state, agreements, deterministic logic. The daemon protects the agent from being neutered by determinism rules: the agent retains discretion, can use the best models, can call external tools. Each layer does what it's good at.

(c) **Real Autonomy:** Finally, the split enables real autonomy. With the daemon-outside-consensus pattern in place, an EverAgent can hold a long-running conversation with a user via HTTP, maintain its own working state in memory across that conversation, call any LLM API (OpenAI, Anthropic, OpenRouter, local llama.cpp — anything that responds to HTTP), sign and submit Xahau transactions on its own behalf, use tools (invoke shell commands, read files, query APIs, anything the agent's plugins expose), and persist state to disk so it survives daemon restarts.

6. EverAgents as Smart Contract Nodes

6.1 With this architecture you can think of the daemon as something like an astronaut roaming cyberspace from inside its instance spaceship, with the contract.js being its empty cargo hold. The Agent can persist inside the spaceship without a contract and do whatever tasks it is set via HTTP.

6.2 What is unique about an EverAgent versus other iterations, like OpenClaw, is that one of the tasks an EverAgent can do is to pick up a contract and join a cluster, almost like a contractor picking up a job.

6.3 I don't believe any other Agent implementation lets its Agent participate as a node in running a smart contract, potentially for profit. This has interesting possibilities when it comes to clients commissioning clusters for their evernode contracts.

7. Future work

7.1 Having solved the problem of getting LLMs to run on Evernode instances, the next problem to solve is how the Agents identify themselves and other Agents.

Adventures in AI Consensus #1. Lex Automagica. May 2026.
Scott Robert Chamberlain.